1. INTRODUCTION

Gesture recognition is a topic in language technology with the goal of interpreting human gestures via mathematical algorithms. Gesture recognition has wide-ranging applications such as developing aids for the hearing impaired; designing techniques for forensic identification; recognizing sign language etc. Here we are implementing Hand gesture recognition in MATLAB by the process of capturing image using image acquisition toolbox. Skin detection algorithm is implemented on the captured images and then further gesture orientations are decided using the different algorithms (Preprocessing and Feature Extraction algorithms like Segmentation, Edge Detection, Filtration, etc). The results are optimized using NNTool in MATLAB. Here Artificial Neural Network plays its role as the NNTool works on the Artificial Neural Networks.

1.1 AIM:

The aim of this project is Gesture recognition in image processing using Artificial Neural Network by using

1. Skin detection algorithm followed by the geometrical method approach.
2. Image direction detection and classifying the images with respect to the target images.
3. Optimization of result using NNTool.

1.2 OBJECTIVE:

MATLAB is one of the tools used for gesture recognition. In previous semester our project was focused on gesture recognition without skin detection using four different target images orientations (0 degree, 90 degree, 180 degree and 270 degree). New algorithm was proposed for the different input samples to decide the orientation of input image. And the result was optimized using NNTool.

In this semester our main objectives are real time acquisition of hand gesture and skin detection of the captured images using eight different target images orientations (0 degree,

45 degree, 90 degree, 135 degree, 180 degree, 225 degree, 270 degree, 315 degree) then proposing algorithm to decide the orientation and further optimizing using NNTool.

1.3 APPROACH

The approach of this project is started with interacting the camera in MATLAB with image acquisition toolbox and thus the image is captured and stored in MATLAB for further processing and then we have used skin detection algorithm on the input and target images. Shape and position information about the hand will be gathered using the skin detection method which differentiates the skin pixels from the non skin pixels. After skin detection the images are thus passed through the preprocessing phase (segmentation, edge detection, noise removal etc). Further feature extraction is done on all the images. Feature extraction is done by using the centroid method for determining the centroid of the captured images. And the centroid thus converted into polar coordinates which values are then processed. Here the input values are compared with the predefined target values for the further result. And then the theta value of input sample is given as the input to the NNTool for applying the Neural Network Technique and theta value of target sample is given as target value in the NNTool and then result is optimized.

2. **PROECT BACKGROUND**

A gesture is considered as a natural way of communication among human, since it is a physical movement of hands, arms, or body which convey meaningful information. Gesture recognition then, is the interpretation of that movement as semantically meaningful command. Gesture recognition has been studied in wide descriptive topics, and has a wide range of applications such as recognizing of sign language, human computer interaction (HCI), robot control, machine vision, smart surveillance, visual environmental manipulating, etc. Skin detection also plays important role in today's scenario as there are various applications using the skin detection algorithm. One of the major uses of skin detection is in the face recognition systems.

2.1 ARTIFICIAL NEURAL NETWORK [1]

Artificial neural network is the most widely and commonly used approach for hand gesture recognition. Artificial neural network consists of one input layer, one output layer and one hidden layer, making a total of three, with each layer fully connected. An Artificial Neural Network is an information processing paradigm that is inspired by the way the biological nervous systems, (such as the brain) processes the information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements called neurons, working in unison to solve the specific problems.

An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

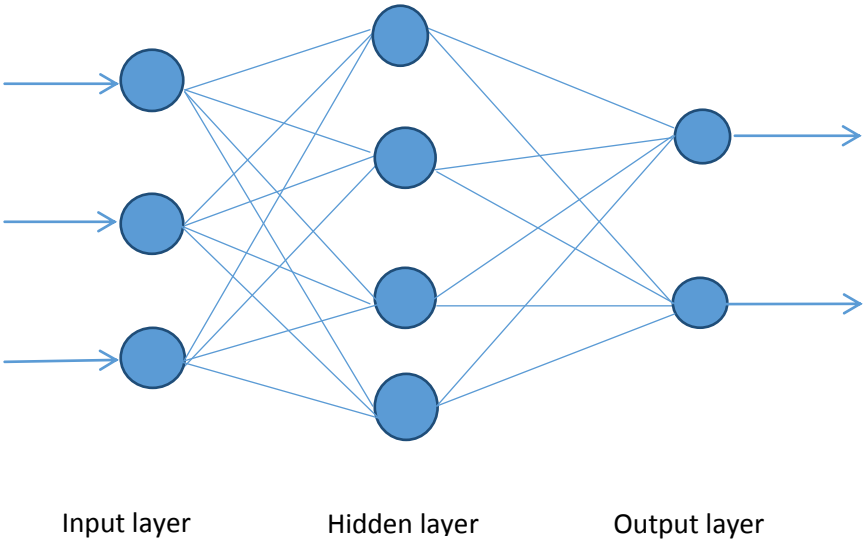


Figure 2.1.1: Artificial Neural network

In the above figure 2.1 there are three different layers input layer, hidden layer, and output layer. Advantages of artificial neural networks include their high tolerance to noisy data as well as their classify patterns on which they have not been trained. The training algorithm for this Artificial Neural Networks is classified in the three classes: gradient descent, quasi-Newton (Levenberg–Marquardt, LM) and genetic algorithm (GA). Two versions of gradient descent back-propagation algorithms are incremental back-propagation (IBP) and batch back-propagation (BBP).

Neural networks are algorithms for optimization and learning, based loosely on concepts inspired by research into the nature of the brain. They generally consist of five components:

1. A directed graph known as the network topology whose arcs we refer to as links.
2. A state variable associated with each node.
3. A real-valued weight associated with each link.
4. A real-valued bias associated with each node.
5. A transfer function for each node which determines the state of a node as a function of

a) Its bias b,

b) The weights, wt of its incoming links

c) The states, x, of the nodes connected to it by these links.

A feed forward network is one whose topology has no closed paths. Its input nodes are the ones with no arcs to them, and its output nodes have no arcs away from them. All other nodes are hidden nodes. When the states of all the input nodes are set, all the other nodes in the network can also set their states as the values propagate through the network. The operation of a feed forward network consists of calculating the outputs given as a set of inputs in this manner. A layered feed forward network is given such that any path from an input node to an output node traverses the same number of arcs. The nth layer of such a network consists of all nodes which are n arc traversals from an input node. A hidden layer is one which contains hidden nodes. Such a network is fully connected if each node in layer I is connected to all nodes in layer i+l or all i. Layered feed forward networks have become very popular for a few reasons. For one, they have been found in practice to generalize well, i.e. when trained on a relatively sparse set of data points; they will often provide the right output for an input not in the training set. Secondly, a training algorithm called back-propagation exists which can often find a good set of weights in a reasonable amount of time [1].

2.2 BACK PROPAGATION ALGORITHM

Back-propagation is a variation on gradient search. It generally uses a least-squares optimality criterion. The key to back-propagation is a method for calculating the gradient of the error with respect to the weights for a given input by propagating error backwards through the network. There are some drawbacks to back-propagation.

For one, there is the "scaling problem". Back-propagation works well on simple training problems. However, as the problem complexity increases (due to increased dimensionality and/or greater complexity of the data), the performance of back-propagation falls off rapidly. The performance degradation appears to generate from the fact that the complex spaces have nearly global minima which are sparse among the local minima. Gradient search techniques tend to get trapped at local minima. With a high enough gain (or momentum), back propagation can escape these local minima. However, it leaves them without knowing whether the next one it finds will be better or worse. When the nearly global minima are well hidden among the local minima, back propagation can end up bouncing between local minima without much overall improvement, thus making for very slow training.[1][2] `

A second shortcoming of back propagation is the differentiability required to compute gradient. Therefore, back-propagation cannot handle discontinuous optimality criteria or discontinuous node transfer functions. This precludes its use on some common node types and simple optimality criteria.

2.3 WORK FLOW FOR NEURAL NETWORK [NNTOOL] [14]

The work flow for the neural network design process has seven primary steps:

1 Collect data

2 Create the network

3 Configure the network

4 Initialize the weights and biases

5 Train the network

6 Validate the network

7 Use the network

Data collection in step 1 generally occurs outside the framework of Neural Network Toolbox software. The Neural Network Toolbox software uses the network object to store all of the information that defines a neural network. After a neural network has been created, it needs to be configured and then trained. Configuration involves arranging the network so that it is compatible with the problem we want to solve, as defined by sample data. After the network has been configured, the adjustable network parameters (called weights and biases) need to be tuned, so that the network performance is optimized. This tuning process is referred to as training the network. Configuration and training require that the network be provided with example data.

There are four different levels at which the Neural Network Toolbox software can be used. The first level is represented by the GUIs that are described in "Getting Started with Neural Network Toolbox". These provide a quick way to access the power of the toolbox for many problems of function fitting, pattern recognition, clustering and time series analysis. The second level of toolbox use is through basic command-line operations. The command-line functions use simple argument lists with intelligent default settings for function parameters. This topic, and the ones that follow, concentrate on command-line operations. The GUIs described in Getting Started can automatically generate MATLAB code files with the command-line implementation of the GUI operations. A third level of toolbox use is customization of the toolbox. This advanced capability allows us to create our own custom neural networks, while still having access to the full functionality of the toolbox. The fourth level of toolbox usage is the ability to modify any of the M-files contained in the toolbox.

2.4 ADVANTAGE OF NEURAL NETWORK COMPUTING [17]

There are a variety of benefits that an analyst realizes from using neural networks in their work.

1. Pattern recognition is a powerful technique for harnessing the information in the data and generalizing about it. Neural nets learn to recognize the patterns which exist in the data set.

2. The system is developed through learning rather than programming. Programming is much more time consuming for the analyst and requires the analyst to specify the exact behavior of the model. Neural nets teach themselves the patterns in the data freeing the analyst for more interesting work.

3. Neural networks are flexible in a changing environment. Rule based systems or programmed systems are limited to the situation for which they were designed--when conditions change, they are no longer valid. Although neural networks may take some time to learn a sudden drastic change, they are excellent at adapting to constantly changing information.

4. Neural networks can build informative models where more conventional approaches fail. Because neural networks can handle very complex interactions they can easily model data which is too difficult to model with traditional approaches such as inferential statistics or programming logic.

5. Performance of neural networks is at least as good as classical statistical modeling, and better on most problems. The neural networks build models that are more reflective of the structure of the data in significantly less time.

6. Neural networks now operate well with modest computer hardware. Although neural networks are computationally intensive, the routines have been optimized to the point that they can now run in reasonable time on personal computers. They do not require supercomputers as they did in the early days of neural network research.

2.5 LIMITATION OF NEURAL NETWORK COMPUTING [17]

There are some limitations to neural computing.

1. The key limitation is the neural network's inability to explain the model it has built in a useful way. Analysts often want to know why the model is behaving as it is. Neural networks get better answers but they have a hard time explaining how they got there.
2. It is difficult to extract rules from neural networks. This is sometimes important to people who have to explain their answer to others and to people who have been involved with artificial intelligence, particularly expert systems which are rule-based. As with most analytical methods, we cannot just throw data at a neural net and get a good answer. We have to spend time understanding the problem or the outcome we are trying to predict. And, we must be sure that the data used to train the system are appropriate and are measured in a way that reflects the behavior of the factors. If the data are not representative of the problem, neural computing will not product good results. This is a classic situation where "garbage in" will certainly produce "garbage out."
3. It can take time to train a model from a very complex data set. Neural techniques are computer intensive and will be slow on low end PCs or machines without math-coprocessors. It is important to remember though that the overall time to results can still be faster than other data analysis approaches, even when the system takes longer to train. Processing speed alone is not the only factor in performance and neural networks do not require the time programming and debugging or testing assumptions that other analytical approaches do.

3. SYSTEM SPECIFICATIONS

3.1 MATLAB

The project involves the working under MATLAB Software. The Image Acquisition tool is used for interfacing the web camera with the software and thus for further processing on the images captured by the web camera.

The MATLB version used in this project is 7.0.1.

3.2 NNTooL [14]

In MATLAB Neural Network Tool is used for generating the Artificial Neural Network used as the optimizing network for the gestures. The Neural Network Toolbox software uses the network object to store all of the information that defines a neural network.

The Neural Network Design textbook includes:

• An Instructor's Manual for those who adopt the book for a class

• Transparency Masters for class use

1. Create Neural Network Object

The easiest way to create a neural network is to use one of the network creation functions. To investigate how this is done, we can create a simple, two-layer feed forward network, using the command feed forward net:

net = feed forward net

This command displays the following:

net =Neural Network

name: 'Feed-Forward Neural Network'

user data: IMAGES

2. Weight and Bias values:

This display is an overview of the network object, which is used to store all of the information that defines a neural network. There is a lot of detail here, but there are a few key sections that can help you to see how the network object is organized.

