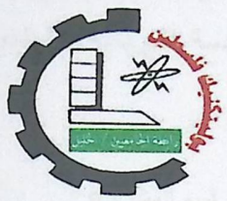
Palestine Polytechnic University



College of Engineering & Technology
Computer and Electrical Engineering Department

Graduation Project

Hand Gesture Based TV

Project Team

Ahmad S. Tamimi Bashar W. Hamuri Motaz R. Qawasmi

Project Supervisor
Dr. Alaa H. Halawani

Hebron – Palestine

May, 2008

جامعة بوليتيكنيك فلسطين

الخليل – فلسطين

كلية الهندسة والتكنولوجيا

دائرة الهندسة الكهربائية و الحاسوب

اسم المشروع :

Hand Gesture Based TV

اسماء الطلاب:

أحمد شوقي التميمي بشار وضاح الحموري معتز رشاد القواسمي

بناء على نظام كلية الهندسة والتكنولوجيا وأشراف و متابعة المشرف المباشر على المشروع ومتابعة اعضاء اللجنة الممتحنة تم تقديم هذ االمشروع إلى دائرة الهندسة الكهربائية و الحاسوب وذلك للوفاء بمتطلبات درجة البكالوريوس في هندسة تخصص أنظمة الحاسوب

توقيع المشرف

.....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

Abstract

Hand Gesture Based TV

This project aims to control the TV through hand gesture without remote control. Two important problems are taken into consideration while determining the gestures; first one is, choosing simple, expressive, and limited number of gestures to make it easy to be memorized. The second one is recognizing these gestures.

A solution to these problems is that, Users have six gestures to handle television control: the open hand for switch ON the TV, the closed hand to switch OFF the TV, the other two gestures are for increment and decrement operations (for both volume and channel), the fifth one to switch from the channel mode to volume mode, and the sixth gesture for no operation mode. While the other issue which is how the computer can recognize the gesture it's solved using artificial Neural Network.

We made a prototype of this system using a computer, webcam, interfacing circuit, parallel port and television, knowing that a black background was adopted. The system was implemented and tested successfully with an accuracy of 95%.

Dedication

To our Parents.

To Osama.

Acknowledgments

To Allah for all the blessing that he gave us to do such work and to think in such away.

To Prophet Mohammed who carried this religion and suffered in order to bring the Islam to humanity.

To parents who stand with us through all of this long and hard way, until we reach this day, for them, we will never forget this, and wish Allah that we will do any thing to compensate for all of this inshallah.

To Dr.Alaa H. Halawani for his support, supervised and patients.

To all our teachers starting from the first grade until this day and the days follows, for these people, we are very grateful for the entire job that people do for us and for all the generations that follow.

To all friends how help us in this project and give us samples, and they how give us support and thoughts.

Contents

List of Table

List of Figure

List of Lists

Chapter 1

Introduction

Chapter Contents:

1.1 General idea about the project and its importance

This study is how to operate a television set remotely. This is a familiar, yet useful problem. People value the ability to control a television set from a distance. In a survey, Americans were asked what "high technology" gadget had improved their quality of life the most. The top responses were "microwave ovens" and “television remote controls" [4].

Contemporary hand-held television remote controls are very successful, yet not without flaws. They can be lost. There is a small industry for products related to losing remote controls replacements remotes, devices which indicate where the remote control is, and televisions which locate the remote control. Even if the remote control is not lost, it can be an inconvenience to have to get it from another part of the room. It is reasonable to study additional ways to control the television remotely [1].

Voice and vision are two natural candidates. Voice has the advantage of a pre-established vocabulary (natural language). However, it may not be appropriate for the protracted issuing of commands, such as while "channel surfing", nor for changing parameters by increments, as with volume control. Vision (Gestural control) may be more appropriate than voice for some tasks, yet lacks a natural vocabulary. For this work, we focused on vision based control. Future systems may ultimately use a combination of the two. There has been much recent interest in the computer vision problem of hand gesture recognition [5, 6, and 7]. However, most methods either do not operate in real time without special hardware, or else are not appropriate for an unpredictable scene. We will tailor our recognition method to the designed user interface. We seek a user interface which new users can instantly master. Here we confront a fundamental problem in the control of machines by hand gestures: the lack of a vocabulary.

There are a set of possible commands, such as "ON", "UP "Channel switch", yet no universal set of hand signals with which to specify them. Main issue is not to require the user to memorize complicated gestures. There is a related problem from the computer's image processing perspective: How can a computer identify and classify the hand gestures quickly and reliably?

1.2 Project objective

This project aims to achieve some objectives that listed, they are as follows:

- Control the television through hand without a remote controller, where the human hand become the main controller to the TV rather than remote controller by suggesting common gesture that are related to their function in control TV in which achieve ease .
- Reliable functionality under different transformations (geometric, photometric).
    - Respond to the gesture with different scales (geometric).
    - Respond to the gesture with different light brightness (photometric).

There are a set of possible commands, such as "ON", "UP "Channel switch", yet no universal set of hand signals with which to specify them. Main issue is not to require the user to memorize complicated gestures. There is a related problem from the computer's image processing perspective: How can a computer identify and classify the hand gestures quickly and reliably?

1.2 Project objective

This project aims to achieve some objectives that listed, they are as follows:

- Control the television through hand without a remote controller, where the human hand become the main controller to the TV rather than remote controller by suggesting common gesture that are related to their function in control TV in which achieve ease .

- Reliable functionality under different transformations (geometric, photometric).
    - Respond to the gesture with different scales (geometric).
    - Respond to the gesture with different light brightness (photometric).

1.3 Literature review

Different Topics are covered, some related to methods that are used, and others were as a form of research.

Here are some of these studies that are in some way are related to our project.

In [1], Freeman and Weissman studied how a viewer can control a television set remotely by hand gestures. And they addressed two fundamental issues of gesture–based human–computer interaction:

1. How one can communicate a rich set of commands without extensive user training and memorization of gestures.

2- How computers recognize the commands in a complicated visual environment.

They use image gradient orientation as a feature extraction method by using a video camera as a capture device, notice that this project deals with the same feature but uses images taken by webcam rather than video.

In [2], Halawani present in his thesis a system for automatic translation of gestures of the manual alphabets in Arabic sign language.

The features used for gesture recognition is the length of the vector from the object center to the boundary of that object at fixed angle interval. The whole work related to this project in its main idea which is the human computer interaction which is presented throw recognition of the human hand to interpret it to voice.

In [3], Vladimir, Sharma and Thomas used hand gestures to provide an attractive alternative to cumbersome interface devices for

human-computer interaction (HCI). In particular, visual interpretation of hand gestures can help in achieving the ease and naturalness desired for HCI. They survey the literature on visual interpretation of hand gestures in the context of its role in HCI. This discussion is organized on the basis of the method used for modeling, analyzing, and recognizing gestures.

All what found about this research that is in IEEE researches and they discuss implemented Gestural systems as well as other potential applications of vision-based gesture recognition and discuss directions of future research in gesture recognition.

1.4 Cost Estimation

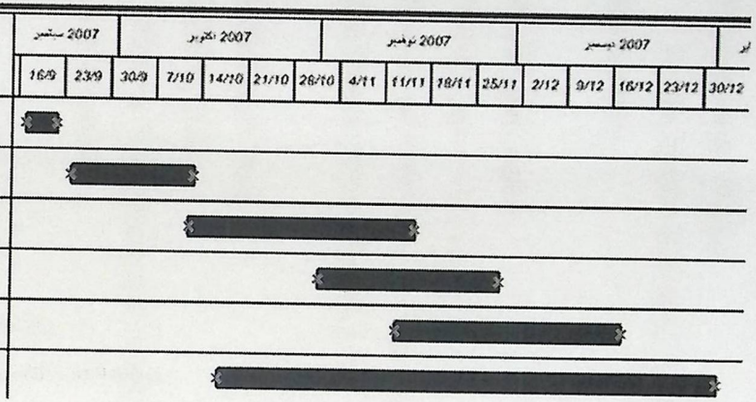
The following table specifies the estimated total cost for the different components in the project.

Number	Object	Cost($)
1	WebCam	20
2	ICs and Chips	10
3	WindowsXP	450
4	Matlab	600
5	Printing	55
Total		1135

Table 1.1: Project components estimated cost.

1.5 Time plane

ID	Task Name	Start	Finish	Duration
1	Idea decision	17/09/2007	21/09/2007	1w
2	Gathering Information	24/09/2007	12/10/2007	3w
3	Data Analyses	12/10/2007	15/11/2007	5w
4	Primary Design	01/11/2007	28/11/2007	4w
5	Primary Implementation	13/11/2007	17/12/2007	5w
6	Decumintation	17/10/2007	01/01/2008	11w



ID	Task Name	Start	Finish	Duration
1	Software implementation	03/03/2008	15/05/2008	10.8w
2	Interface circuit	25/04/2008	15/05/2008	3w
3	Testing	02/05/2008	15/05/2008	2w
4	Documentation	10/04/2008	22/05/2008	6.2w

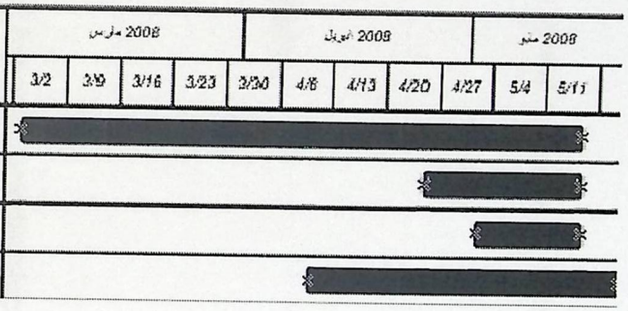


Table 1.2: project scheduling stages and time tables

1.6 Project risk management

- **Technology Risks**

  Some risks may occur because of the software or hardware used in the system.

Hardware risks:

- TV / Receiver malfunctions.
- Computer / Microprocessor malfunctions.
- Used webcam malfunctions.

Software risks:

- Incompatibility.
- Problems that might occur with using the software development environment.

- **People / Staff Risks**

- Member of the team gets ill.
- Member of the team becomes unavailable for any reason.

- **Organizational Risks**

- Facing financial problems.
- Facing project resources problems.

- **Tools Risks**

- Lose of any supported software or hardware used to develop the system.

- **Requirements Risks**

Risks that might occur if new changes are required in system requirements that need major changes in the system design, like facing

unexpected problems in hardware or software that requires a significant change in the project's requirements.

- **Estimation Risks**

Risks that may drive from the wrong estimation in the system design, implementation, resources and management.

Risk Avoidance

- Taking care when using hardware components and using them according to their specifications.
- Using only compatible software development environment to implement the software.
- Taking care of team member's health during the project development.
- Good estimation and usage of the project budget and resources.
- Good estimation of system requirements.

Risk Management

- Software development environment risks will be handled by the backup of software.
- Including an extra amount of hardware components we all ready have, so when any problem occurs we can find an alternative for the component we lost.
- People risks are handled by using work load balancing on members especially when a member can't perform some of his tasks, then it will be done by other members.

1.7 Report Outline

After choosing the project idea, we start looking for any previous studies, paper, projects related to this idea, which guide us through general idea , problems ,theory and techniques that are considered as abase to go farther to reach objectives.

Image and signal processing is the field that this project is related to, now by looking more closely to this idea two main approaches are clearly observed, first approach is about how to deal with the input image, how to identify the hand, and what methods to use to analysis and classify it. The second approach is the control interface between the workstation and TV.

More details about these two approaches and what hardware devices and software that are need, discussed in chapter two in more details.

Beside the design option for the two approaches such as image capturing, project environment, hands sign and recognition, it is better to use a remote control as a media between workstation and TV or to connect directly wire from the workstation to TV while the workstation lack the responsibility of sending suitable signals to TV, or to use an infrared signal sends from the work station directly to the TV, how we implement, modeling ,simulation, block diagram, technical design, all these topics are covered in chapter three.

Chapter four represents system implementation for both hardware and software, containing flow chart and class diagram, etc.

Chapter five specifies the experimental results for to classifier nearest neighbor and neural network.

Chapter six states our conclusions, expectations and advisements for future work to enhance this system.

Chapter 2

Theoretical background

Chapter Contents:

2.1 Overview

This chapter lists the requirements of the system, and describes the fields of image processing that used to analyze the captured image to extract its feature like edge detection, histograms and Thresholding. Other fields covered in this chapter are classification, neural networks and K-nearest neighbor that recognize the gestures to its related classes. Finally describe the tools that may use in interfacing as infrared and parallel port.

2.2 System Requirements

This section is listing the main requirements that must be met in project in order to set the main services that will be provided.

2.2.1 Functional Requirements

1- Image capturing.
2- Image analysis.
3- Feature extraction.
4- Image classification.
5- TV controlling.

2.2.2 Non-Functional Requirements

1- Robustness

Taking into consideration the light and distance variance, the system should be able to deal with.

2- Reliability

The system will be accurate, reliable, fault tolerant.

3- Performance

The system will provide the user with a fast reaction regarding the whole system speed.