The dimension section stores the overall structure of the network. Here we can see that there is one input to the network (although the one input can be a vector containing many elements), one network output and two layers. The connection section stores the connections between components of the network. For example, here there is a bias connected to each layer, the input is connected to layer 1, and the output comes from layer 2. We can also see that layer 1 is connected to layer 2. (The rows of net layer Connect represent the destination layer, and the columns represent the source layer. One in this matrix indicates a connection, and zero indicates a lack of connection. For this example, there is a single one in the 21 element of the matrix.) The key sub objects of the network object are inputs, layers, outputs, biases, input Weights and layer Weights. View the layer's sub-object for the first layer with the command net layer 1.

The weight function represents standard matrix multiplication (dot product).The size of this layer weight should be 0 by 20. The reason that we have zero rows is because the network has not yet been configured for a particular data set. The number of output neurons is determined by the number of elements in our target vector. During the configuration process, we will provide the network with example inputs and targets, and then the number of output neurons can be assigned.

3.3 CAMERA

The web camera used for this process is having specifications as, 25-30 frames per second and still image at 1.3 MP. Here we have used webcam in windows and the features of the webcam are:

InstalledAdaptors: {'coreco' 'winvideo'}

MATLABVersion: '7.10 (R2010a)'

ToolboxName: 'Image Acquisition Toolbox'

ToolboxVersion: '3.5 (R2010a)'

AdaptorDllName: [1x87 char]

AdaptorDllVersion: '3.5 (R2010a)'

AdaptorName: 'winvideo'

DeviceName: 'USB2.0 Camera'

4. PROPOSED SYSTEM IMPLEMENTATION

4.1 PROPOSED SYSTEM PROCEDURE

Many researchers have been suggested on gesture recognition system for different applications, with different recognition phases but they all agree with the main structure of the gesture recognition system. These are carried in following steps;

1. Image Acquisition
2. Skin Detection
    - HSV Evaluation
    - YCbCr Evaluation
    - RGB Evaluation
3. Pre-processing
    - Segmentation
    - Background Removal
    - Noise Removal
4. Feature Extraction
    - Centroid Method
5. Classification

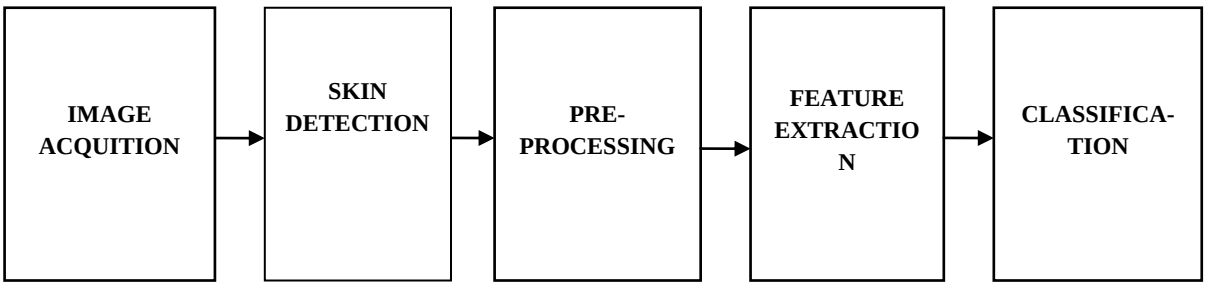


FIG 4.1: BLOCK DIAGRAM FOR PROPOSED GESTURE RECOGNITIONSYSTEM

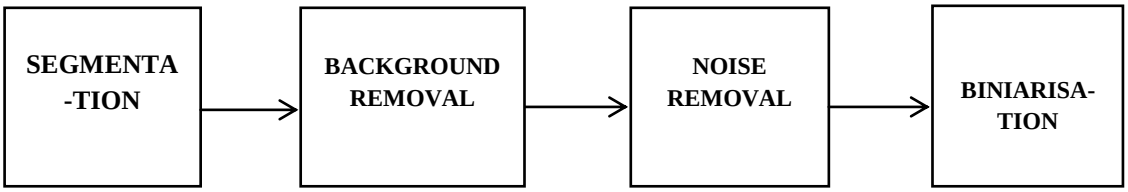


FIG 4.1.2: BLOCK DIAGRAM OF PRE-PROCESSING

Object recognition consists of locating the positions and possibly orientations and scales of instances of objects in an image. Images that are acquiescing from video sequence, passed through the preprocessing phase to obtain enrich data for feature extraction.

Preprocessing of the image is a very important part of the gesture recognition as the image captured is needed to be made suitable for further process i.e., feature extraction. Pre-processing include segmentation, edge detection and noise removal of segmented image.

The features are the useful information that can be extracted from the segmented hand object by which the machine can understand the meaning of that posture. The numerical representation of these features can be obtained from the vision perspective of the segmented hand object, which forms the feature extraction phase. The features extracted are thus classified to distinguish them in order to get the gestures recognized by the system for further use, under various applications.

4.2 PROPOSED SYSTEM IMPLEMENTATION FLOW CHART

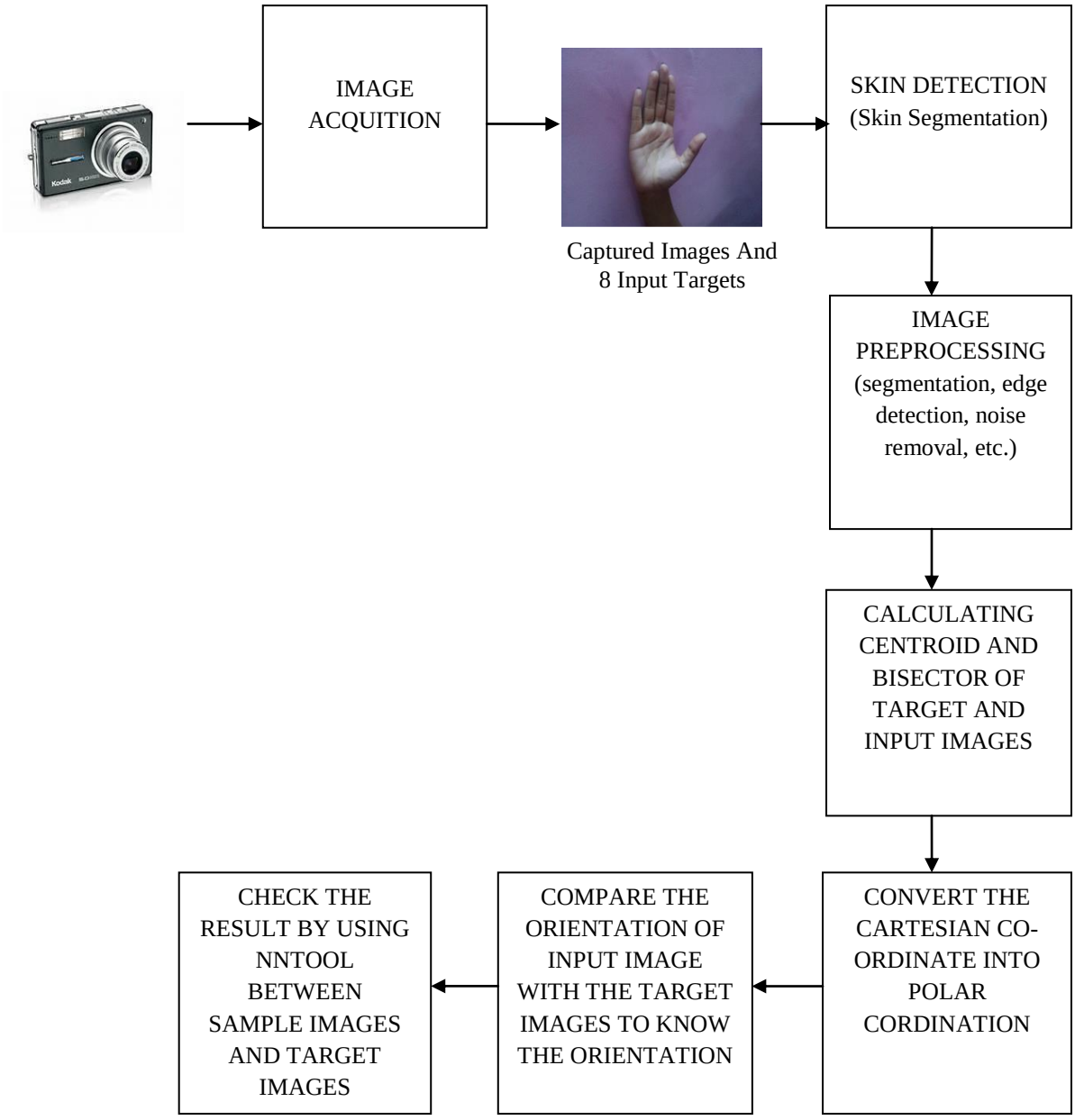


FIG 4.3 PROPOSED SYSYTEM FLOW CHART

4.3 PROPOSED SYSYTEM FLOW CHART STEPS

A. Video Capturing and Image Acquisition

Image Acquisition is initial level to start the process of feature extraction. Here we have to select captured video from camera as input. We are now ready to start extracting frames from the videos. The MATLAB software is the basic tool used in this process. The image acquisition tool in MATLAB is used here for taking images as input to the system.

B. Skin Detection [16]

Skin detection process has two phases: a training phase and a detection phase. Training a skin detector involves three basic steps:

1. Collecting a database of skin patches from different images. Such a database typically contains skin-colored patches from a variety of people under different illumination conditions.
2. Choosing a suitable color space.
3. Learning the parameters of a skin classifier.

Given a trained skin detector, identifying skin pixels in a given image or video frame involves:

1. Converting the image into the same color space that was used in the training phase.
2. Classifying each pixel using the skin classifier to either a skin or non-skin.
3. Typically post processing is needed using morphology to impose spatial homogeneity on the detected regions.

In any given color space, skin color occupies a part of such a space, which might be a compact or large region in the space. Such region is usually called the skin color cluster. A skin classifier is a one-class or two-class classification problem. A given pixel is classified and labeled whether it is a skin or a non-skin given a model of the skin color cluster in a

given color space. In the context of skin classification, true positives are skin pixels that the classifier correctly labels as skin. True negatives are non-skin pixels that the classifier correctly labels as non-skin. Any classifier makes errors: it can wrongly label a non-skin pixel as skin or a skin pixel as a non-skin. The former type of errors is referred to as false positives (false detections) while the later is false negatives. A good classifier should have low false positive and false negative rates. As in any classification problem, there is a tradeoff between false positives and false negatives. The more loose the class boundary, the less the false negatives and the more the false positives. The tighter the class boundary, the more the false negatives and the less the false positives. The same applies to skin detection. This makes the choice of the color space extremely important in skin detection. The color needs to be represented in a color space where the skin class is most compact in order to be able to tightly model the skin class. The choice of the color space directly affects the kind of classifier that should be used.

C. Image Preprocessing

Image preprocessing consists of segmentation, edge detection, noise removal as shown in the figure 3.2.

C.1 Hand region segmentation

The initial step of hand gesture recognition is the detection of hand region from the background. This step is also known as hand detection. It involves detecting and extracting hand region from background and segmentation of hand image. Previous methods made use of following two approaches that is the color based model and statistical based model. Different features such as skin color, shape, motion and anatomical models of hand are used in different methods. The output of this step is a binary image in which skin pixels have value 1 and non-skin pixels have value 0.

C.2 Edge detection [9]

In an image, an edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image.

To find edges, you can use the edge function. This function looks for places in the image where the intensity changes rapidly, using one of these two criteria:

- Places where the first derivative of the intensity is larger in magnitude than some threshold
- Places where the second derivative of the intensity has a zero crossing

edge provides a number of derivative estimators, each of which implements one of the definitions above. For some of these estimators, you can specify whether the operation should be sensitive to horizontal edges, vertical edges, or both. edge returns a binary image containing 1's where edges are found and 0's elsewhere.

The most powerful edge-detection method that edge provides is the Canny method. The Canny method differs from the other edge-detection methods in that it uses two different thresholds (to detect strong and weak edges), and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

C.3 Image enhancement and Remove noise [14]

Noise reduction is the process of removing noise from a signal. Noise reduction techniques are conceptually very similar regardless of the signal being processed, however a priori knowledge of the characteristics of an expected signal can mean the implementations of these techniques vary greatly depending on the type of signal. The median filter is a

nonlinear digital filtering technique, often used to remove noise. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image). Median filtering is very widely used in digital image processing because under certain conditions, it preserves edges while removing noise.

D. Feature extraction and gesture recognition

The next important step is hand tracking and feature extraction. Tracking means finding frame to frame correspondence of the segmented hand image to understand the hand movement. This is done by using centroid method i.e detecting the centroid of the input images and then comparing with the target images.

5. SYSTEM ANALYSIS AND DESIGN

We have started the system analysis by capturing images in the MATLAB using image acquisition toolbox. We have used MATLAB (7.0.1) and camera (USB 2.0a). Image of eight different orientations and some input samples are captured (0 degree, 45 degree, 90 degree, 135 degree, 180 degree, 225 degree, 270 degree, 315 degree) as given in the chapter number 6. The steps of algorithm are implemented on the images as shown in the figure number 3.3. These steps are skin detection followed by images acquisition. Shape and position information about the hand will be gathered using the skin detection method which differentiates the skin pixels from the non skin pixels. Pre processing is the steps consist of segmentation, edge detection and noise removal. Segmentation is the process of converting skin pixels to 1 and rest all non- skin to 0. This is implemented using ostu algorithm. An edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image. For this we have used canny and sobel edges detection method [9]. Followed by noise removal using med2filter. After pre-processing it will undergo feature extraction process. Feature extraction process uses geometrical approach and non geometrical approach for extracting features. Here we are using centroid method. In centroid method the centroid of all the eight target images and some input samples are calculated. Then the centroid of input sample of random orientation is compared with the target images to decide the orientation. The result is further optimized using NNTool in MATLAB. In this we have given the theta value of centroid of the input sample as input and theta value of centroid of target images as the target the network is train. After training the error will decide the exact orientation.

6. RESULT

6.1 TARGET IMAGES CAPTURED BY CAMERA

We have captured the eight different orientation images of hand rotation, using image acquisition tool and web camera. These different types of captured images are interacted with the MATLAB system, as the image acquisition tool act as the interface to the web camera and the MATLAB. The eight different orientations as given under are taken as the target and further skin detection process followed by pre-processing (segmentation, edge detection, noise removal etc) are implemented on them. And lastly we have used geometrical approach of feature extraction i.e centroid method.

- Orientation 1: Orientation rightwards(0 degree)

Fig.6.1.1

- Orientation 45 degree

Fig.6.1.2

- Orientation upwards(90 degree)

Fig6.1.3

- Orientation 135 degree

Fig.6.1.4

- Orientation leftward(180 degree)

Fig.6.1.5

- Orientation 225 degree

Fig.6.1.6

- Orientation downwards(270 degree)

Fig.6.1.7

- Orientation 315 degree

Fig.6.1.8

6.2 INPUT SAMPLE IMAGE

Input images are captured through the the camera and these captured images are used by Matlab through the image acquisition toolbox and the process of this is given in the chapter number 5.3.

Input 1

Fig.6.2.1

Input 2

Fig.6.2.2

6.3 PROPOSED STEPS IMPLEMENTATION FOR THE TARGET IMAGES

6.3 .1 IMAGE ACQUISITION

Image acquisition is done by capturing different hand movements at different angles using cameras. These captured images are the data and pixels of these images are noted. This is done by image acquisition toolbox.

Image Acquisition Toolbox enables us to acquire images and video from cameras directly into MATLAB. We can detect hardware automatically and configure hardware properties. Image Acquisition Toolbox provides an application and functions and a programmatic interface to help us work with image acquisition hardware in MATLAB.

6.3.1.1 CONNECTING TO HARDWARE

Image Acquisition Toolbox automatically detects compatible image and video acquisition devices. Each device connection is encapsulated as an object, providing an interface for configuration and acquisition. We can create multiple connection objects for simultaneous acquisition from as many devices as our PC and imaging hardware support. Image Acquisition Toolbox is used on Windows and it be used on Linux, and Macintosh systems, enabling us to reuse code when connecting to the same camera in different operating systems.

6.3.1.2 ACQUIRING IMAGE DATA

Image Acquisition Toolbox supports several modes, including background acquisition and continuous acquisition, while processing the acquired data. The image acquisition engine is designed to acquire imagery as fast as our camera and computer can support, enabling analysis and processing of high-speed imaging applications. Data can be acquired in a wide range of data types, including signed or unsigned 8-, 16-, and 32-bit integers and

single- or double-precision floating point. The toolbox supports any color space provided by the image acquisition device including RGB, YUV, or gray scale. Raw sensor data in a Bayer pattern can be automatically converted into RGB data.

Fig.6.3.1

6.3.2 SKIN DETECTION

The skin detection algorithim is implemented on the image acquired by the image acquisition toolbox. The goal of skin color segmentation is to reject non-skin color regions from the input image. It is based on the fact that the color of the human face across all races agrees closely in its chrominance value and varies mainly in its luminance value.

We chose the HSV (Hue, Saturation, Value) color space for segmentation since it decouples the chrominance information from the luminance information. Thus we can only focus on the hue and the saturation component. The faces in each training image were extracted using the ground truth data and a histogram was plotted for their H and S color component (Figure 2). The histograms reveal that the H and S color components for faces are nicely clustered. This information was used to define appropriate thresholds for H and S space that correspond to faces. The threshold values were embedded into the color segmentation routine.

During the execution of the detector, segmentation is performed as follows:

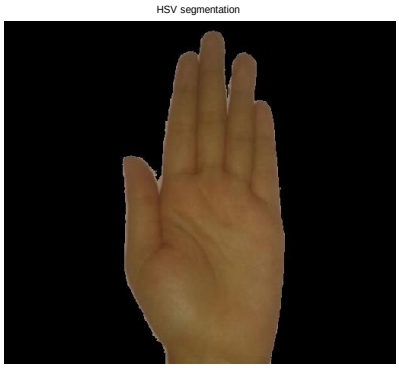
1. The input image is sub-sampled at 2:1 to improve computational efficiency

2. The resulting image is converted to HSV color space

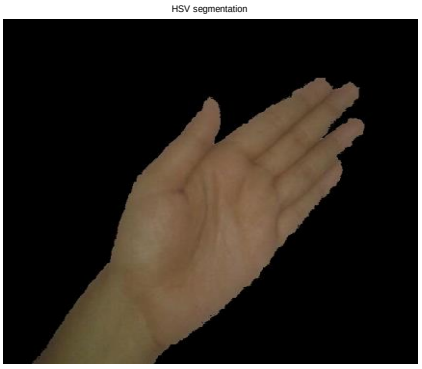
3. All pixels that fall outside the H and S thresholds are rejected (marked black).

The output for the skin segmentation is given below. Here the result shown gives the HSV segmentation of the image.

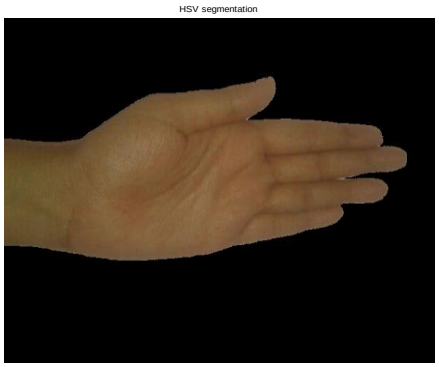
a.



b.

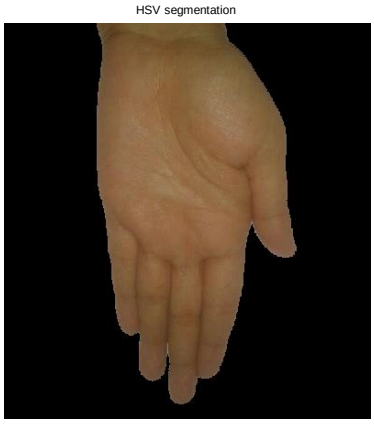


c.

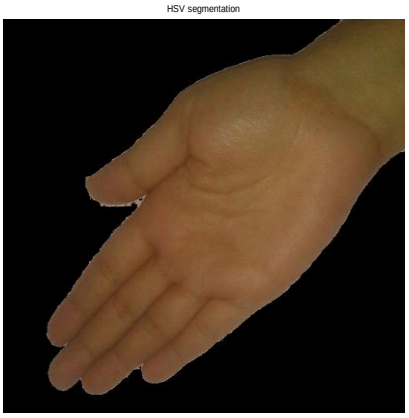


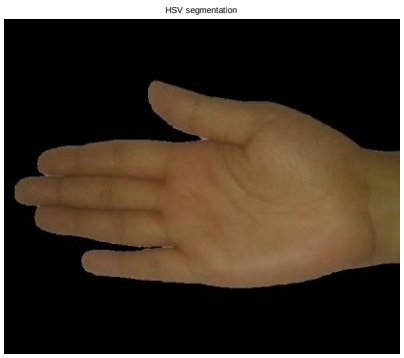
d.

e.

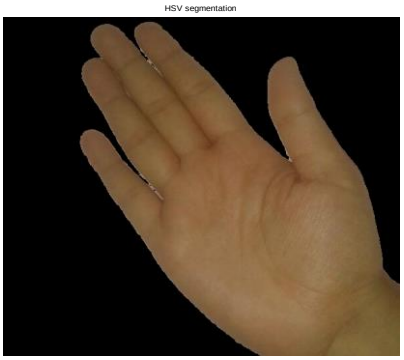


f.





g.



h.

Fig.6.3.2.1Skin detection for 8 target images

6.3.3 PREPROCESSING (SEGMENTATION)

The skin segmented image is processed through different other segmentation algorithm to get more and more information related to the images. Image segmentation is the process of dividing an image into multiple parts. This is typically used to identify objects or other relevant information in digital images. There are many different ways to perform image segmentation, including:

1.1 Threshold methods such as Otsu's method

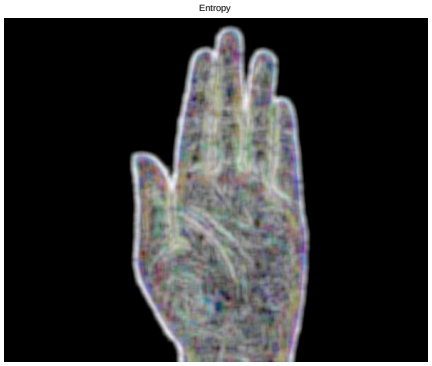
1.2 Color-based Segmentation such as K-means clustering

1.3 Transform methods such as watershed segmentation

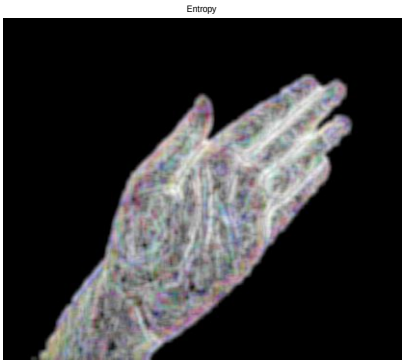
1.4 Texture methods such as texture filters

An effective approach to performing image segmentation includes using algorithms, tools, and a comprehensive environment for data analysis, visualization, and algorithm development. Here we are using Otsu's method and we are going to explain this method.

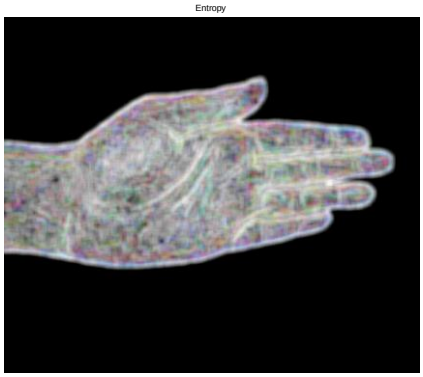
The output images generated during the steps involved:



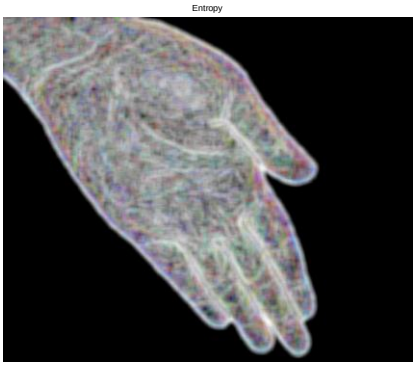
a.



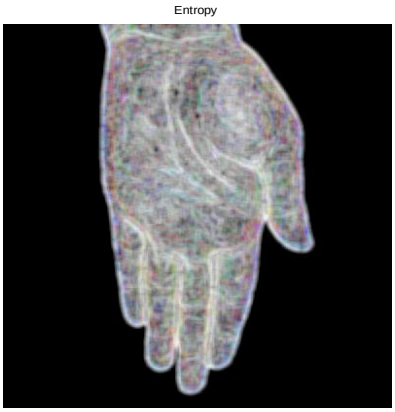
b.



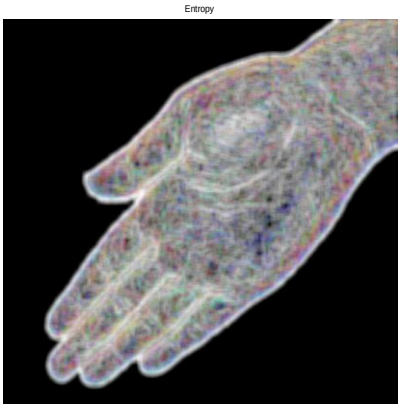
c.