2.3 Machine vision (MV)

Machine vision is the application of computer vision to industry and manufacturing. Whereas computer vision is mainly focused on machine-based image processing, machine vision most often requires also digital input/output devices and computer networks to control other manufacturing equipment such as robotic arms. Just as human inspectors working on assembly lines visually inspect parts to judge the quality of workmanship, so machine vision systems use digital cameras, smart cameras and image processing software to perform similar inspections [11].

Computers do not 'see' in the same way those human beings are able to. Cameras are not equivalent to human optics and while people can rely on inference systems and assumptions, computing devices must 'see' by examining individual pixels of images, processing them and attempting to develop conclusions with the assistance of knowledge bases and features such as Pattern recognition engines. Although some machine vision algorithms have been developed to mimic human visual perception, a number of unique processing methods have been developed to process images and identify relevant image features in an effective and consistent manner. Machine vision and computer vision systems are capable of processing images consistently, but computer-based image processing systems are typically designed to perform single, repetitive tasks, and despite significant improvements in the field, no machine vision or computer vision system can yet match some capabilities of human vision in terms of image[11].

2.4 Image processing

This section covers image processing field that are needed in this project.

2.4.1 Image

A computer image is a matrix (a two-dimensional array) of *pixels*. The value of each pixel is proportional to the *brightness* of the corresponding point in the scene. The matrix of pixels, the image, is usually square as $N \times N$ m-bit pixels where N is the number of points along the axes and m controls the number of brightness values.

2.4.1.1 Color images

Instead of using just one image plane, color images are represented by three intensity components. These components generally correspond to red, green, and blue (the RGB model) although there are other color schemes. For example, the CMYK color model is defined by the components cyan, magenta, yellow and black [2].

2.4.2 Histograms

Histogram is used to graphically summarize and display the distribution of a process data set [8].

A histogram can be constructed by segmenting the range of the data into equal sized bins (also called segments, groups or classes). For example, if your data ranges from 1.1 to 1.8, you could have equal bins of 0.1consisting of 1 to 1.1, 1.2 to 1.3 to 1.4, and so on.

The vertical axis of the histogram is labeled Frequency (the number of counts for each bin), and the horizontal axis of the histogram is labeled with the range of your response variable as shown in Figure 2.1.

You then determine the number of data points that reside within each bin and construct the histogram. The bins size can be defined by the user, by some common rule, or by software methods (such as MatLab).

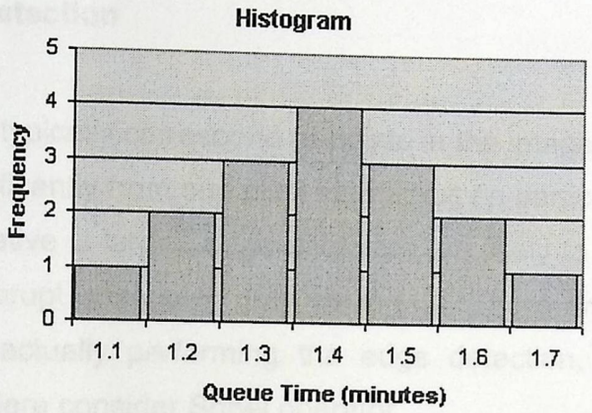


Figure 2.1: An example of histogram

2.4.3 Thresholding

Thresholding is the simplest method of image segmentation. Individual pixels in a grayscale image are marked as "object" pixels if their value is greater than some threshold value (assuming an object to be brighter than the background) and as "background" pixels otherwise [12]. Typically, an object pixel is given a value of "1" while a background pixel is given a value of "0."

The key parameter in thresholding is obviously the choice of the threshold value. Several different methods for choosing a threshold value exist. The simplest method would be to choose the mean or median value, the rationale being that if the object pixels are brighter than the background, they should also be brighter than the average [12]. In a noiseless image with uniform background and object values, the mean or median will work beautifully as the threshold, however generally speaking; this will not be the

case. If the original image is f (i , j) the output (threshold) image, b(i , j), is calculated as shown in equation 2.1.

$$b(i,j)=\begin{cases}1 \ if \ \ f(i,j)\geq T\\ 0 \ if \ \ f(i,j)<T\end{cases} \qquad (2.1)$$

Where T is the threshold. The pixels b(i , j) =1 correspond to the object and the pixel b(i , j) =0 correspond to the background (or vice versa).

2.4.4 Edge Detection

Edges typically correspond to points in the image where the gray value changes significantly from one pixel to the next (in particular, the magnitude of the first derivative is large), edge detectors are likely to pick out areas where there is an abrupt change in gray value [13]. There are several well known methods for actually performing the edge detection, most are based on convolution. Here consider Sobel operator.

2.4.4.1 Sobel operator

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges [10]. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image.

In theory at least, the operator consists of a pair of 3×3 convolution kernels as shown in Figure 2.2. One kernel is simply the other rotated by 90°.

What makes sobel operator suitable for this project that it find orientation of the image pixels while some other operators don't.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 2.2: Sobel convolution kernels

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by equation 2.2:

$$|G| = \sqrt{Gx^2 + Gy^2} \tag{2.2}$$

Typically, an approximate magnitude is computed using equation 2.3:

$$|G| = |Gx| + |Gy| \tag{2.3}$$

This is much faster to compute.

The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by equation 2.4:

$$\theta = \arctan(Gy / Gx) \tag{2.4}$$

2.5 Feature Extraction

Feature extraction involves simplifying the amount of resources required to describe a large set of data accurately. When performing analysis of complex data one of the major problems is the large number of variables generally requires a large amount of memory and computation power or a classification algorithm which overfits the training sample and generalizes poorly to new samples. Feature extraction is a general term for methods of constructing combinations of the variables to get around these problems while still describing the data with sufficient accuracy [14].

2.6 Classification

In general, classification is a technique of distributing objects into classes or categories of the same type, different methods are used, k-nearest neighbor and neural networks are what considered in this project.

2.7 The k-nearest neighbor rule

In application, usually we have a description of a feature vector and we want to find which element of a database best matches that sample. This classification used to associate the appropriate class label (type of feature vector) with the test sample by using the measurements that describe it. One way to make the association is by finding the member of the class (the sample of a known feature vector) with measurements which differ by the least amount from the test sample's measurements. In terms of Euclidean distance, the difference d between the M descriptions of a sample, s, and the description of a known feature vector, k, is

$$d = \sqrt{\sum_{i=1}^{M} (si - ki)^2} \qquad (2.5)$$

If we have M measurements of N known samples of feature vector and we have O samples of each, then we have an M-dimensional feature space that contains the N X O points. If we select the point, in the feature space, which is closest to the current sample, then we have selected the samples nearest neighbor. This is illustrated in Figure 2.3 where we have a two-dimensional feature space produced by the two measures made on each sample, measure 1 and measure 2. Each sample gives different values for these measures but the samples of different classes give rise to clusters in the feature space where each cluster is associated with a single class. In Figure 2.3 we have seven samples of two known feature vector: Class A and Class B depicted by X and O, respectively. We want to classify a test sample, depicted by +, as belonging either to Class A or to Class B (i.e. we assume that the training data contains representatives of all possible classes). Its nearest neighbor, the sample with least distance, is one of the samples of Class A so we could then say that our test appears to be another sample of Class A (i.e. the class label associated with it is Class A). Clearly, the clusters will be far apart for measures that have good discriminatory ability whereas the clusters will overlap for measures that have poor discriminatory ability. That is how we can choose measures for particular tasks. Before that, let us look at how best to associate a class label with our test sample [10].

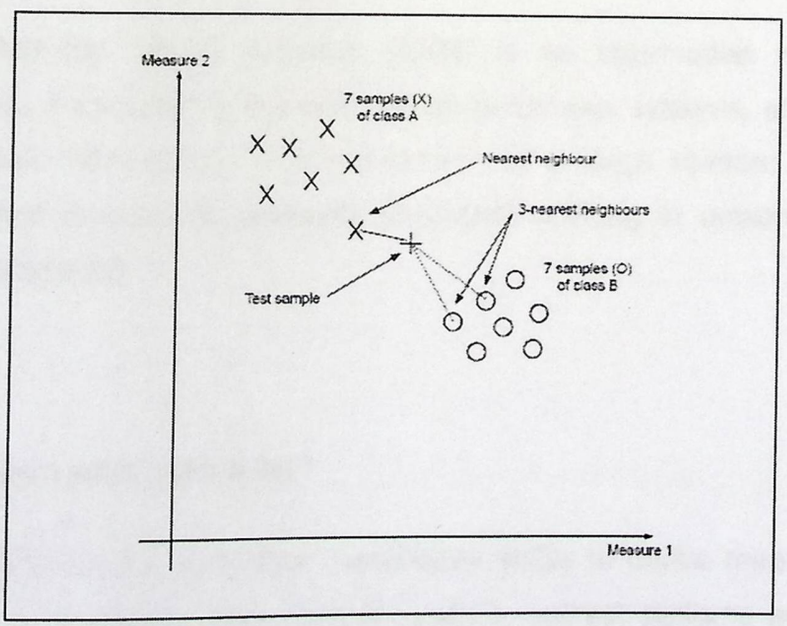


Figure 2.3: Feature space and classification [10]

Classifying a test sample as the training sample it is closest to in feature space is actually a specific case of a general classification rule known as the k-nearest neighbor rule. In this rule, the class selected is the mode of the samples nearest k neighbors. By the k-nearest neighbor rule, for k = 3, we select the nearest three neighbors (those three with the least distance) and their mode, the maximally represented class, is attributed to the sample. In Figure 2.3, the 3-nearest neighbor is actually Class B since the three nearest Samples contain one from Class A (its nearest neighbor) and two from Class B. Since there are two elements of Class B, then the sample is attributed to this class by the 3-nearest neighbor rule. As such, selection from more than one point introduces a form of feature space smoothing and allows the classification decision not to be affected by noisy outlier points. Clearly, this smoothing has greater effect for larger values [10].

2.8 Neural Network

This section will define and discuss neural network, Back-Propagation learning methods, neural network layers and transfer function in details.

2.8.1 What is a Neural Network?

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems [9].

2.8.2 Why use neural networks?

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer

techniques. Neural networks have other advantages that make it different from others; the most important advantage is Adaptive learning; which it is the ability to learn how to do tasks based on the data given for training or initial experience.

Artificial neural networks are made up of interconnecting artificial neurons (usually simplified neurons) designed to model (or mimic) some properties of biological neural networks. Figure 2.4 shows simple artificial neural network architecture. Artificial neural networks can be used to model the modes of operation of biological neural networks, whereas cognitive models are theoretical models that mimic cognitive brain functions without necessarily using neural networks.

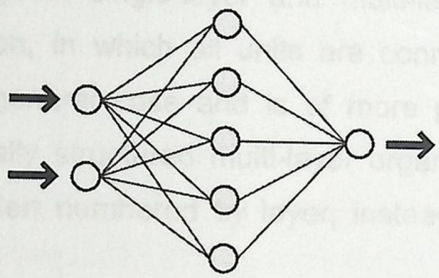


Figure 2.4: A simplified view of an artificial neural network

2.8.3 Network layers

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "input" units is connected to a layer of "hidden" units, which is connected to a layer of "output" units. (See Figure 2.5)

1. The activity of the input units represents the raw information that is fed into the network.
2. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units.

3. The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units.

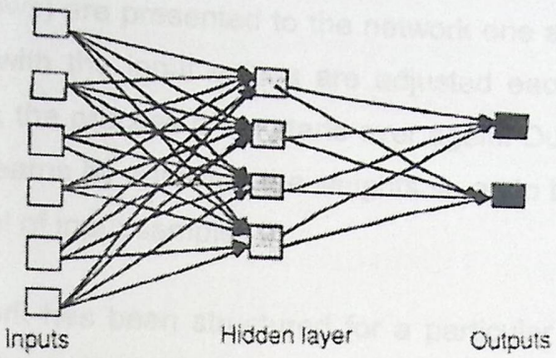


Figure 2.5: general view of ANN

We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering.

2.8.4 Training an Artificial Neural Network

In the training phase, the correct class for each record is known (this is termed supervised training), and the output nodes can therefore be assigned "correct" values -- "1" for the node corresponding to the correct class, and "0" for the others. It is thus possible to compare the network's calculated values for the output nodes to these "correct" values, and calculate an error term for each node [9]. These error terms are then used to adjust the weights in the hidden layers so that, hopefully, the next time around the output values will be closer to the "correct" values.

2.8.4.1 The Iterative Learning Process

A key feature of neural networks is an iterative learning process in which data cases (rows) are presented to the network one at a time, and the weights associated with the input values are adjusted each time. After all cases are presented, the process often starts over again. During this learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of input samples.

Once a network has been structured for a particular application, that network is ready to be trained. To start this process, the initial weights are chosen randomly. Then the training, or learning, begins.

The network processes the records in the training data one at a time, using the weights and functions in the hidden layers, and then compares the resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record to be processed. This process occurs over and over as the weights are continually tweaked. During the training of a network the same set of data is processed many times as the connection weights are continually refined.

2.8.4.2 Feedforward, Back-Propagation

Back-propagation architecture is the most popular, effective, and easy-to-learn model for complex, multi-layered networks. Its greatest strength is in non-linear solutions to ill-defined problems. The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there are just one or two. Some work has been done which indicates that a maximum of five layers (one input layer, three hidden layers and an output layer) are required to solve problems of any complexity. Each layer is fully connected to the succeeding layer.

The training process normally starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times a scaling factor for global accuracy. During the learning process, a forward sweep is made through the network, and the output of each element is computed layer by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function, and the connection weights are normally adjusted. This process proceeds for the previous layer(s) until the input layer is reached as algorithm shows below.

For each training set

- For each output unit j

$$Err_k = O_k(1-O_k)(T_k - O_k) \tag{2.6}$$

- For each hidden unit k

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk} \tag{2.7}$$

- Update the weight

$$w_{ij} = w_{ij} + (l)Err_j O_i \tag{2.8}$$

- Done

2.8.4.3 Transfer Function

The behavior of an ANN (Artificial Neural Network) depends on both the weights and the input-output function (transfer function) that is specified for the units [9]. This function typically falls into one of three categories:

- linear (or ramp)
- threshold
- sigmoid

For *linear units*, the output activity is proportional to the total weighted output.

For *threshold units*, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

For *sigmoid units*, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations.

To make a neural network that performs some specific task, we must choose how the units are connected to one another, and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence.

Teaching a three-layer network to perform a particular task by using the following procedure:

1. Present the network with training examples, which consist of a pattern of activities for the input units together with the desired pattern of activities for the output units.
2. Determine how closely the actual output of the network matches the desired output.
3. Change the weight of each connection so that the network produces a better approximation of the desired output.

2.9 Hardware Components

This section lists in details the component that may be use in this project.

2.9.1 Webcam

Capturing gesture images using webcam with the following features:

Miscellaneous	
Compatibility	PC
Key Features	
Interface Type	USB
Still Image Capture Resolution	640 x 480
Video Capture Resolution	640 x 480
Digital Video Capture Speed	30 frames per second

Table 2.1: Webcam features

2.9.2 Parallel Port

A parallel port is a type of socket found on personal computers for interfacing with various peripherals. It is also known as a printer port.

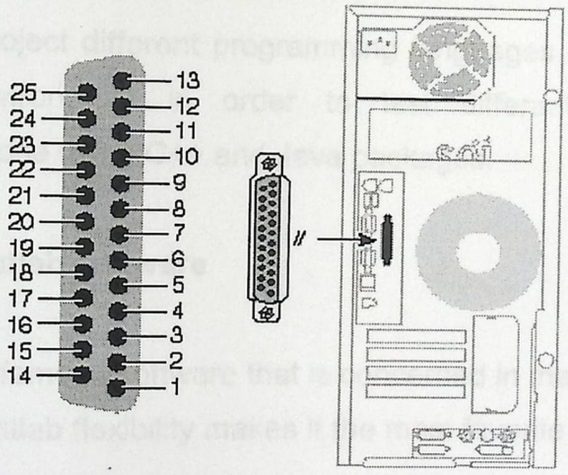


Figure 2.6: Parallel port

2.9.3 Relay

A relay is an electrically operated switch. Current flowing through the coil of the relay creates a magnetic field which attracts a lever and changes the switch contacts. The coil current can be on or off so relays have two switch positions and they are double throw (changeover) switches.

Relays allow one circuit to switch a second circuit which can be completely separate from the first. For example a low voltage battery circuit can use a relay to switch a 230V AC mains circuit. There is no electrical connection inside the relay between the two circuits; the link is magnetic and mechanical. [16]

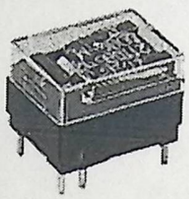


Figure 2.7: Relay

2.10 Software tools

In this project different programming languages used, Matlab is used as testing environment in order to test different algorithms before implementing those using C++ and Java packages.

2.10.1 Matlab software

Matlab is famous software that is concerned in mathematics and matrix manipulation, Matlab flexibility makes it the most favorite software to deal with matrices.

Matlab offers a mass of different toolboxes; one of these toolboxes that we are concerned in it in this project is image processing and neural network toolbox.

2.10.2 Java software

The object oriented way of programming makes it much easier to represent the objects as real life way of thinking, all other languages that not concerned abut object oriented way of programming are at most a procedural way of programming, which make it not as easy for programming as object oriented.

Even thought the libraries that the java provides such as image processing library make it the most convenient one to use as project programming language.

2.10.3 C++

Is a general-purpose programming language. C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features. It is a statically typed, free-form, multi-paradigm, usually compiled language supporting procedural programming, data abstraction, object-oriented programming, and generic programming. [15]

2.11 Summary

This chapter covered different subjects that are related to image and image processing. Machine vision concepts are defined and explained, as well as feature extraction and classification.

The concept of neural networks was also defined in this chapter. Also explained the hardware and the software that were used in our project.

Chapter 3

System Design

Chapter Contents:

3.1 Overview

This chapter goes further in system design description, a block diagram of the system will be included, and all stages of the system design will be briefly discussed.

3.2 Chosen Classes

One of the most important issues of the system is the user shouldn't memorize a lot of gestures, we define as much as possible a few gestures that the user will use, until now have six gestures, in future work classes are able for expansion.

As shown in Figure 3.1 gesture (a) represents the switch TV ON, (b) switch TV OFF, (C) up operation, (d) down operation, and (e) switch between channel and voice modes, (f) no operation.

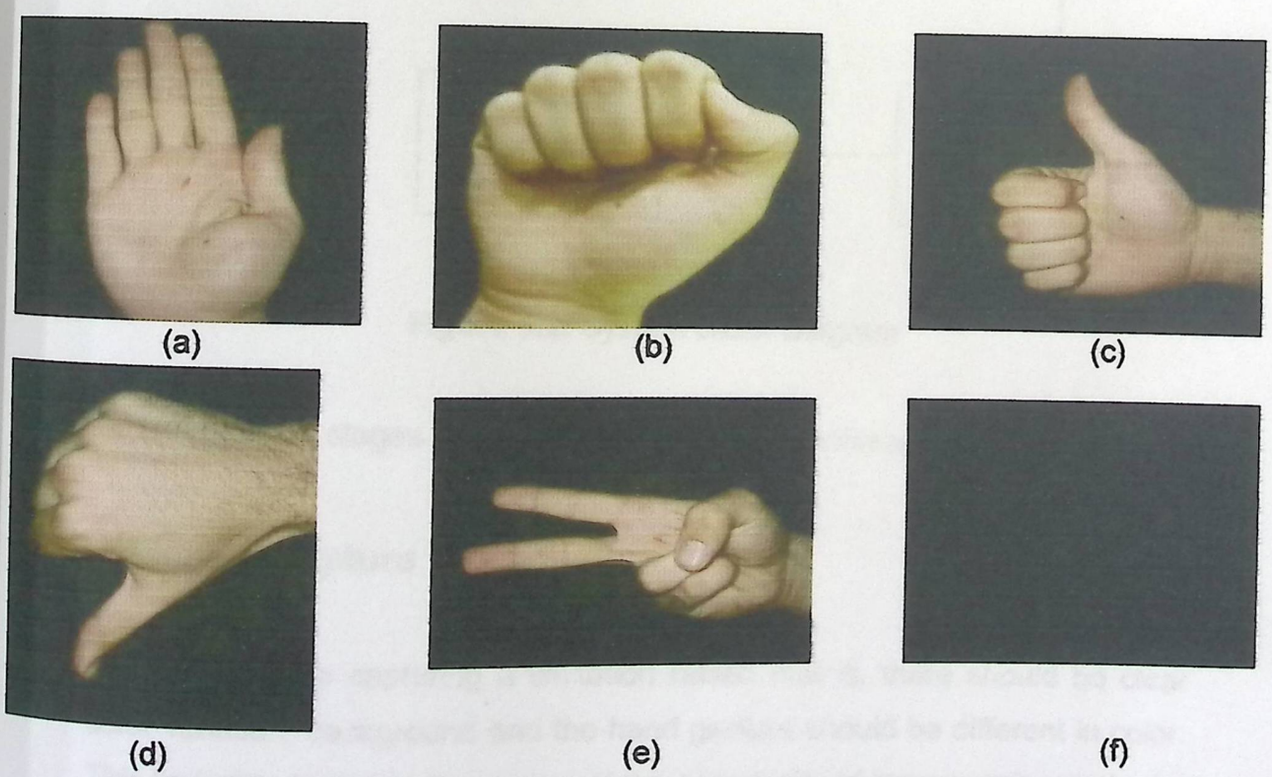


Figure 3.1: Gesture Options (a) On, (b) Off, (c) Up, (d) Down, (e) Switch, (f) No Operation

3.3 System Structure

In Figure 3.2, a general structure for the overall system is shown. In the first stage the image is captured, with some restriction that will be discussed later, and then after the image is captured it is entered to the preprocessing stages that includes image resizing and convert to gray scale. After that the image enters to feature extraction phase in which the image edges are detected, then edge orientation histogram is entered to the classification phase in which the histograms are compared to find the right gesture class. After that a signal is generated to control the TV as the proposed gesture.

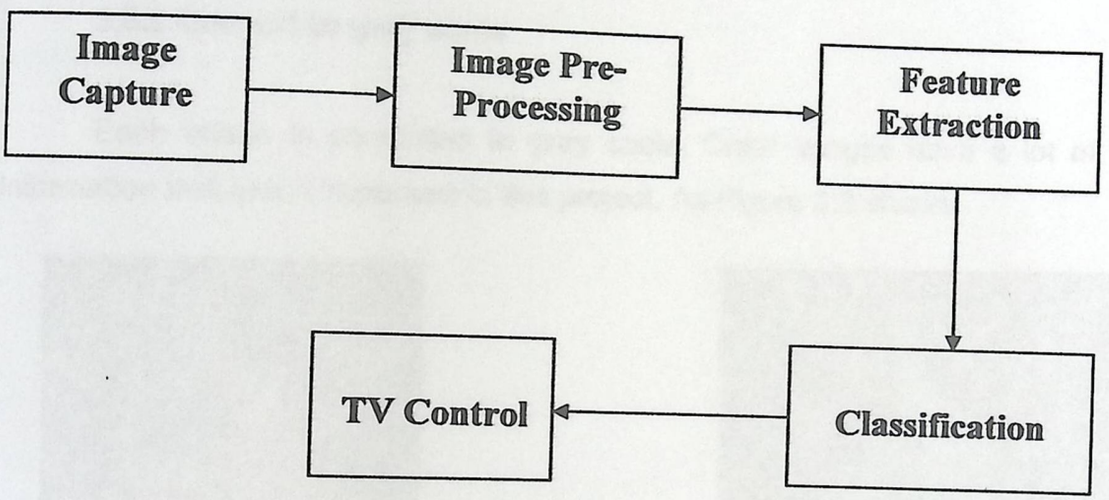


Figure 3.2: System Block Diagram

Details of these stages are given in the following sections.

3.4 Image Capture

For image capturing a limitation raised that is, there should be clear color variation, background and the hand gesture should be different in color. This limitation is due to the nature of the complexity of programming and a lot of related work that are going to face if deal with complex environment.

3.5 Image Pre-Processing

This section will discuss the operations that are necessary before image extractions.

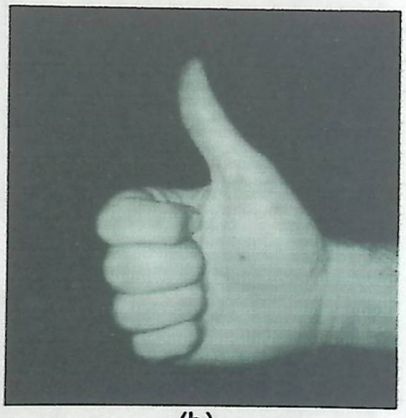
3.5.1 Image Resizing

This project resizes all captured images on 400 x 300 resolutions to have cleared images with clear features; this resizing for each image immediately reduces the size of the data that directly affects the speed of image processing.

3.5.2 Convert to gray scale

Each image is converted to gray scale. Color images have a lot of information that aren't important in this project. As Figure 3.3 shows.

(a)



(b)

Figure 3.3: (a) Original Image, (b) Gray Image

3.6 Feature extraction

This section covers the classification methods that needed in our project.

3.6.1 Edge Detection

As was mentioned in chapter 2, about edge detection, sobel operator used to find the edges, this operator have two Kernels, one is vertical and the other is horizontal, as shown in Figure 2.2. For more information see (section 2.4.4).

Figure 3.4 shows the stages of edge detection, (a) where the image which is converted to gray, in Figure 3.4(b) the image is convolved with the horizontal mask Gx of sobel operator, in Figure 3.4 (c) the image is convolved with the vertical mask Gy of sobel operator, to have the final image with it is edges, a combine between these two images should done, the vertical and the horizontal, to have this calculate the magnitude for them as seen in equation 2.2, this is shown in Figure 3.4 (d).