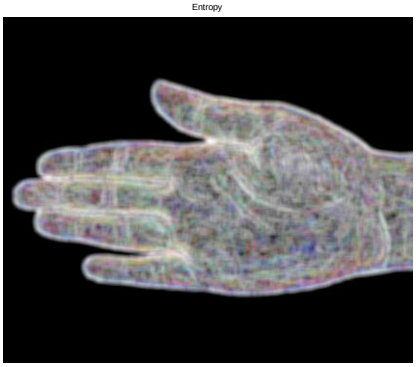
d.



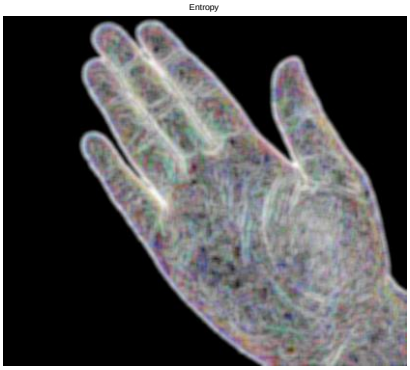
e.



f.

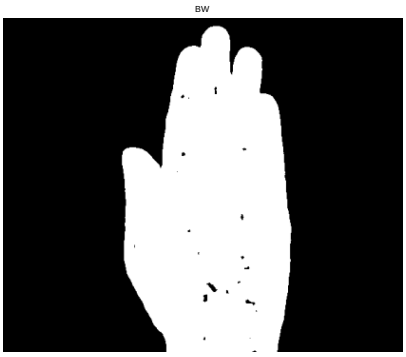


g.



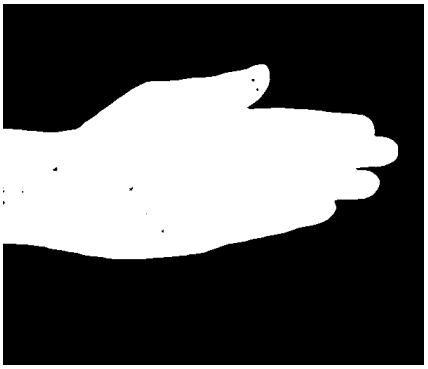
h.

Fig.6.3.3.1 The images shows the Entropy of the captured images



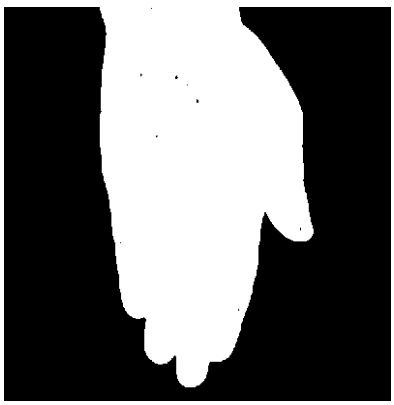
a.

b.



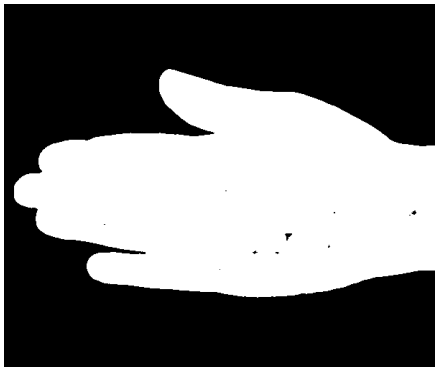
c.

d.



e.

f.



g.

h.

Fig.6.3.3.2 The images shows the RGB to Gray converted images

6.3.4 PREPROCESSING (EDGE DETECTION)

In an image, an edge is a curve that follows a path of rapid change in image intensity. Edges are often associated with the boundaries of objects in a scene. Edge detection is used to identify the edges in an image.

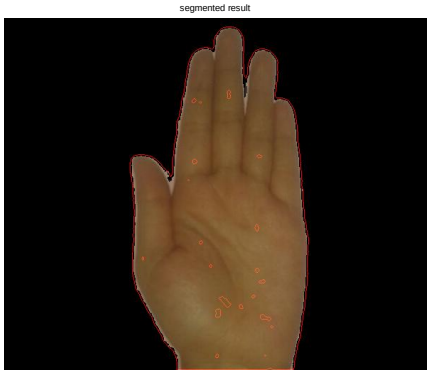
To find edges, you can use the edge function. This function looks for places in the image where the intensity changes rapidly, using one of these two criteria:

- Places where the first derivative of the intensity is larger in magnitude than some threshold
- Places where the second derivative of the intensity has a zero crossing

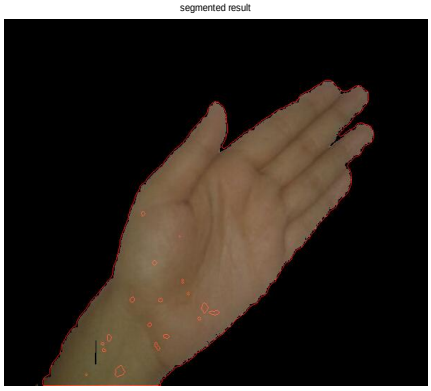
edge provides a number of derivative estimators, each of which implements one of the definitions above. For some of these estimators, you can specify whether the operation should be sensitive to horizontal edges, vertical edges, or both. edge returns a binary image containing 1's where edges are found and 0's elsewhere.

The most powerful edge-detection method that edge provides is the Canny method. The Canny method differs from the other edge-detection methods in that it uses two different thresholds (to detect strong and weak edges), and includes the weak edges in the output only if they are connected to strong edges. This method is therefore less likely than the others to be fooled by noise, and more likely to detect true weak edges.

a.

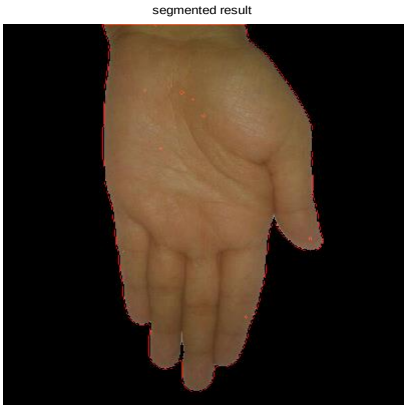


b.

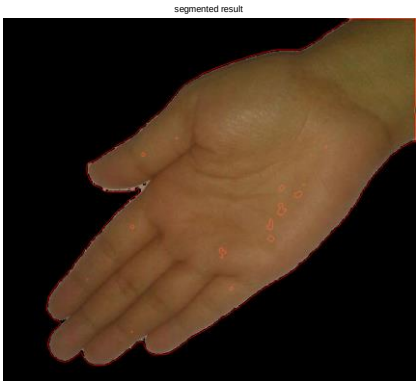


c.

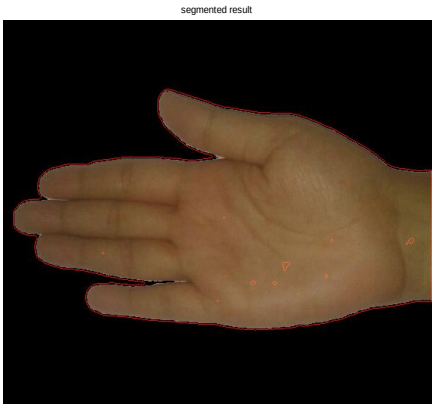
d.



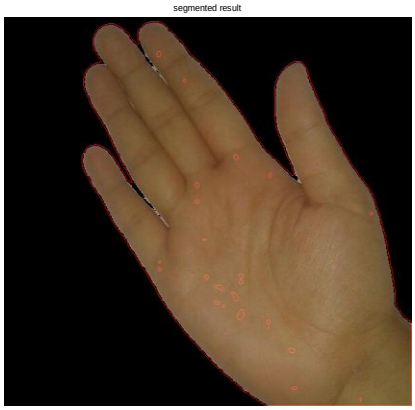
e.



f.



g.



h.

Fig.6.3.5 The images shows the output after the Edge Detection implementation.

6.3.5 PREPROCESSING (NOISE REMOVAL)

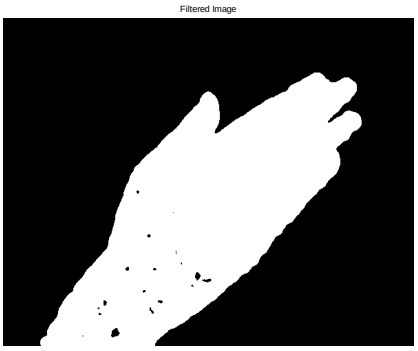
The segmented images are further processed for noise removal. Noise removal is done by various methods like linear filtering, median filtering and adaptive filtering. Here we have used median filter for noise removal.

Median filtering is similar to using an averaging filter, in that each output pixel is set to an average of the pixel values in the neighborhood of the corresponding input pixel. However, with median filtering, the value of an output pixel is determined by the median of the neighborhood pixels, rather than the mean. The median is much less sensitive than the mean to extreme values. Median filtering is therefore better able to remove these outliers

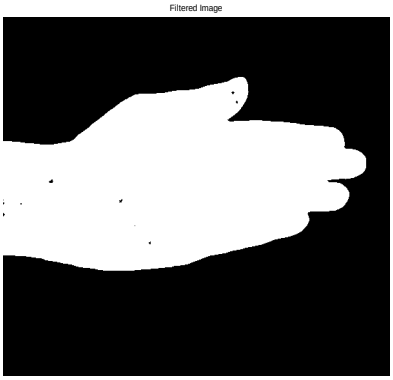
without reducing the sharpness of the image. The median filter2 function implements median filtering.

The output images during the process of filtration:

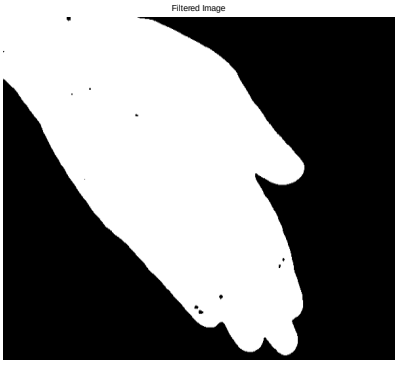
a.



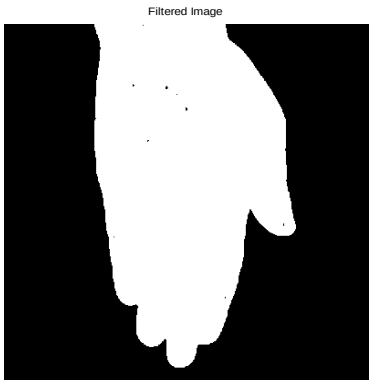
b.



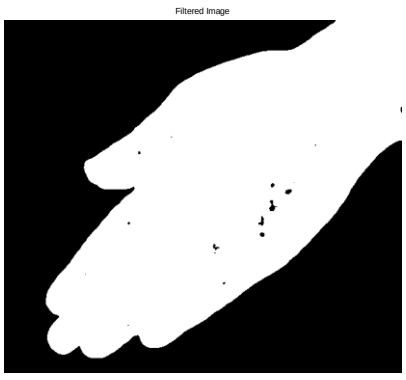
c.



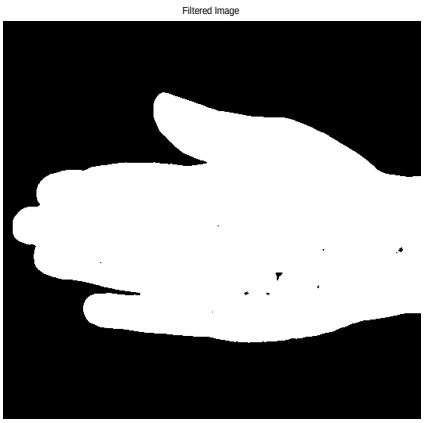
d.



e.



f.



g.

h.

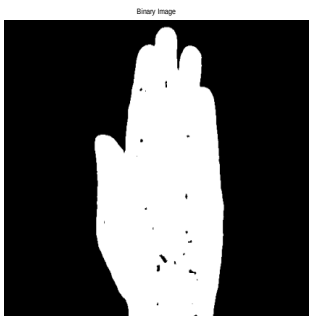
figure 6.3.4.1 filtered image

The Binarisation function is the next step for the further process. The function regionprops is very useful for measuring the properties of objects in a binary image. There are documentation examples and product demos showing how to do this.

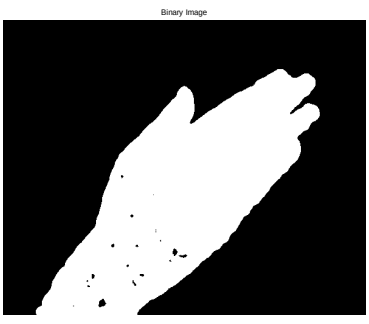
But sometimes we need to process pixel values in the "original" grayscale image. In other words, suppose our process is something like this:

1. Segment grayscale image to get a binary image of objects.
2. Analyze the original grayscale pixel values corresponding to each object in the binary image.