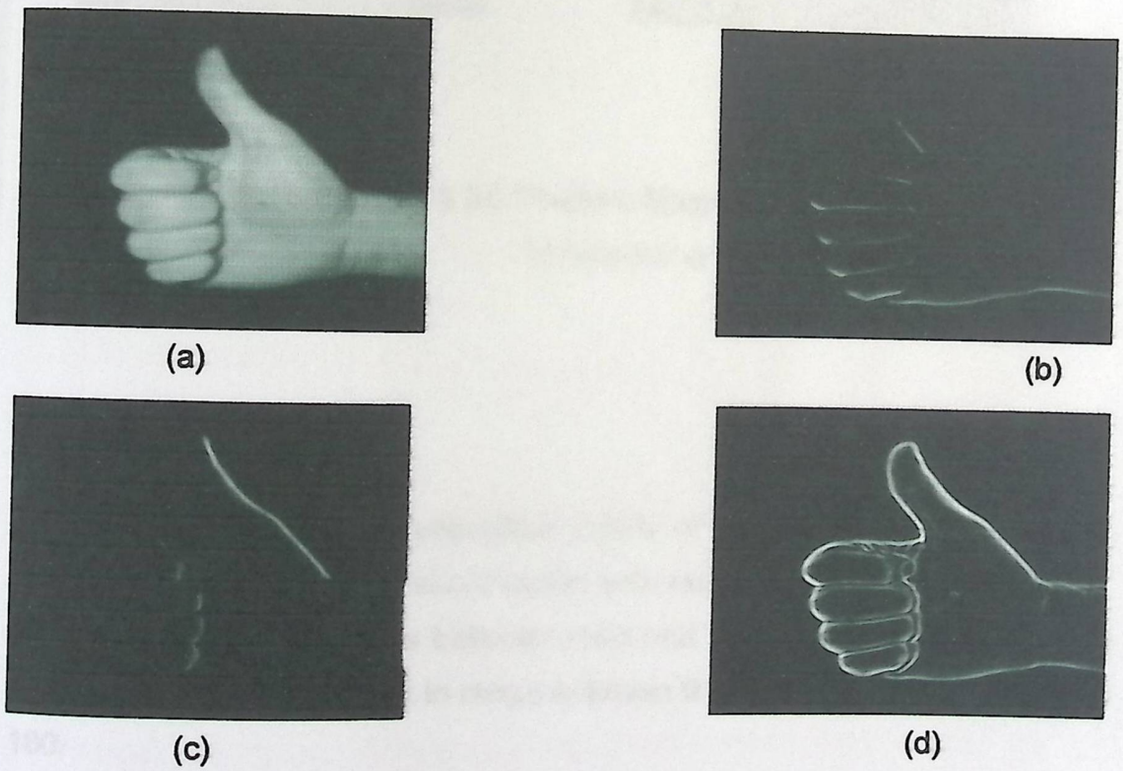
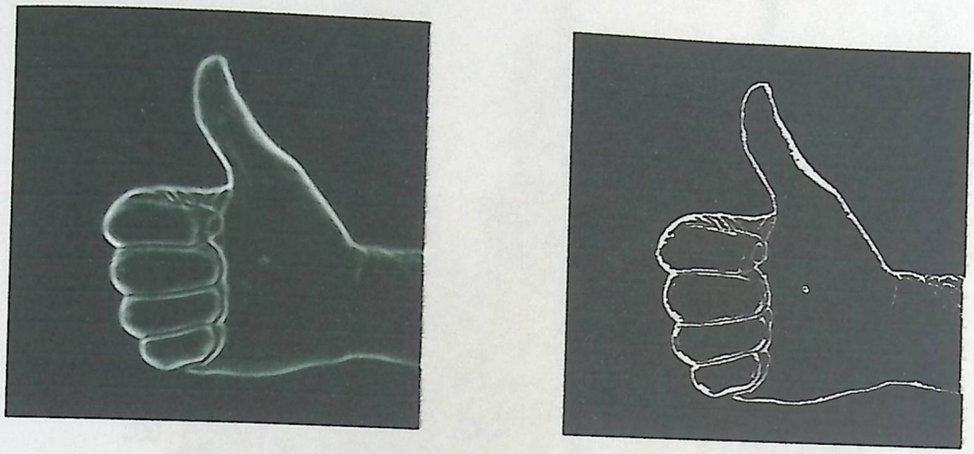


Figure 3.4: Sobel Edge Detector (a) Filtered Image, (b) Horizontally Filtered, (c) Vertically Filtered, (d) Gradient Magnitude

3.6.2 Thresholding

Thresholding (Section 2.4.3) is used to detect the strong edges; in this system threshold value is set to be 4.5 times mean value of gradient magnitude. This value has been determined empirically after several experiments (appendix B), which was found to give the best results. As shown in Figure 3.5.



(a) (b)

Figure 3.5: Thresholding (a) Gradient Magnitude Image, (b) Image after Thresholding

3.6.3 Histogram

In this stage, gradient orientation matrix of the edge is calculated, using the (equation 2.4). This resultant matrix with radian angles, these angles are converted to degree ranges between -180 and 180, to solve the problem of negative values, normalized to range between 0 and 360 in degree by adding 180.

Depending on the value of Thresholding, all values that larger than it in the gradient magnitude image is taken, which represent the strong edges that we need. After determining these strong edges we take the angles that corresponding to it.

The histogram is created for angles, since angle values range between 0 and 360 we chose the vector of the histogram with 36 elements, each bin in this histogram represents a range of 10 angles. Example of the orientation histogram shown in Figure 3.6.

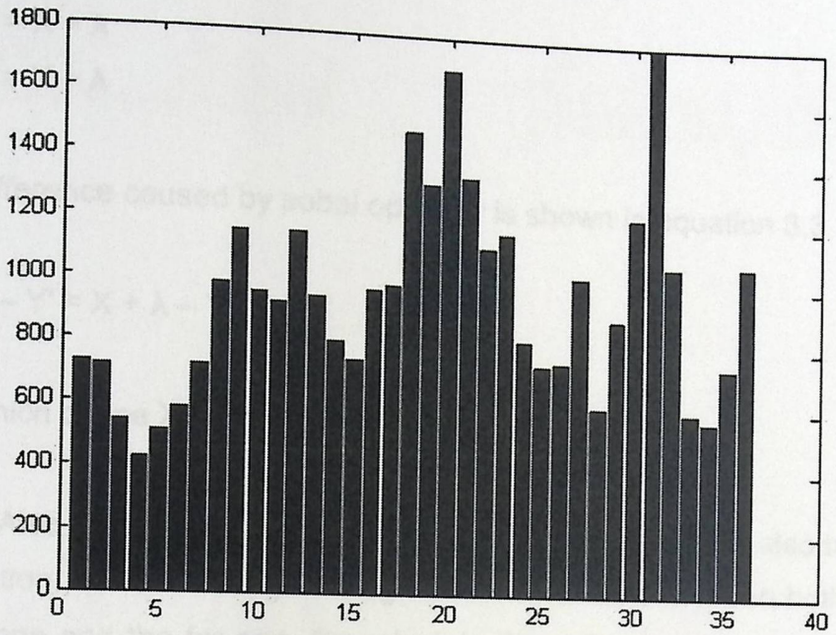


Figure 3.6: Orientation Histogram

3.7 Classification

The edge orientation histogram is the feature that was chosen for each image to distinguish between the different classes that were determined before. Similar features were used by Freeman and Weissman [1].

To examine if the chosen feature is robust, nearest-neighbor and neural network classifiers is used, which explained in (sections 2.7 and 2.8).

Two problems raised should be addressed; change in scale, and lighting condition.

For the light and its effect on image while capturing, different images with different brightness are take. The result shows no big difference between them, which means that no problem in the histograms from different light

images. This is explained, assume λ as light difference on the image pixels, this difference will disappear after applying sobel operator. Suppose that two pixels X and Y in noised image, the value of the pixel will become as in equations 3.1, 3.2.

$$X' = X + \lambda \tag{3.1}$$

$$Y' = Y + \lambda \tag{3.2}$$

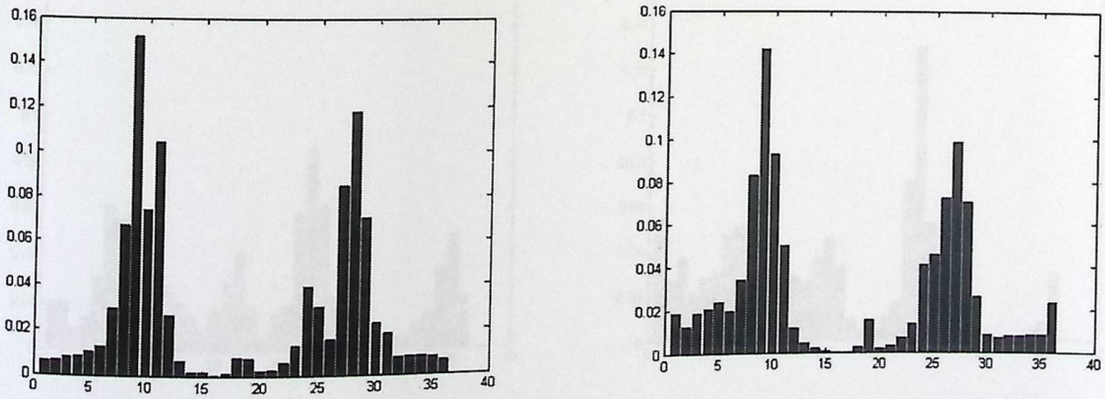
Difference caused by sobel operator is shown in equation 3.3

$$X' - Y' = X + \lambda - Y - \lambda \tag{3.3}$$

Which cause $X' - Y'$ equals $X - Y$.

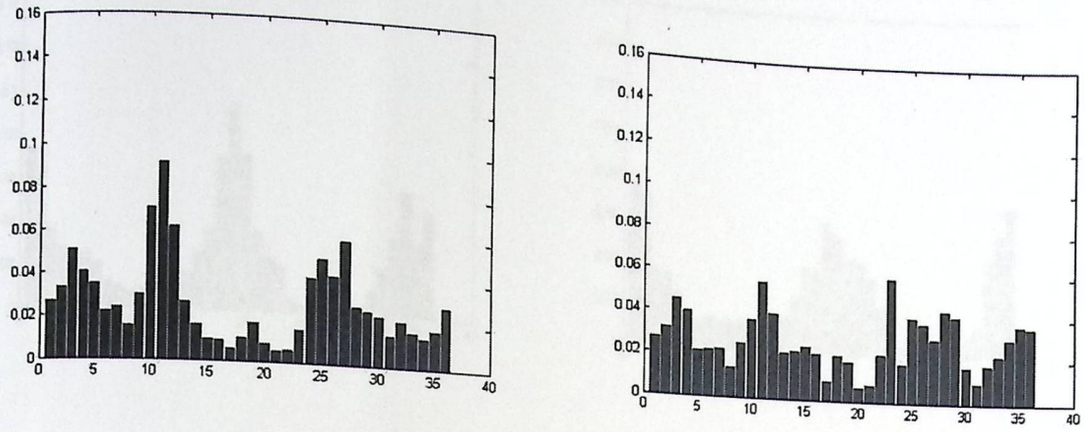
Also for the distance difference from the camera, it is also taken into consideration, solve it through histogram normalization between both images the near one and the far one. By calculate the ratio of each bin value in the histogram to the total summation values of all bins.

In Figure 3.7 shows different histograms for different classes, and for each class we have two different histograms, Figure 3.8 shows the difference between different classes histograms.

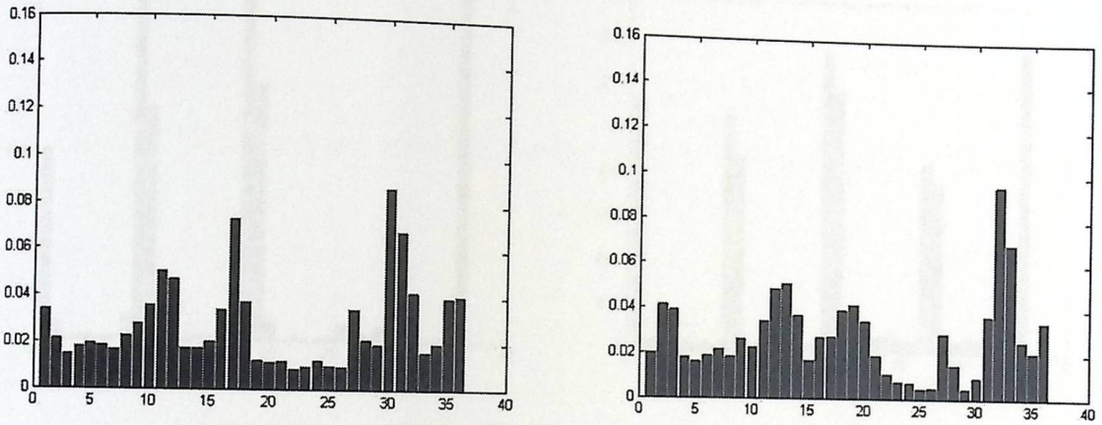


(a)

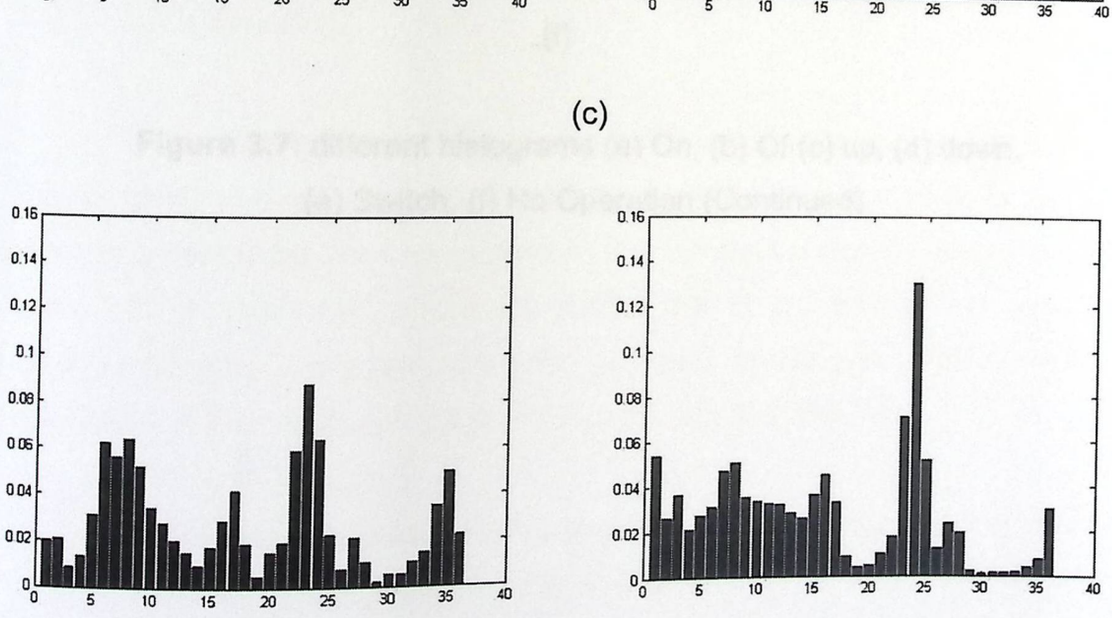
Figure 3.7: different histograms (a) On, (b) Off, (c) up, (d) down, (e) Switch, (f) No Operation



(b)

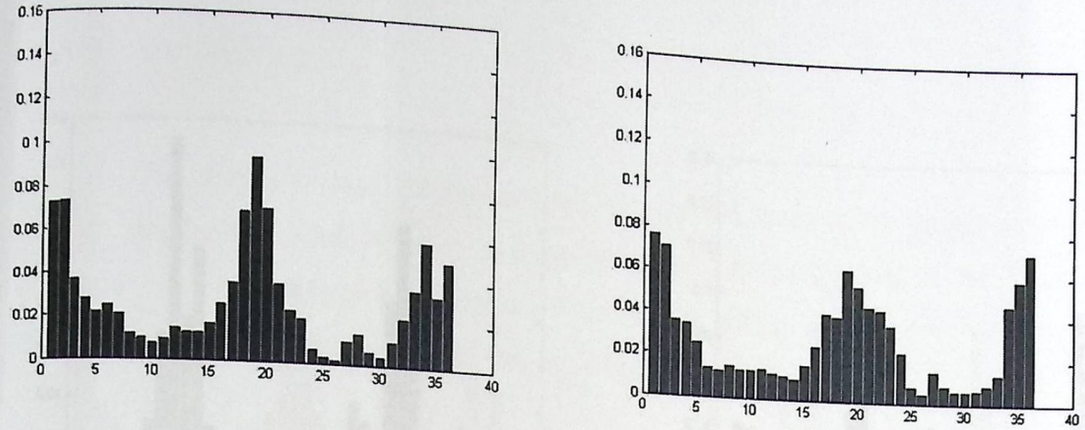


(c)

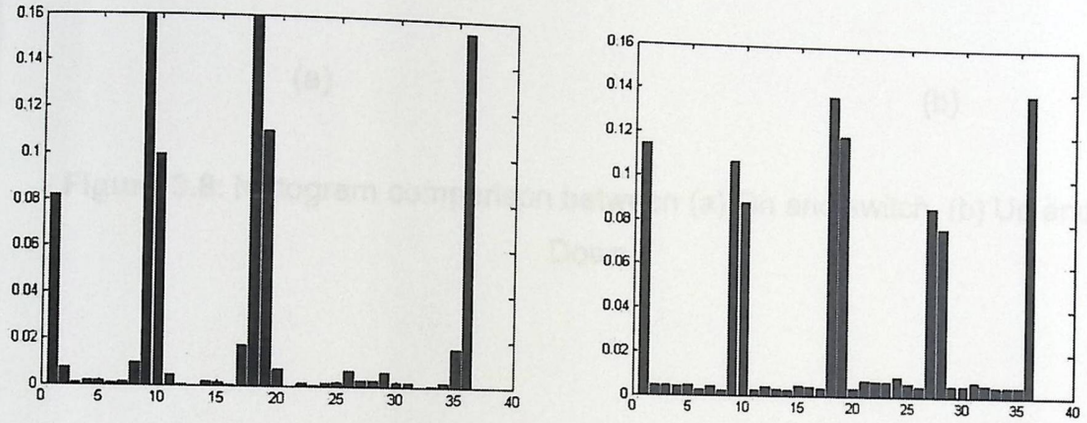


(d)

Figure 3.7: different histograms (a) On, (b) Off, (c) up, (d) down,
(e) Switch, (f) No Operation (Continued)

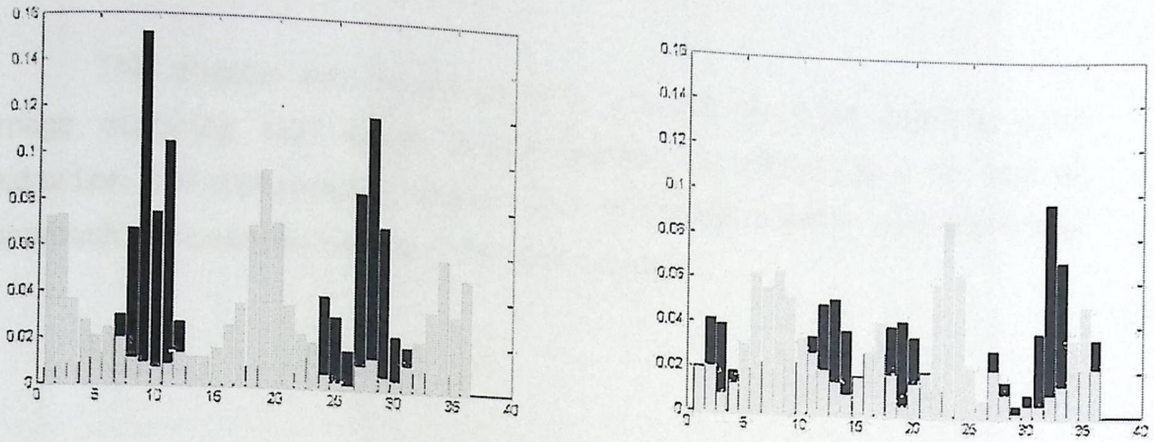


(e)



(f)

Figure 3.7: different histograms (a) On, (b) Of (c) up, (d) down, (e) Switch, (f) No Operation (Continued)



(a) (b)

Figure 3.8: histogram comparison between (a) On and switch, (b) Up and Down

3.8 TV Control

This stage has three assumptions. The First approach is to connect the computer to the TV directly, in which the signals are transferred to the TV through a cable, the Second approach is to connect a remote control to the computer, in which the remote will control the TV in order of the signal it receives from the computer, the Third approach is using an infrared device connected to the computer, that transmits the signal to the TV.

In this project second approach is the choice for signal transmission and TV control, through programming and parallel port, this is the simplest approach, mean while it is cheapest and supports the portability for the project.

3.9 Summary

This chapter has seven sections, second and third sections cover image capturing and image pre-processing for preparing it to feature extraction that discussed in section four, while classification and interfacing approach discussed in details in the rest sections.

Chapter 4

Implementation

Chapter Contents:

4.1 Overview

This chapter focuses on the implementation phases of the project. For software showing Use Case diagram, Class Diagram, Sequence Diagram, Flow Chart Diagram and Pseudo Code. Finally a description of the interfacing with TV is given.

4.2 Software implementation

This section describes the system core, what it dose and how it works by viewing a set of models that describe the actions of the software different components in addition to a pseudo code section .

4.2.1 General Description

The software starts from the moment the image is captured till the moment the TV is controlled.

When the is image captured, its Gradient Orientation histogram is calculated as a feature, then classifying it using neural network and at the end, a signal is generated to an interface circuit that controls the TV.

4.2.2 System specifications:

There are some specifications related to the project that should be known before proceeding into details.

These specifications are:

- A delay time between captured images is 3 seconds.

- There should be a clear color variation, background and the hand gesture should be different in color. The project deals with black background and doesn't deal with complex environment.

- The project deals only with the right hand gestures to control the TV, since that the system was trained to deal with right hand gestures.

- When the system is run up, two windows appear. The first window is a video player that captures the environment as a video and the other is a snapshot window that shows the captured images from the player window.

- The processing time needed starting from the image capture until the TV control is about 800ms, and as maximum 870ms.

4.2.3 Software model

This section shows the models that specify the actions of the software and how it handles each event.

4.2.3.1 Use Case Diagram

Figure 4.1 shows system Use Case diagram this diagram show a set of activities within a system which presented from the point of view of the associated actors (those actors interacting with the system like core system) . the basic component of the use case diagram is use-cases and actors ,this use-cases must deliver a value to an actors while the aggregate of all use-cases is the system complete functionality

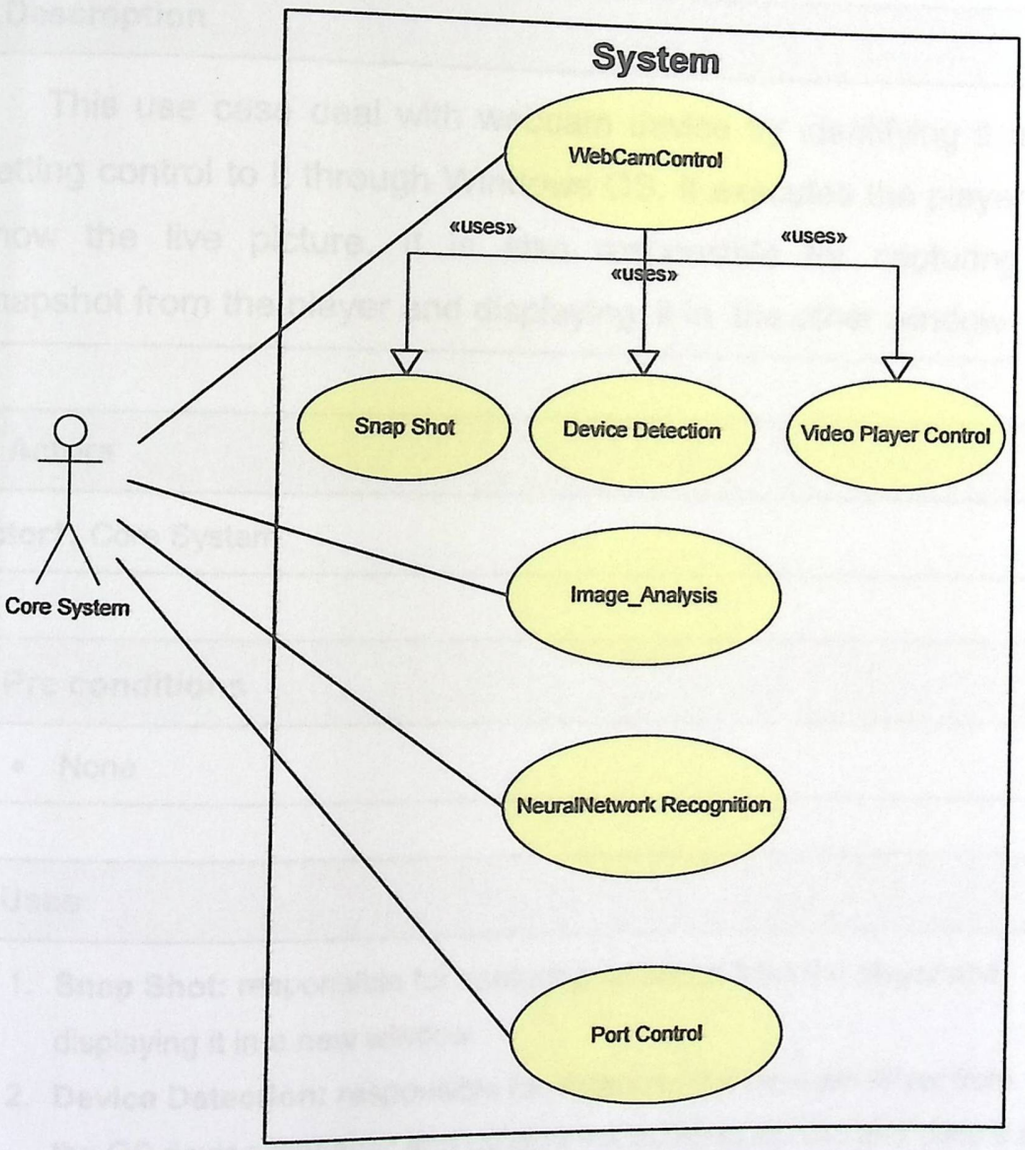


Figure4.1: Use Case diagram

4.2.3.1.1 Use Case Diagram Specification

This section shows the description for each use case as shown in figure 4.1.