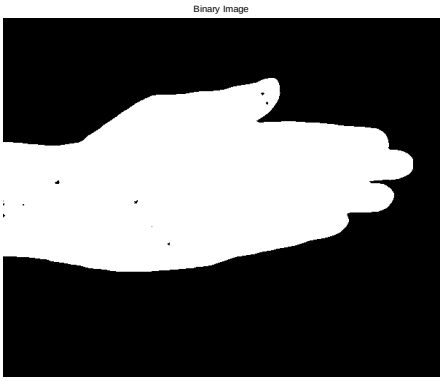
Segment the gray scale image by creating a binary image containing the objects in the image.



a.

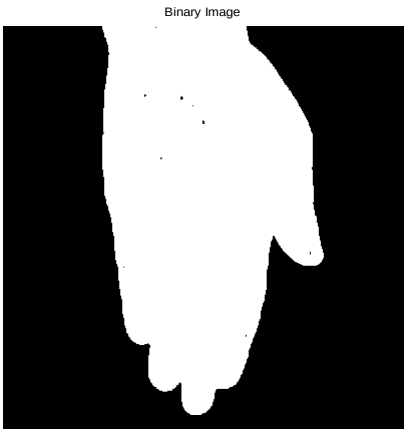


b.



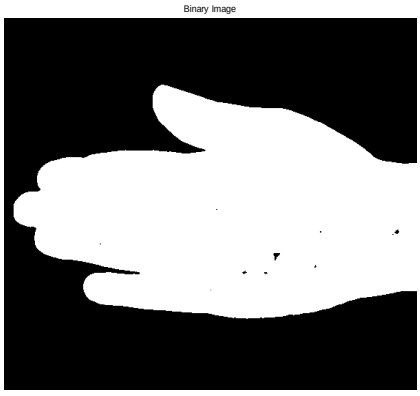
c.

d.

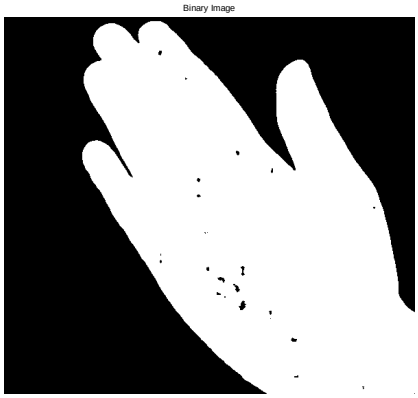


e.

f.



g.



h.

figure 6.3.6 binary images

The regionprops function supports several properties that can be used with grayscale images, including 'WeightedCentroid', 'MeanIntensity', 'MinIntensity', and 'MaxIntensity'. These properties use the original pixel values of the objects for their calculations.

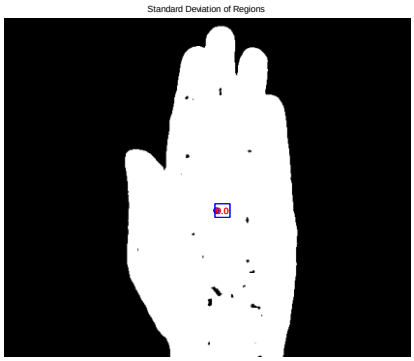
We have use regionprops to calculate both the centroid and weighted centroid of objects in the image. We pass the binary image (BW) containing our objects and the original grayscale image (I) as arguments into regionprops.

s = regionprops(BW, I, {'Centroid','WeightedCentroid'});

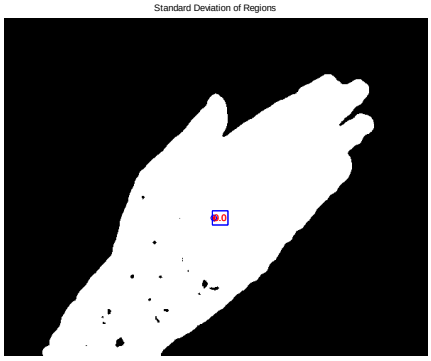
To compare the weighted centroid locations with the unweighted centroid locations, we display the original image and then, using the hold and plot functions, superimpose the centroids on the image.

6.3.6 FEATURE EXTRACTION (CENTROID METHOD)

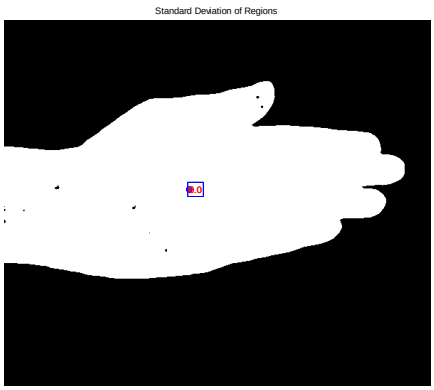
The output image after calculating the pixel value of the centroid:



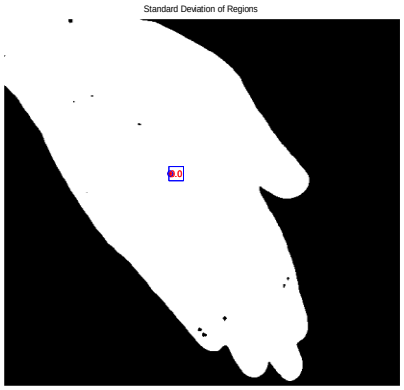
a.



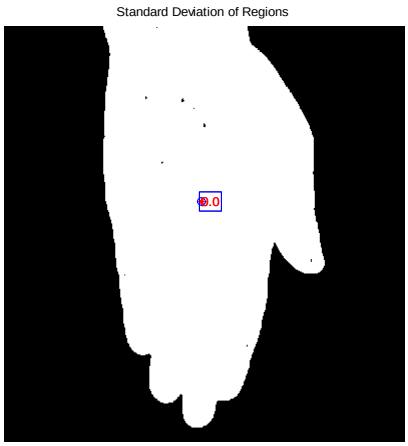
b.



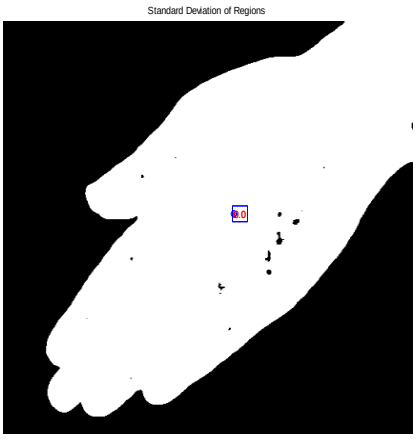
c.



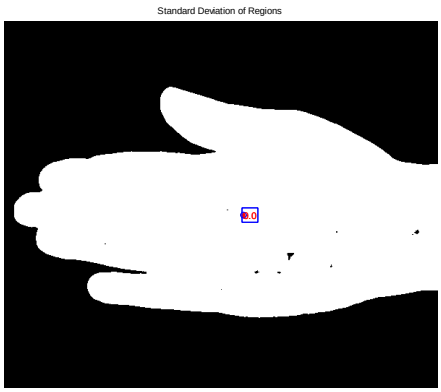
d.



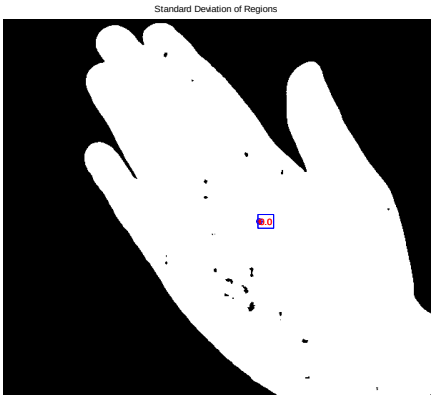
e.



f.



g.



h.

Fig. 6.3.7: Centroid

The pixel values for centroid are converted into polar values and are saved as the theta vaues as shown below:

th1 = finalpolcor(k1)

Different theta values are obtained for all the target and input values

th1 =312.0982 283.3707(Input1)

th2 =298.0000 248.5038(Target1)

th3 =299.0027 237.9975(Target2)

th4 =300.5009 258.0002(Target3)

th5 =299.0018 258.0049(Target4)

th6 =300.5003 324.4980(Target5)

th7 =299.9994 261.0008(Target6)

th8 = 300.5013 254.4999(Target7)

th9 =300.0002 274.5006(Target8)

6.3.7 COMPARING RESULT OF INPUT SAMPLE WITH TARGET AND CLASSIFICATION

In this we apply the algorithm and then compare the result with the target images such that to obtain the orientation of the input samples.

x1=th1-th2

x2=th1-th3

x3=th1-th4

x4=th1-th5

x5=th2-th3

x6=th2-th4

x7=th2-th5

x8=th3-th4

x9=th3-th5

x10=th4-th5

The output of the above comparisons

x1 =14.0982   34.8670

x2 =13.0956   45.3732

x3 =11.5973 25.3705

x4 =13.0964 25.3658

x5 = -1.0026 10.5062

x6 = -2.5009 -9.4965

x7 = -1.0018 -9.5012

x8 = -1.4983 -20.0027

x9 = 0.0009 -20.0074

x10 = 1.4992 -0.0047

We have compared the theta values of the input samples with target samples.

6.3.8 OPTIMISING USING NNTOOL

After this we apply the artificial neural network algorithm for the gesture recognition, using nntool in matlab. Here we input data as the pixels of the image and we simulate it using nntool to get the correct output nntool opens the Network/Data Manager window, which allows us to import, create, use, and export neural networks and data.

The NNTool operates the following steps:

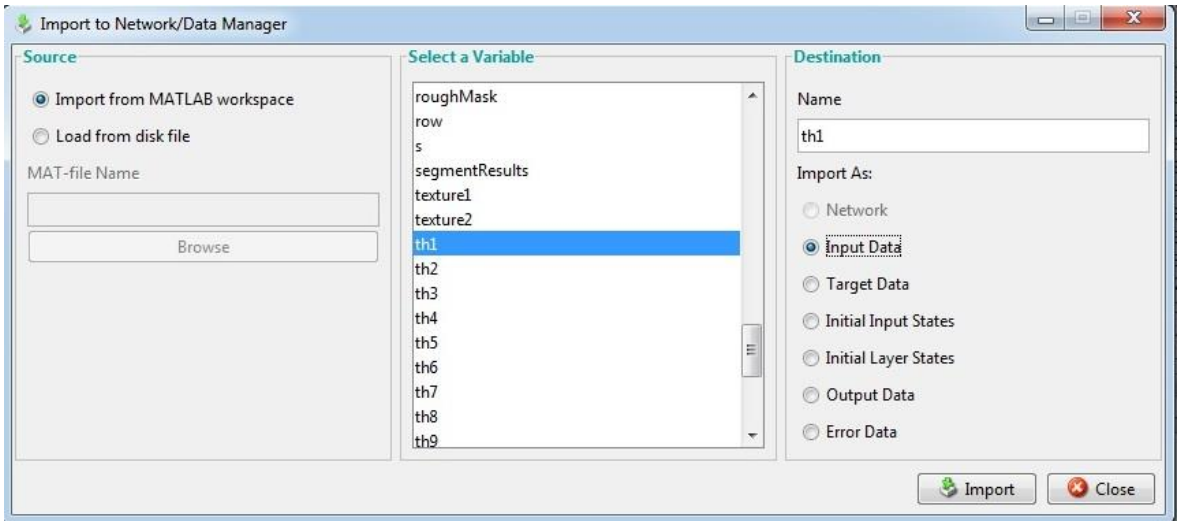


Fig. 6.3.8.1 Import from MATLAB the value of theta of Input Data

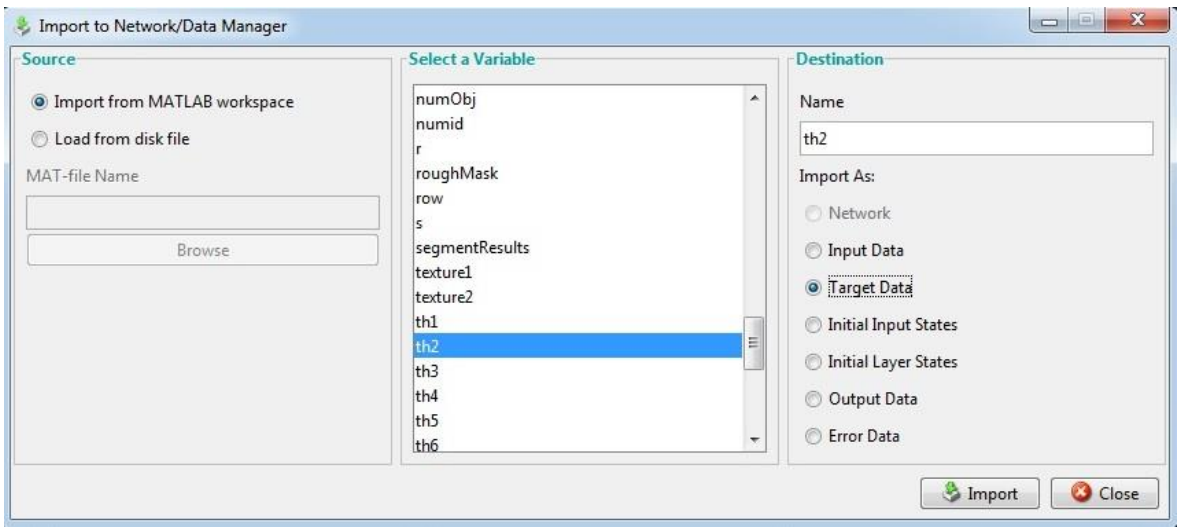


Fig. 6.3.8.2 Import from MATLAB the value of theta of Target Data

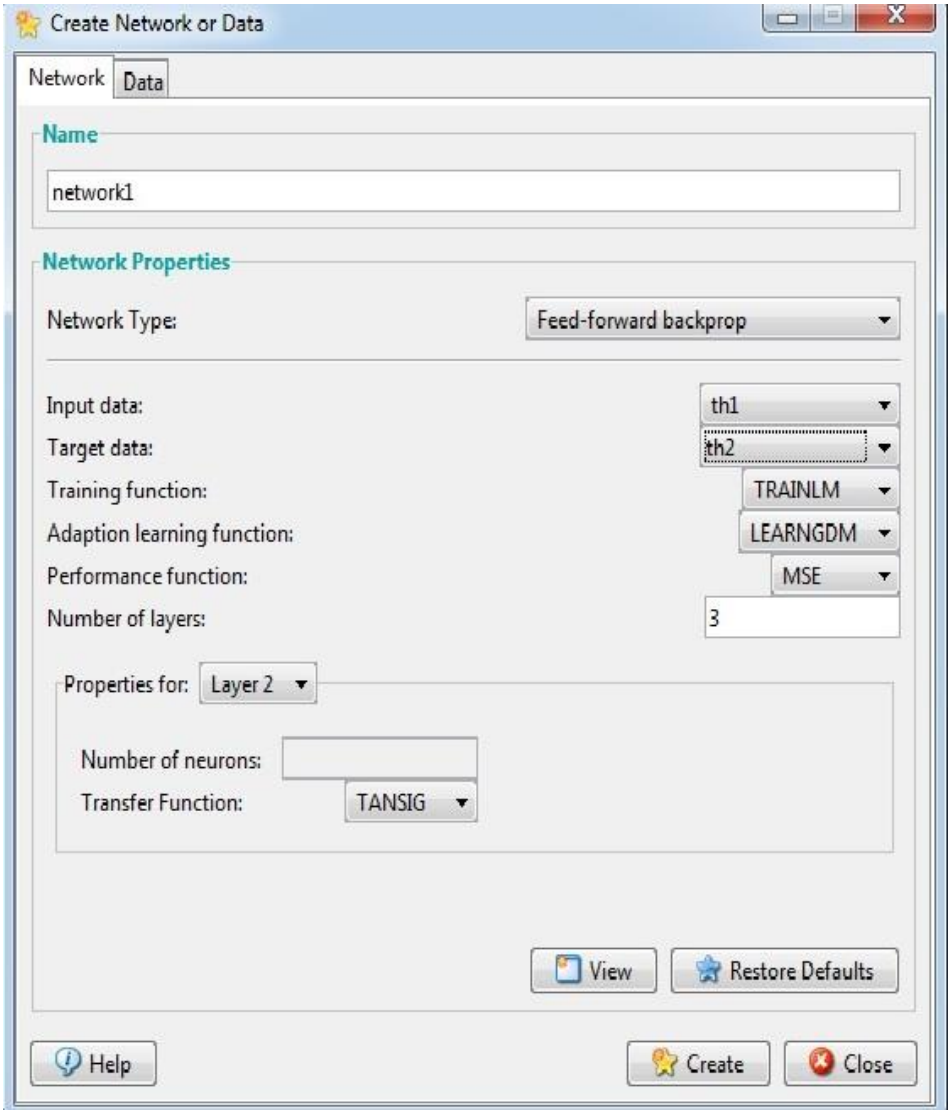


Fig. 6.3.8.3 Creating Networking

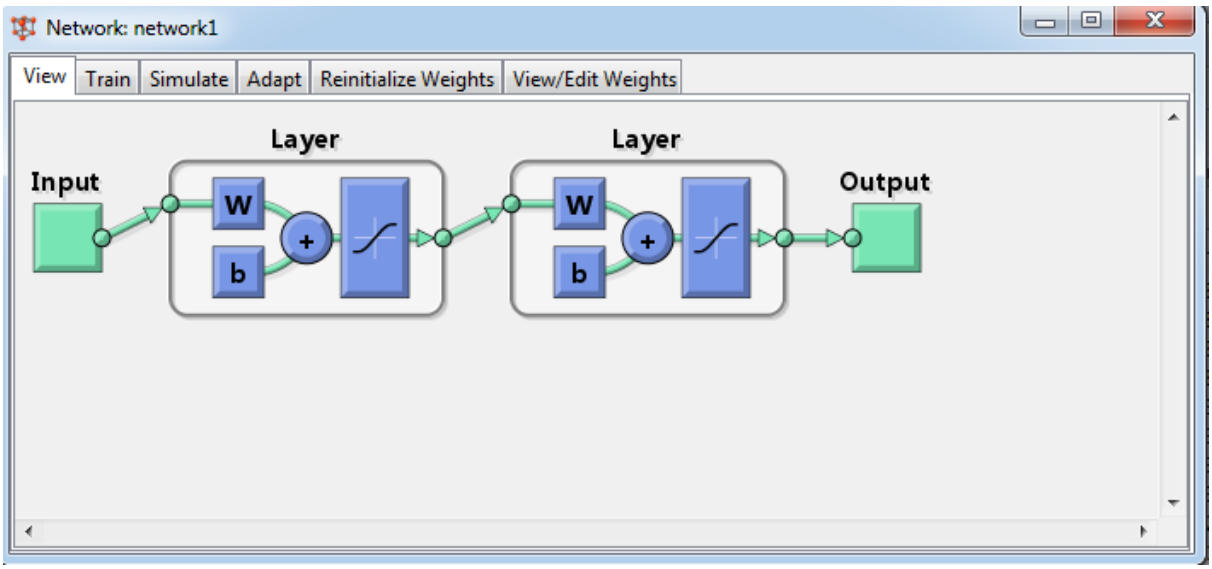


Fig. 6.3.8.4 Networks 1 created

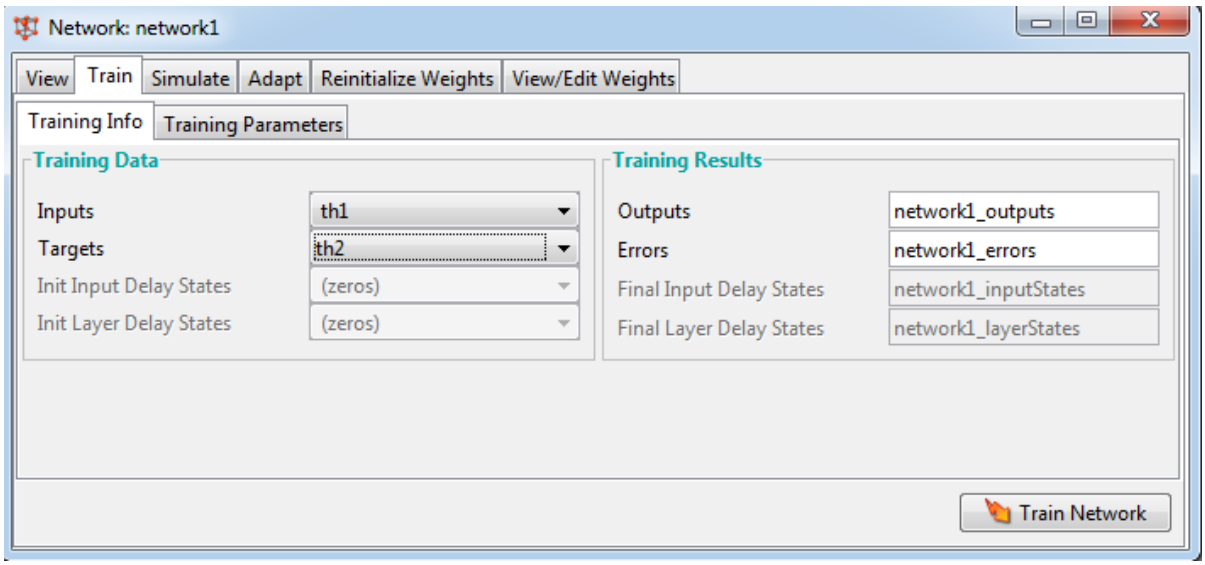


Fig. 6.3.8.5 Training

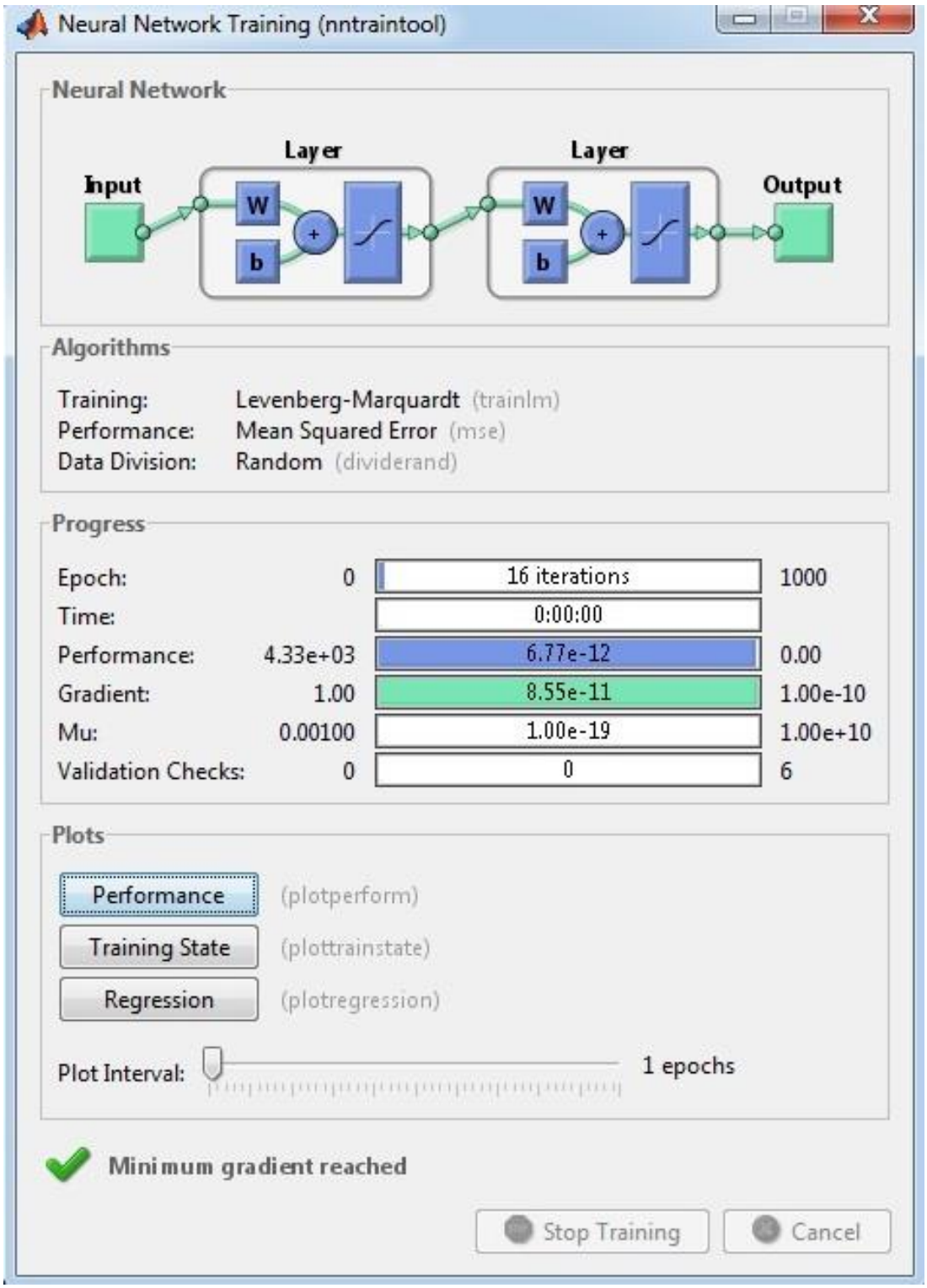


Fig. 6.3.8.6Training of the network

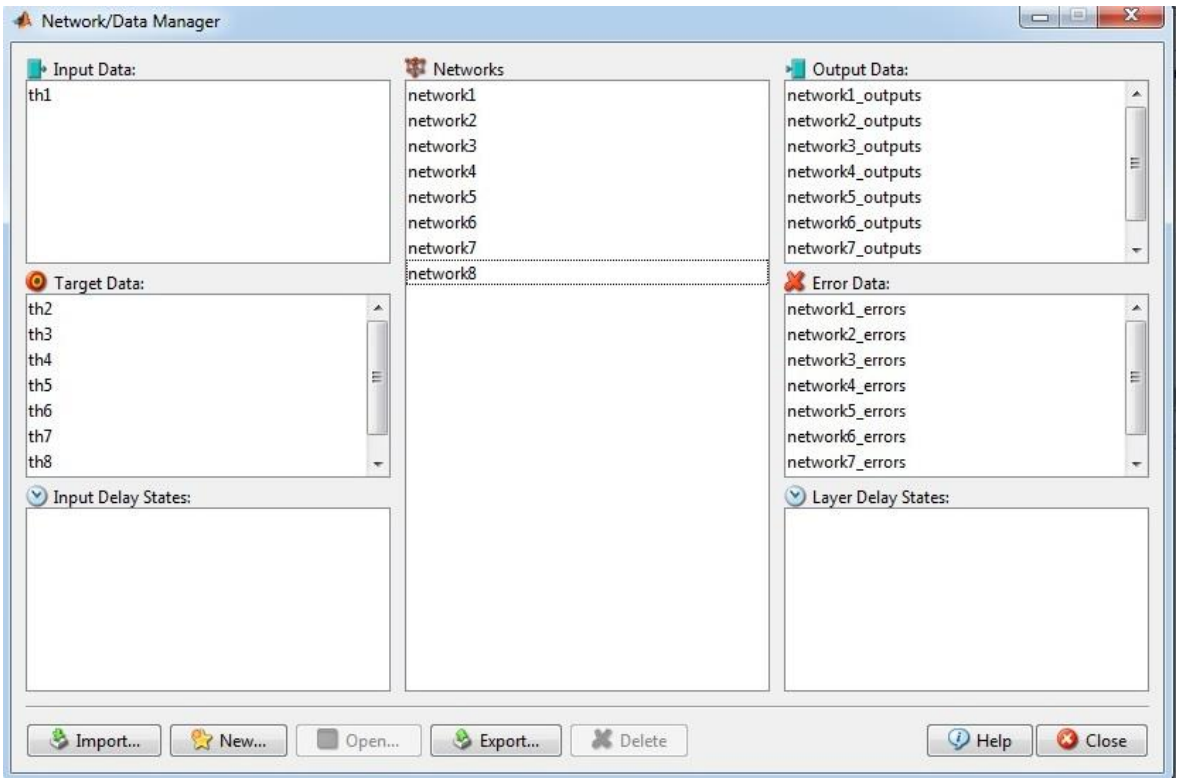


Fig.6.3.8.7 Network Data Manager describing all the networks' output data and error data

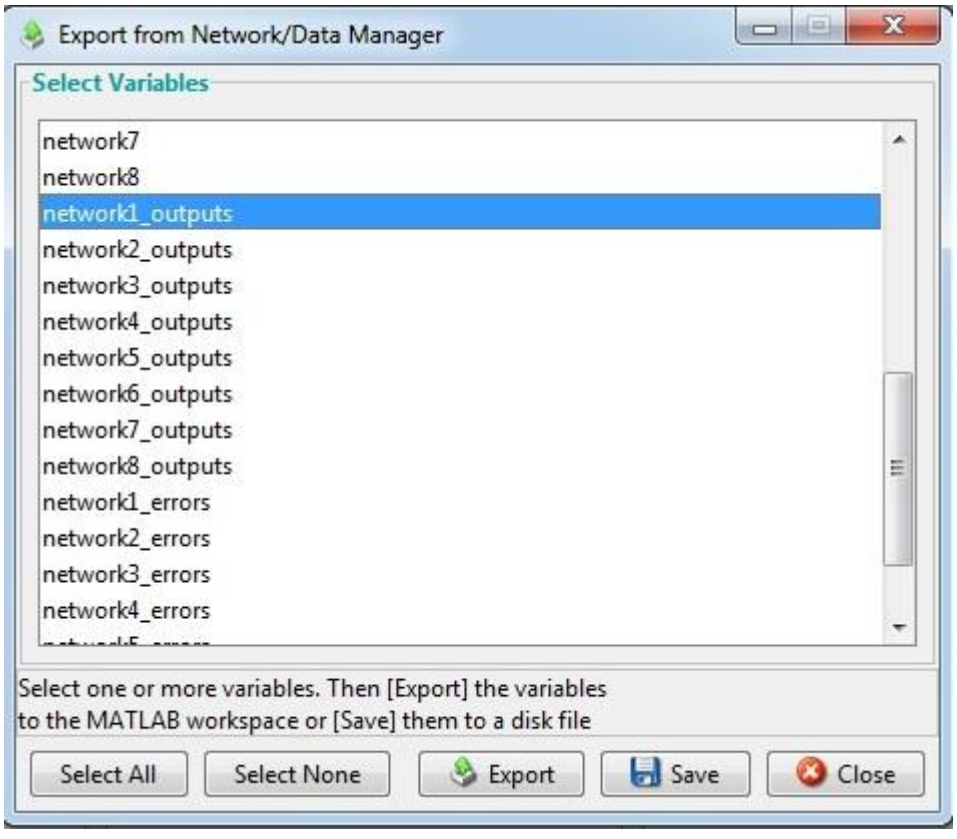


Fig.6.3.8.8 Variables of the network are exported to the workspace

The errors obtained by the NNTool for the sample and target images are given below:

Error for network 1

[-1.7008e-006 2.5708e-007]

Error for network 2

[-2.0179e-006 1.008e-006]

Error for network 3

[-1.3909e-009 0]

Error for network 4

[-2.2332e-006 5.9765e-008]

Error for network 5

[-9.9629e-007 2.5492e-006]

Error for network 6

[-4.1408e-007 2.2096e-006]

Error for network 7

[-3.4937e-006 1.158e-006]

Error for network 8

[-3.2673e-006 1.4828e-006]

The minimum error of input image is optimized. For input image 1 the networks are created between the sample and the targets and the neural network is trained which gives the network errors. The network errors give us the information regarding how much the input or the sample image is closer or falls in the category of the target image. Here the second network created gives us the minimum error and thus we can conclude that the input image falls in the category of target image 2 of orientation of 45 degrees. Thus this approach of using NNTool can be used for different image orientations analysis.

7 CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

In this project, we have discussed how hand gestures recognition can be done using MATLAB and optimization of results is done using NNTool in MATLAB. NNTool is based on Artificial Neural Network concept. We are using camera as a detecting device as well as input device for Reality System. One of the most effective of software computing techniques is Artificial Neural Networks that has many applications on hand gesture recognition problem. The eight target images of different orientations (0 degree, 45 degree, 90 degree, 135 degree, 180 degree, 225 degree, 270 degree, 315 degree) and input samples are captured using image acquisition toolbox and went under skin segmentation where skin detection algorithm is applied on them. The image is converted in skin pixel and non skin pixels and further the preprocessing is done on the skin detected images like edge detection, background removal, noise removal, and binarization. Then feature extraction must be done, different methods can be used like geometric features or non-geometric features, geometric features that use angles and orientations, palm center. Non geometric such as color, silhouette and textures, but they are in adequate in recognition. Here we have used geometric feature extraction using various geometric algorithms. Then the geometric feature of input sample images were compared with the eight target images of different orientations (0 degree, 45 degree, 90 degree, 135 degree, 180 degree, 225 degree, 270 degree, 315 degree). Centroid method of geometrical feature extraction is used and thus theta value is calculated for each target and input samples. And these theta values are given as input and target values for creating 8 different networks giving using us the network errors such that we can decide the orientation of the input sample with respect to different targets images. The network giving the minimum error is decided as