Project name:	Hand Gesture TV
Use-case name:	Web Cam Control
Unique identifier:	1

1.Descreption
This use case deal with webcam device by identifying it and getting control to it through Windows OS. It executes the player to show the live picture. It is also responsible for capturing a snapshot from the player and displaying it in the other window

2. Actors
Actor1: Core System

3. Pre conditions
• None

4. Uses
1. **Snap Shot:** responsible for capturing an image from the player and displaying it in a new window. 2. **Device Detection:** responsible for detecting the webcam driver from the OS device manager and gaining the ability to access and control it. 3. **Video Player Control:** responsible for handling the video player format.

Project name:	Hand Gesture TV
Use-case name:	Image Analysis
Unique identifier:	2

1.Descreption
This use case deals with the captured images by analyzing it and extracting its features.

2. Actors
Actor1: Core System

3. Pre conditions
• Web Cam Control

4. Uses
• None

Project name:	Hand Gesture TV
Use-case name:	Neural Network Recognition
Unique identifier:	3

1.Descreption
This use case recognizes the extracted features using neural network architecture and classifies it to one of system classes.

2. Actors
Actor1: Core System

3. Pre conditions
• Web Cam Control • Image Analysis

4. Uses
• None

Project name:	Hand Gesture TV
Use-case name:	Port Control
Unique identifier:	4

1.Descreption
This use case handles the interface circuit by sending the needed signals related to the class that the captured image was classified into.

2. Actors
Actor1: Core System

3. Pre conditions
• Web Cam Control • Image Analysis • Neural Network Recognition

4. Uses
• None

4.2.3.2 Flow Chart Diagram

Figure 4.2 shows overall System Flow Chart Diagram

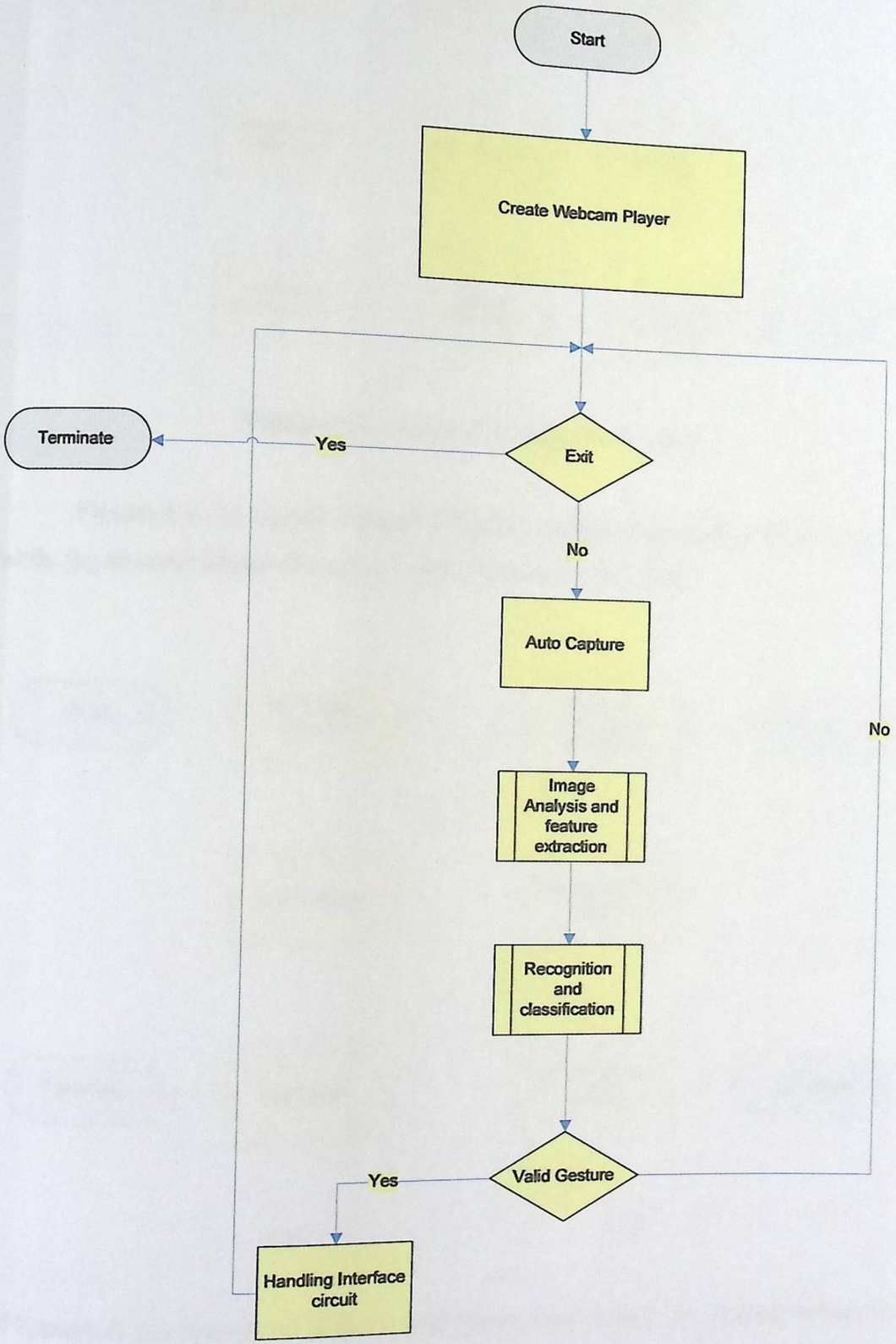


Figure4.2: Overall System Flow Char

Figure 4.3 shows Image Analysis Flow Chart

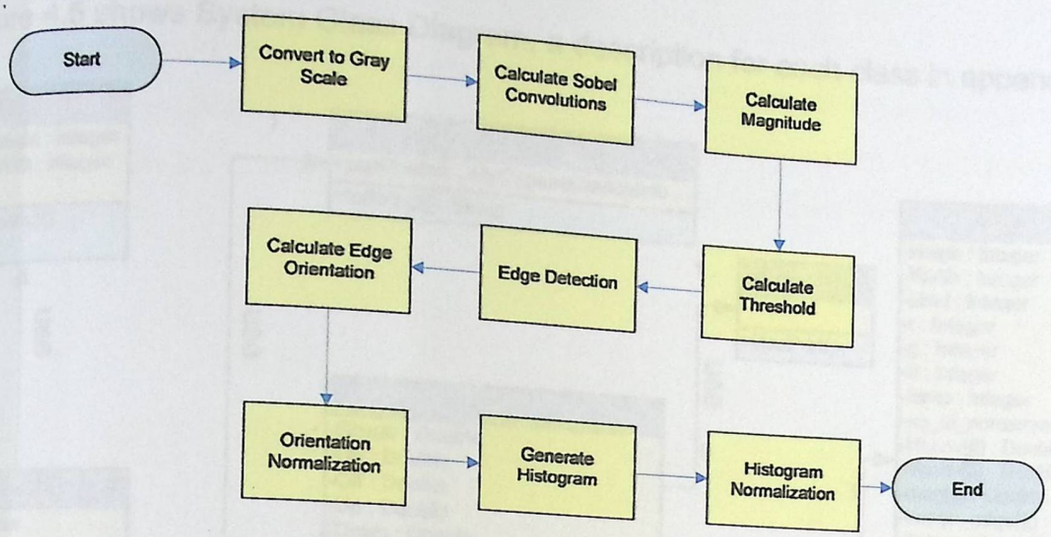
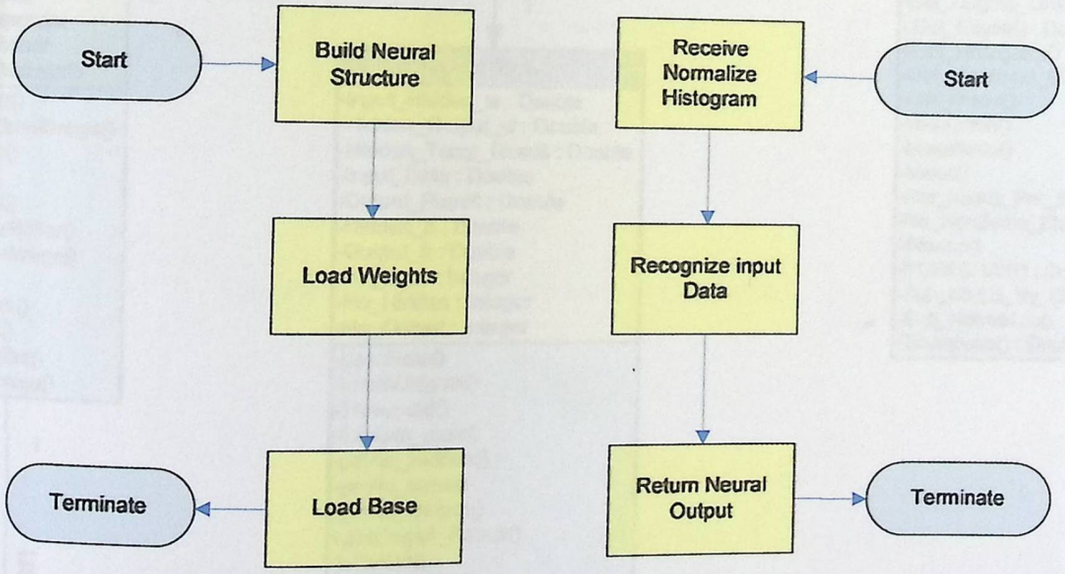


Figure4.3: Image Analysis Flow Chart

Figure 4.4 (a) shows Neural Network Object Initialization Flow Chart, while (b) shows Neural Network Data Process Flow Chart



(a) (b)

Figure4.4: (a) Neural Network Initialization Flow Chart, (b) Neural Network Data Process Flow Chart

4.2.3.3 Class Diagram

Figure 4.5 shows System Class Diagram, a description for each class in appendix A.

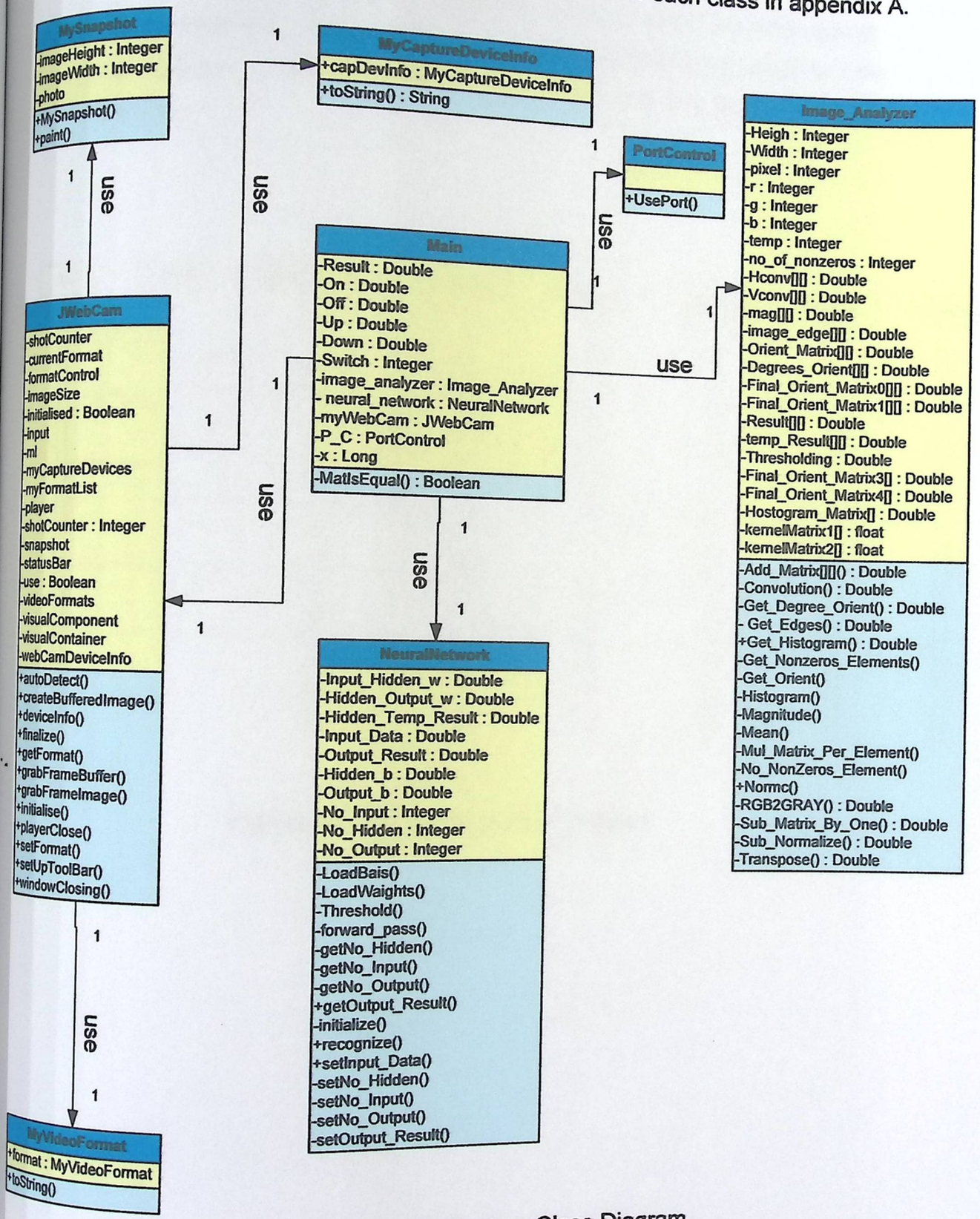


Figure 4.5: System Class Diagram

4.2.3.4 Sequence Diagram

Figure 4.6 shows System Sequence Diagram, which show the system object interaction (i.e. Main object send the returned histogram result from the ImageAnalyzer to NeuralNetwork object to identify the class that this gesture related to).

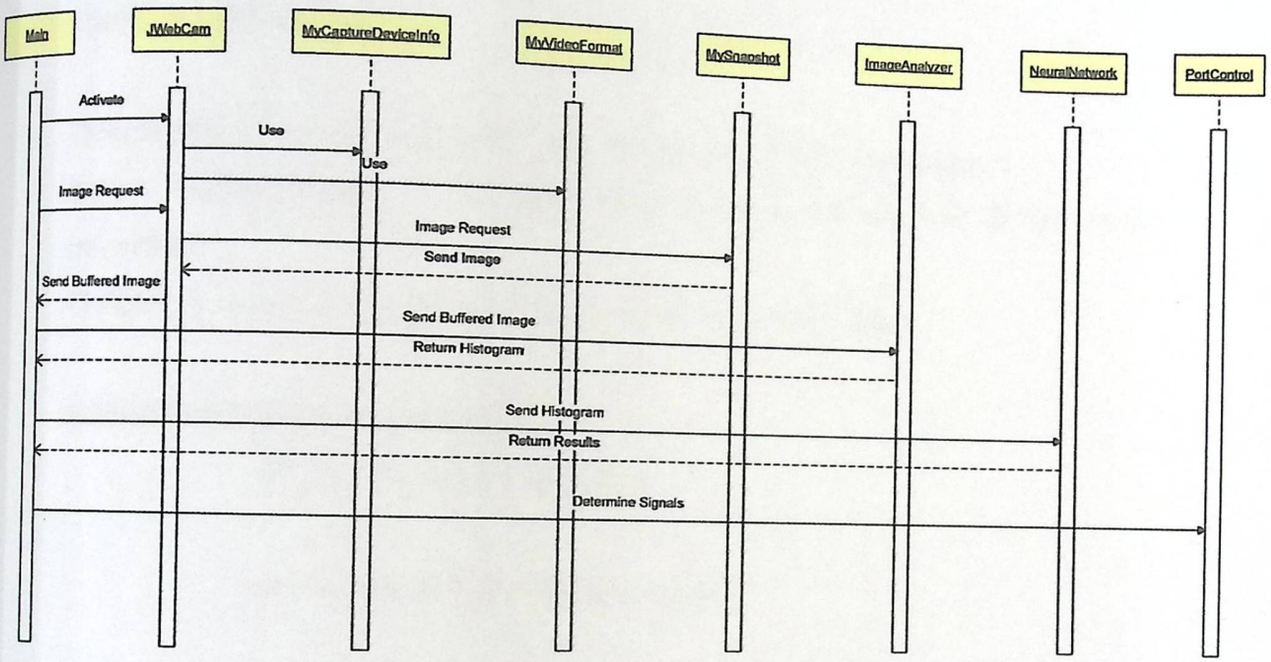


Figure 4.6: System Sequence Diagram

4.2.3.5 Pseudo Code

This section describe the main functionality of the image analysis, listing the pseudo code of converting a color image to a gray one, histogram construction, data normalization (neural network input data), convolution, gradient magnitude calculation, getting the angels in degree value from orientation.

List 4.1: RGB2GRAY

Description: convert colored image pixels from RGB to gray scale.
Input: BufferedImage – a buffer that contains the colored image to be converted.
Output: a matrix contains the image converted pixels value.

```
RGB2GRAY (BufferedImage) {
        Rmatrix (Height) (Width)

        for i=0 to BufferedImage height

            for j=0 to BufferedImage width {

                pixel = getRGBPixal(i, j)
                r = Get Red value from pixel
                g = Get Green value from pixel
                b = Get Blue value from pixel
                Rmatrix (i) (j) = (0.2989 * r + 0.5870 * g + 0.1140
                                    * b) / 255

            }
        Return Rmatrix
}
```

List 4.2: Histogram

Description: build a gradient orientation histogram of 36 bins, each bin present a count of orients appeared in an interval of length 10.
Input: Matrix – a matrix contains the sobel operator generated orients for the captured image.

Size – the size of Matrix.

Bins – number of histogram bins.

Output: a matrix contains the histogram values.

```
Histogram (Matrix, Size, bins) {

    Rmatrix (bins)

        for k= 0 to bins
            Rmatrix(k) = 0

        for i = 0 to matrix Size
        {
            if (Matrix (i) equal 360)
                add one to Rmatrix (35)
            else
                Add one to Rmatrix (Matrix (i) / 10)
        }

        for i = 0 to bins
            Rmatrix (i) = Rmatrix (i) /Size
        Return Rmatrix
}
```

List 4.3: Normc

Description: matrix column normalization.
Input: Matrix – a histogram matrix
Output: a matrix contains the normalized histogram values.

```
Normc (Matrix) {

        Rmatrix (Matrix length)
        sum = 0
        for  i = 0 to matrix length
                sum =sum + power of Matrix(i)

        temp = square of (1 / sum)

        for i = 0 to Matrix length

                Rmatrix (i) = temp * Matrix (i)

        Return Rmatrix
    }
```

List 4.4: Convolution

Description: a convolution of any matrix with any 3X3 kernels.
Input: Matrix – a matrix of gray value from the captured image.
Height – Matrix height.
Width – Matrix Width
Kernel – a matrix of 3X3 that will convoluted with Matrix.
Output: a matrix contains the result of the convolution.

```
Convolution (Matrix, Height, Width, Kernel) {

        Rmatrix (Height - 2) (Width - 2)

        for  i = 1 to Height - 1
                for  j = 1 to Width - 1

                Rmatrix(i - 1)(j - 1) =  Matrix(i - 1)(j - 1)  *  Kernel(0) +
                                         Matrix(i - 1)(j)      *  Kernel(1) +
                                         Matrix(i - 1)(j + 1)  *  Kernel(2) +
                                         Matrix(i)(j - 1)      *  Kernel(3) +
                                         Matrix(i)(j)          *  Kernel(4) +
                                         Matrix(i)(j + 1)      *  Kernel(5) +
                                         Matrix(i + 1)(j - 1)  *  Kernel(6) +
                                         Matrix(i + 1)(j)      *  Kernel (7)+
                                         Matrix(i + 1)(j + 1)  *  Kernel(8)
    }

    Return Rmatrix;
}
```

List 4.5: Magnitude

Description: calculate the magnitude of any two symmetric matrixes.

Input: Matrix1 – first matrix.

Matrix2 – second matrix.

Height – Matrix1 Height

Width – Matrix1 Width

Output: a matrix contains the calculate magnitude.

```
Magnitude (Matrix1, Matrix2, Height, Width) {

        Rmatrix (Height) (Width)

                for  i = 0 to Height
                        for j = 0 to Width
                                Rmatrix (i) (j) = absolute value (Matrix1 (i) (j)) +
                                                  absolute value (Matrix2 (i) (j))
Return Rmatrix
}
```

List 4.6: Get_Degree_Orient

Description: convert radian matrix value to degrees.
Input: Matrix – a matrix contains radians values.
Height – Matrix Height
Width – Matrix Width
Output: a matrix contains the degrees values.

```
Get_Degree_Orient( Matrix,  Height,  Width) {

        Rmatrix (Height) (Width)

for i = 0 to Height
for  j = 0 to Width
Rmatrix (i) (j) = (Matrix (i) (j) / ((2 * PI) / 360)) + 180

Return Rmatrix
}
```

4.3 Interface circuit

Controlling TV using Workstation should be through an interfacing circuit that is responsible to control remote control.

The interfacing circuit is connected to the Workstation through a parallel port; this circuit received data signals and sends the appropriate signals to the TV.

The interfacing circuit consists of the following components:

- Resistors
- Transistor
- Relay.

Figure 4.7 shows the circuit diagram, it consist of five data lines that transfer data from the Workstation to the circuit through parallel port, transistors to generate suitable current to run the relays, resistors connected to transistors that protect them from fail down.

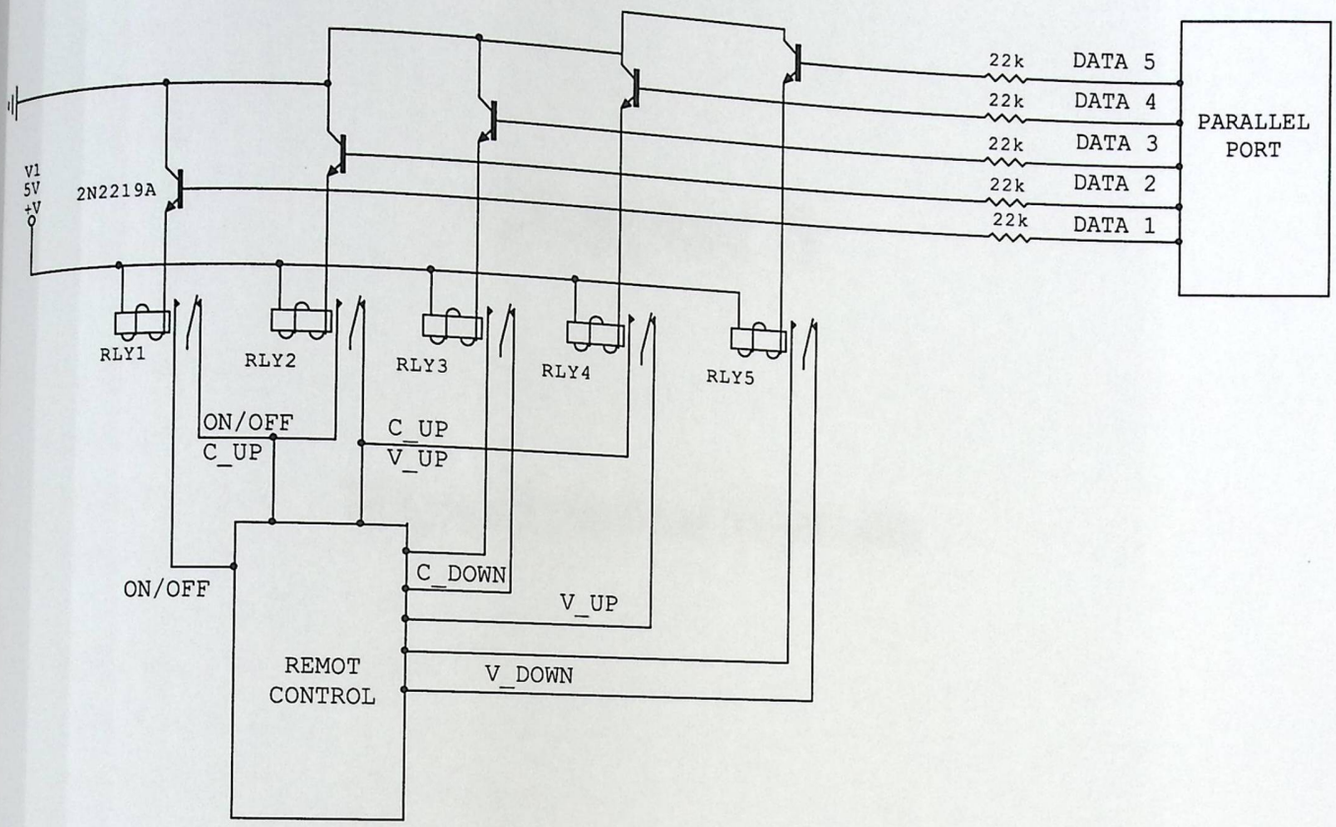


Figure 4.7: Interface circuit diagram

4.4 Summary

This chapter introduced the implementation in details for both the hardware and software; in software it lists a pseudo code of the methods used in image analysis, class diagram, and its functionality.