the best network for the input sample and the target values. Here the second network created gives us the minimum error and thus we can conclude that the input image falls in the category of target image 2 of orientation of 45 degrees. Thus this approach of using NNTool can be used for different image orientations analysis.

FUTURE ENHANCEMENT

Area under the Hand gesture based human-computer interaction is very vast. This project recognizes hand gesture to work under the real time purposes. Hand recognition system can be useful in many fields like robotics, computer human interaction and so making this system for real time will be the future work to do.

- Support Vector Machine can be modified for reduction of complexity. Reduction of complexity leads us to a less computation time. Reduced complexity provides us less computation time so we can make system to work real time.
- Further the project extends its approach by recognizing hand postures independently of the hand orientation while using a better representation of the hand compared to the mostly available ones.
- Can be used in security system
- We will bring new opportunities for mechanical engineering based companies to use Augmented Reality for simplification of their complex tasks.

7. BIBLIOGRAPHY

[1] Artificial Neural Network –an introduction by Kelvin L. Priddy and Paul E. Keller

[2] Artificial Neural Network in real life application by Jaun R. Raunal and Julin Dorado

[3] C. L. Lisetti and D. J. Schiano ,"Automatic classification of single facial images," Pragmatics Cogn., vol. 8, pp. 185–235, 2000.

[4] G. R. S. Murthy, R. S. Jadon. "A Review of Vision Based Hand Gestures Recognition," International Journal of Information Technology and Knowledge Management, vol. 2(2), pp. 405-410. 2009.

[6]C. L. Lisetti and D. J. Schiano, "Automatic classification of single facial images," Pragmatics Cogn., vol. 8, pp. 185–235, 2000.

[7]P. Garg, N. Aggarwal and S. Sofat. "Vision Based Hand Gesture Recognition," World Academy of Science, Engineering and Technology vol. 49, pp. 972-977, 2009.

[8] Real-Time Gesture Recognition by Means of Hybrid Recognizers, Volume 2298, pp 34-47,2002

[9] J. Mathews: "An Introduction to Edge Detection: The Sobel Edge Detector," 2002.

[10] Joseph J. LaViola Jr.,"A Survey of Hand Posture and Gesture Recognition Techniques and Technology," Master Thesis, NSF Science and Technology Center for Computer Graphics and Scientific Visualization, USA, 1999.

[11] M. M. Hasan, P. K. Mishra. "HSV Brightness Factor Matching for Gesture Recognition System", International Journal of Image Processing (IJIP), vol. 4(5), pp. 456-467.2010

[12] J. Canny, "A computational approach to edge detection," IEEE Trans. Pattern Anal. Machine. Intel1, vol. 8, no. 6, pp. 679–698, Nov. 1986.

[13] Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intellignece 22 (2000)

[14] http://www.mathworks.in

[15]Lalit Gupta and Suwei Ma "Gesture-Based Interaction and Communication: Automated Classification of Hand Gesture Contours", IEEE transactions on systems, man, and cybernetics—part c: applications and reviews, vol. 31, no. 1, February 2001

[16] http://www.mathworks.in/matlabcentral/fileexchange/28565-skin-detection

[17] Final Report - Hand Gesture Recognition using Neural Networks by UNIS

ANNEXURE I

PROGRAM

```
clear all;

close all;

clc;


  %%

  %hsv segmentation

  img_orig=imread('C:\Documents and Settings\Administrator\Desktop\c.jpg');

  img=img_orig; %copy of original image

  hsv=rgb2hsv(img);

  h=hsv(:,:,1);

  s=hsv(:,:,2);


  [r c v]=find(h>0.25 | s<=0.3 | s>0.9); %non skin

  numid=size(r,1);


  for i=1:numid

     img(r(i),c(i),:)=0;

  end

  imshow(img);


  %ycbcr segmentation

  img_ycbcr=img;  %image from the previous segmentation

  ycbcr=rgb2ycbcr(img_ycbcr);
```

```
cb=ycbcr(:,:,2);
cr=ycbcr(:,:,3);

%Detect Skin
%[r,c,v] = find(cb>=77 & cb<=127 & cr>=133 & cr<=173);
[r c v] = find(cb<=77 | cb >=127 | cr<=133 | cr>=173);
numid = size(r,1);

%Mark Skin Pixels
for i=1:numid
    img_ycbcr(r(i),c(i),:) = 0;
   % bin(r(i),c(i)) = 1;
end
figure
title('ycbcr segmentation');
imshow(img_ycbcr);

%rgb segmentation
img_rgb=img_ycbcr;
r=img_rgb(:,:,1);
g=img_rgb(:,:,2);
b=img_rgb(:,:,3);
[row col v]= find(b>0.79*g-67 & b<0.78*g+42 & b>0.836*g-14 & b<0.836*g+44 );
%non skin pixels
numid=size(row,1);
```

```
for i=1:numid
    img_rgb(row(i),col(i),:)=0;
end
imshow(img_ycbcr);
imshow(img_rgb);

% segmentation
E = entropyfilt(I);
Eim = mat2gray(E);
imshow(Eim);
BW1 = im2bw(Eim, .8);
imshow(BW1);
figure, imshow(I);
BWao = bwareaopen(BW1,2000);
imshow(BWao);
nhood = true(9);
closeBWao = imclose(BWao,nhood);
imshow(closeBWao)
roughMask = imfill(closeBWao,'holes');
imshow(roughMask);
figure, imshow(I);
I2 = I;
I2(roughMask) = 0;
imshow(I2);
E2 = entropyfilt(I2);
E2im = mat2gray(E2);
```

```
imshow(E2im);
BW2 = im2bw(E2im,graythresh(E2im));
imshow(BW2)
figure, imshow(I);
mask2 = bwareaopen(BW2,1000);
imshow(mask2);
texture1 = I;
texture1(~mask2) = 0;
texture2 = I;
texture2(mask2) = 0;
imshow(texture1);
figure, imshow(texture2);
boundary = bwperim(mask2);
segmentResults = I;
segmentResults(boundary) = 255;
imshow(segmentResults);
% filteration for noise removal
h = fspecial('unsharp');
I2 = imfilter(mask2,h);
imshow(mask2), title('Original Image')
figure, imshow(I2), title('Filtered Image')

% binarization
BW = I2 > 0;
imshow(BW)
title('Binary Image');
```

```
% converting to double
k=BW;
k1=mat2gray(double(k))
imshow(k1);
% calculating centroid  and pixel value


s = regionprops(BW, k1, {'Centroid','WeightedCentroid'});
hold on
numObj = numel(s);
for k = 1 : numObj
plot(s(k).WeightedCentroid(1), s(k).WeightedCentroid(2), 'r*');
plot(s(k).Centroid(1), s(k).Centroid(2), 'bo');
end
hold off
s = regionprops(BW, k1, {'Centroid','PixelValues','BoundingBox'});
imshow(k1);
title('Standard Deviation of Regions');
hold on
for k = 1 : numObj
s(k).StandardDeviation = std(double(s(k).PixelValues));
text(s(k).Centroid(1),s(k).Centroid(2), ...
sprintf('%2.1f', s(k).StandardDeviation), ...
'EdgeColor','b','Color','r');
end
hold off
th1 = finalpolcor(k1)
```

```
% target 1
a1 = imread('C:\Users\ShristiPriya\Desktop\target 1.png');
figure, imshow(a1), title('original image 1');
% converting input image from rgb to gray
J = rgb2gray(a1);
imshow(J)
% segmentation
E = entropyfilt(J);
Eim = mat2gray(E);
imshow(Eim);
BW = im2bw(Eim, .8);
imshow(BW);
figure, imshow(J);
BWao = bwareaopen(BW,2000);
imshow(BWao);
nhood = true(9);
closeBWao = imclose(BWao,nhood);
imshow(closeBWao)
roughMask = imfill(closeBWao,'holes');
imshow(roughMask);
figure, imshow(J);
J2 = J;
J2(roughMask) = 0;
imshow(J2);
E2 = entropyfilt(J);
```

```
E2im = mat2gray(E2);

imshow(E2im);

BW2 = im2bw(E2im,graythresh(E2im));

imshow(BW2)

figure, imshow(J);

mask2 = bwareaopen(BW2,1000);

imshow(mask2);

texture1 = J;

texture1(~mask2) = 0;

texture2 = J;

texture2(mask2) = 0;

imshow(texture1);

figure, imshow(texture2);

boundary = bwperim(mask2);

segmentResults = J;

segmentResults(boundary) = 255;

imshow(segmentResults);

% filteration for noise removal

h = fspecial('unsharp');

I2 = imfilter(mask2,h);

imshow(mask2), title('Original Image 1')

figure, imshow(J), title('Filtered Image 1')


% binarization

BW = J2 > 0;

imshow(BW)
```

```
title('Binary Image 1');
k2=BW;
k3=mat2gray(double(k2))
imshow(k3);
% calculating centroid  and pixel value

s = regionprops(BW, k3, {'Centroid','WeightedCentroid'});
hold on
numObj = numel(s);
for k = 1 : numObj
plot(s(k).WeightedCentroid(1), s(k).WeightedCentroid(2), 'r*');
plot(s(k).Centroid(1), s(k).Centroid(2), 'bo');
end
hold off
s = regionprops(BW, k3, {'Centroid','PixelValues','BoundingBox'});
imshow(k3);
title('Standard Deviation of Regions 1');
hold on
for k = 1 : numObj
s(k).StandardDeviation = std(double(s(k).PixelValues));
text(s(k).Centroid(1),s(k).Centroid(2), ...
sprintf('%2.1f', s(k).StandardDeviation), ...
'EdgeColor','b','Color','r');
end
hold off
th2 = finalpolcor(k3)
```

```
%target 2
a2 = imread('C:\Users\ShristiPriya\Desktop\target 2.png');
figure, imshow(a2), title('original image 2');
% converting input image from rgb to gray
L = rgb2gray(a2);
imshow(L)
% segmentation
E = entropyfilt(L);
Eim = mat2gray(E);
imshow(Eim);
BW1 = im2bw(Eim, .8);
imshow(BW1);
figure, imshow(L);
BWao = bwareaopen(BW1,2000);
imshow(BWao);
nhood = true(9);
closeBWao = imclose(BWao,nhood);
imshow(closeBWao)
roughMask = imfill(closeBWao,'holes');
imshow(roughMask);
figure, imshow(L);
L2 = L;
L2(roughMask) = 0;
imshow(L2);
E2 = entropyfilt(L2);
```

```
E2im = mat2gray(E2);

imshow(E2im);

BW2 = im2bw(E2im,graythresh(E2im));

imshow(BW2)

figure, imshow(L);

mask2 = bwareaopen(BW2,1000);

imshow(mask2);

texture1 = L;

texture1(~mask2) = 0;

texture2 = L;

texture2(mask2) = 0;

imshow(texture1);

figure, imshow(texture2);

boundary = bwperim(mask2);

segmentResults = L;

segmentResults(boundary) = 255;

imshow(segmentResults);

% filteration for noise removal

h = fspecial('unsharp');

L2 = imfilter(mask2,h);

imshow(mask2), title('Original Image 2')

figure, imshow(L2), title('Filtered Image 2')


% binarization

BW = L2 > 0;

imshow(BW)
```

```
title('Binary Image 2');
k4=BW;
k5=mat2gray(double(k4))
imshow(k5);
% calculating centroid  and pixel value


s = regionprops(BW, k5, {'Centroid','WeightedCentroid'});
hold on
numObj = numel(s);
for k = 1 : numObj
plot(s(k).WeightedCentroid(1), s(k).WeightedCentroid(2), 'r*');
plot(s(k).Centroid(1), s(k).Centroid(2), 'bo');
end
hold off
s = regionprops(BW, k5, {'Centroid','PixelValues','BoundingBox'});
imshow(k5);
title('Standard Deviation of Regions 2');
hold on
for k = 1 : numObj
s(k).StandardDeviation = std(double(s(k).PixelValues));
text(s(k).Centroid(1),s(k).Centroid(2), ...
sprintf('%2.1f', s(k).StandardDeviation), ...
'EdgeColor','b','Color','r');
end
hold off
th3 = finalpolcor(k5)
```

```
%target 3
a3 = imread('C:\Users\ShristiPriya\Desktop\target 3.jpg');
figure, imshow(a3), title('original image 3');
% converting input image from rgb to gray
M = rgb2gray(a3);
imshow(M)
% segmentation
E = entropyfilt(M);
Eim = mat2gray(E);
imshow(Eim);
BW1 = im2bw(Eim, .8);
imshow(BW1);
figure, imshow(M);
BWao = bwareaopen(BW1,2000);
imshow(BWao);
nhood = true(9);
closeBWao = imclose(BWao,nhood);
imshow(closeBWao)
roughMask = imfill(closeBWao,'holes');
imshow(roughMask);
figure, imshow(M);
M2 = M;
M2(roughMask) = 0;
imshow(M2);
E2 = entropyfilt(M2);
```

```
E2im = mat2gray(E2);

imshow(E2im);

BW2 = im2bw(E2im,graythresh(E2im));

imshow(BW2)

figure, imshow(M);

mask2 = bwareaopen(BW2,1000);

imshow(mask2);

texture1 = M;

texture1(~mask2) = 0;

texture2 = M;

texture2(mask2) = 0;

imshow(texture1);

figure, imshow(texture2);

boundary = bwperim(mask2);

segmentResults = M;

segmentResults(boundary) = 255;

imshow(segmentResults);

% filteration for noise removal

h = fspecial('unsharp');

M2 = imfilter(mask2,h);

imshow(mask2), title('Original Image 3')

figure, imshow(M2), title('Filtered Image 3')


% binarization

BW = M2 > 0;

imshow(BW)
```

```
title('Binary Image 3');
k6=BW;
k7=mat2gray(double(k6))
imshow(k7);
% calculating centroid  and pixel value


s = regionprops(BW, M2, {'Centroid','WeightedCentroid'});
hold on
numObj = numel(s);
for k = 1 : numObj
plot(s(k).WeightedCentroid(1), s(k).WeightedCentroid(2), 'r*');
plot(s(k).Centroid(1), s(k).Centroid(2), 'bo');
end
hold off
s = regionprops(BW, M2, {'Centroid','PixelValues','BoundingBox'});
imshow(M2);
title('Standard Deviation of Regions 3');
hold on
for k = 1 : numObj
s(k).StandardDeviation = std(double(s(k).PixelValues));
text(s(k).Centroid(1),s(k).Centroid(2), ...
sprintf('%2.1f', s(k).StandardDeviation), ...
'EdgeColor','b','Color','r');
end
hold off
th4 = finalpolcor(M2)
```

```
% target 4


a4 = imread('C:\Users\ShristiPriya\Desktop\target 4.png');
figure, imshow(a4), title('original image 4');
% converting input image from rgb to gray
N = rgb2gray(a4);
imshow(N)
% segmentation
E = entropyfilt(N);
Eim = mat2gray(E);
imshow(Eim);
BW1 = im2bw(Eim, .8);
imshow(BW1);
figure, imshow(N);
BWao = bwareaopen(BW1,2000);
imshow(BWao);
nhood = true(9);
closeBWao = imclose(BWao,nhood);
imshow(closeBWao)
roughMask = imfill(closeBWao,'holes');
imshow(roughMask);
figure, imshow(N);
N2 = N;
N2(roughMask) = 0;
imshow(N2);
```

```
E2 = entropyfilt(N2);

E2im = mat2gray(E2);

imshow(E2im);

BW2 = im2bw(E2im,graythresh(E2im));

imshow(BW2)

figure, imshow(N);

mask2 = bwareaopen(BW2,1000);

imshow(mask2);

texture1 = N;

texture1(~mask2) = 0;

texture2 = N;

texture2(mask2) = 0;

imshow(texture1);

figure, imshow(texture2);

boundary = bwperim(mask2);

segmentResults = N;

segmentResults(boundary) = 255;

imshow(segmentResults);

% filteration for noise removal

h = fspecial('unsharp');

N2 = imfilter(mask2,h);

imshow(mask2), title('Original Image 4')

figure, imshow(N2), title('Filtered Image 4')


% binarization

BW = N2 > 0;
```

```
imshow(BW)
title('Binary Image 4');
k8=BW;
k9=mat2gray(double(k8))
imshow(k9);
% calculating centroid  and pixel value


s = regionprops(BW, k9, {'Centroid','WeightedCentroid'});
hold on
numObj = numel(s);
for k = 1 : numObj
plot(s(k).WeightedCentroid(1), s(k).WeightedCentroid(2), 'r*');
plot(s(k).Centroid(1), s(k).Centroid(2), 'bo');
end
hold off
s = regionprops(BW, k9, {'Centroid','PixelValues','BoundingBox'});
imshow(k9);
title('Standard Deviation of Regions 4');
hold on
for k = 1 : numObj
s(k).StandardDeviation = std(double(s(k).PixelValues));
text(s(k).Centroid(1),s(k).Centroid(2), ...
sprintf('%2.1f', s(k).StandardDeviation), ...
'EdgeColor','b','Color','r');
end
hold off
```

th5=finalpolcor(k9)

th1

th2

th3

th4

th5

x1=th1>th2

x2=th1>th3

x3=th1>th4

x4=th1>th5

x5=th2>th3

x6=th2>th4

x7=th2>th5

x8=th3>th4

x9=th3>th5

x10=th4>th5

x11=th3>th4