And for hardware a circuit description is mentioned.

Chapter 5

Experimental Results

Chapter Contents:

5.1 Overview

This chapter introduces several experiments that were carried out and discusses the results that we got using different methods and parameters. Each experiment compared to other by Recognition Rate (%), which is the ratio of Recognized Samples to Misclassified Samples times 100%.

5.2 Data Set

The data set is used for training and testing the system, which consists of images for all six gestures and one no operation class shown in Figure 3.1. Many samples for each gesture were taken from different persons. Different experiments were done for simulating the system as shown below.

The samples were taken from different distances with variations in brightness, so that can help to examine the capabilities of the features extracted. For example, some of the samples that were taken for the gesture *switch* are shown in Figure 5.1.

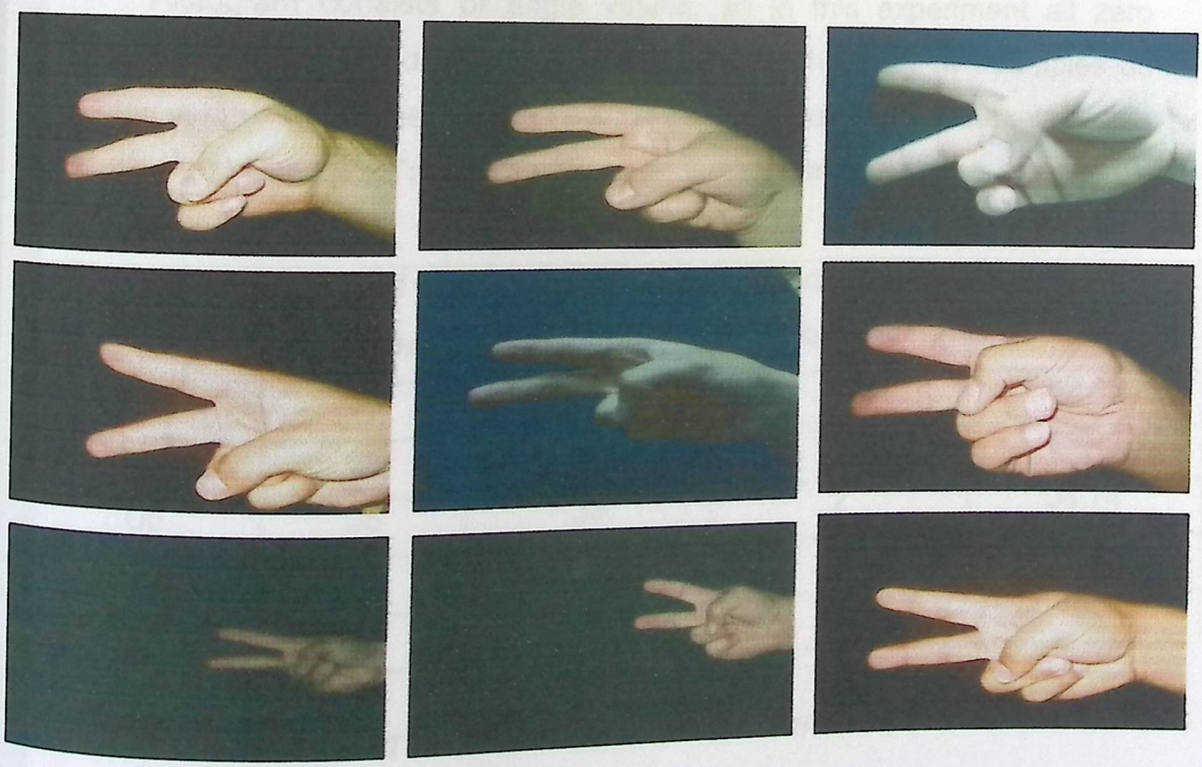


Figure 5.1: Some of samples taken for gesture switch

5.3 Experimental Results using Nearest Neighbors

Many experiments were conducted during the development of this project to examine its feature extraction scheme. This section will discuss some of these experiments, and show there results.

To go on with this chapter, two concepts should be clarified, training and testing.

For any experiment, there is data that is used to teach the system to behave as what the system is expected to do. This process is called training. To test the system after the training, there should be new data, that wasn't used before, to test the systems behavior. This process is called testing.

We start with the experiments that use the nearest Neighbors as a classifier. Then in section 5.4 we introduce the results that were obtained using neural network classifier

5.3.1 Non-zeros Experiment

Histogram consists of angle values, so in this experiment all zero values of angles were discarded. This experiment has just two samples as training samples per class, using nearest neighbor rule that was explained in (Section 2.7) with k=1. Table 5.1 shows the experiment results.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	35	29	6	82.86
OFF	35	32	3	91.43
Up	35	33	2	94.74
Down	35	29	6	82.86
Switch	35	35	0	100

Table 5.1: Results for Non-zeros experiment

Results are very good, but one should note that scale changing samples are not used.

5.3.2 Strong zeros Experiment

After determining the strong edges as mentioned in section (3.6.3) we take the angles that correspond to it, which contain zero values; these zeros are still in the edge orientation histogram.

As the pervious experiment, using nearest neighbor rule that explained in (Section 2.7) with k=1 and two samples as training samples per class. Table 5.2 shows the experiment results.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	35	29	6	82.86
OFF	35	32	3	91.43
Up	35	33	2	94.74
Down	35	29	6	82.86
Switch	35	35	0	100

Table 5.2: Results for using zero angles

Results are the same as previous experiment, since that all what this experiment does is taking into consideration the first bin in the histogram, which seems by experiment it doesn't make a difference, knowing that, this experiment and the previous one, neglect the scale changes.

5.3.3 Different scales

This experiment takes all strong zero angel values in histogram, and takes many samples with different scales.

Using nearest neighbor rule with k=1, and two samples for training purpose, results are shown in table 5.3.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	40	33	7	82.50
OFF	40	33	7	82.50
Up	43	41	2	95.34
Down	41	31	10	75.60
Switch	40	40	0	100

Table 5.3: Results for using different scale gesture

Notice that the recognition rate is less than that in the previous experiments. This is because the images that were used for testing are with scale changes, which cause this drop in the results as shown in Table 5.4

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	4	3	1	75
OFF	4	1	3	25
Up	4	4	0	100
Down	4	0	4	0
Switch	4	4	0	100

Table 5.4: Results using only the samples with scale changes

This table shows the recognition rate for the images with scale changes only.

5.3.4 Reducing the effect of Scale changes

The problem of scale changes is alleviated as mentioned in section (3.7). Using nearest neighbor rule with k=1, and two samples for training purpose, Table 5.5 shows the result of this experiment.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	40	28	12	70
OFF	40	36	4	90
Up	43	40	3	93
Down	41	32	9	78
Switch	40	40	0	100

Table 5.5: Results after using scale solution

Notice that results here were expected to improve, but in contrast the results of some classes are worse. This is because the solution of the scale images problem change the histogram by reducing the bins range that affect old picture recognition. Table 5.6 shows the scaled image effective result.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	4	2	2	50
OFF	4	4	0	100
Up	4	4	0	100
Down	4	1	3	25
Switch	4	4	0	100

Table 5.6: Results using only the samples with scale changes

5.3.5 Training Database expansion

The main difference in this experiment is that the samples for training purpose are expanded to four samples instead of two samples. Using nearest neighbor rule with k=1. Results are listed in Table 5.7.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	40	32	8	80
OFF	40	34	6	85
Up	43	40	3	93
Down	41	36	5	87.80
Switch	40	40	0	100

Table 5.7: Results after database expanding

This table shows some unexpected results in which the recognition rate of some class became lower. This is because the expansion of the training database makes some test images be misclassified to the newly added training images in the database.

Comments on the results

As can be noticed from the results that are obtained, the switch gesture is always recognized with 100% recognition rate. The reason is that the switch gesture has an orientation histogram that is different from all other classes, as can be noticed in Figure 3.7 (e).

5.4 Experimental Results using Neural Networks

There are many experiments conducted using a neural network structure, which consists of three layers; the Input layer with 36 neurons, the hidden layer with 72 neurons, and the output layer with 6 neurons.

The values at the output layer neurons satisfy the input gesture feature to which class it should classified (i.e. if the out put is 1 0 0 0 0 0 then the gesture is classified to On class).

There will be different experimental results shown here using different training methods and different parameters.

5.4.1 Using 40 samples

Table 5.8 shows the results obtained after 800 epochs, backprorogation training method is used, and using both column matrix normalization and reduction of scale effect. 30 samples are used for training and 10 samples are used for testing. The threshold value is set to 2.5 times the mean value of the gradient magnitude matrix.

Class	Training samples Test (%)	Testing sample Test (%)
ON	100	86.6667
OFF	100	80
Up	100	90
Down	100	80
Switch	100	100

Table 5.8: Results obtained using 40 samples

This result is bad because recognition rate for some class is 80%, we trying to solve this problem by increasing the number of samples as shown in 5.4.2.

5.4.2 Increasing the Number of Samples

Table 5.9 shows the results obtained after 800 epochs, and as the previous experiment (5.4.1), both matrix column normalization and reduction of scale effect were used. In this experiment 105 samples were used; 70 samples for training and 35 samples for testing. The threshold value here is set to 4.5 times the mean value of the gradient magnitude matrix.

Class	Training samples Test (%)	Test sample Test (%)
ON	100	97.14
OFF	98.57	88.57
Up	100	100
Down	98.57	94.28
Switch	98.57	91

Table 5.9: Results after increasing samples

Increasing the number of samples lead to better results. That refers to two reasons. First, training samples are increased, and second, is threshold value of 4.5 times the mean value of the gradient magnitude matrix enhances the recognition rate greatly.

5.4.3 Adding No Operation Class

A six class (no operation) is added; table 5.10 shows the results with the newly added that obtained after 800 epochs, and both matrix column normalization and reduction of scale effect were used. In this experiment 105 samples were used; 70 samples for training and 35 samples for testing. The threshold value here is set to 2.5 times the mean value of the gradient magnitude matrix.

Class	Training samples Test (%)	Test sample Test (%)
ON	100	97.1429
OFF	98.58	88.5714
up	98.58	94.2857
down	100	88.5714
switch	98.58	77.1429
no operation	100	100

Table 5.10: Results after added no operation class

This result is bad which is expected because the effect of the new samples of the added class, this problem is solving in the next experiment.

5.4.4 Final Experiment

Table 5.11 shows the results obtained after 800 epochs, and both matrix column normalization and reduction of scale effect were used. As the previous experiment, there are 70 samples used for training and 35 samples used for testing. The used threshold value is 4.5 times the mean value of the gradient magnitude matrix.

Class	Training samples Test (%)	Test sample Test (%)
ON	100	100
OFF	98.57	88.57
Up	95.57	94.28
Down	98.57	91.42
Switch	98.57	88.57
No Operation	97.14	100

Table 5.11: Results obtained from 630 samples after added no operation class

This is the best result that we got from all experiments. To see all experiments with their results check appendix B.

5.4.5 Nearest neighbor comparison experiment

This experiment has just four samples as training samples per class, using nearest neighbor rule that is explained in (Section 2.7) with k=1. Table 5.12 shows the experimental results with 150 samples in the testing database.

Gesture	Total samples	Recognized Samples	Misclassified Samples	Recognition Rate (%)
ON	150	68	32	68.5714
OFF	150	93	7	93.3333
Up	150	69	31	69.5238
Down	150	61	39	61.9048
Switch	150	95	5	95.2381
no operation	150	98	2	98.0952

Table 5.12: Results for gestures when no operation class is included

This experiment is done to realize the comparison between nearest neighbor and neural network classifier shows in experiment 5.4.4 after increasing the number of samples to be more than 600, and it shows that the neural network classifier performs better than nearest neighbor classifier. This is expected because the training process of the neural network classifier does tuning to different weights and parameters the mater that leads to better recognition rates for both training and testing data.

5.4 Summary

This chapter presented results that are produced from all experiments done in this project. Experiments include variant parameters, different classifiers and comparison between their results.

The best result that used in this project which obtained in experiment 5.4.4, while the comparison between nearest neighbor classifier and neural network that mentioned in experiment 5.4.5 proves that neural network is better for this project.

Chapter 6

Conclusion

Chapter Contents:

6.1 Results and conclusion

In the hand gesture Based TV project, a neural network was built, trained using a database of pictures and tested in real time using gestures captured through a webcam. The simulation phase the results were very good as shown in chapter five, but there is some difference in real time results, since that the training process depends on captured images using digital camera, while in real time the tested images are captured using a webcam.

The results that we reached satisfy the main requirements we stated, and following are the results that were achieved:

1. Building a system that captures the images of gestures, and upon that controlling the TV.
2. Building the necessary interfacing circuit.
3. Connecting the interfacing circuit with Workstation using the parallel port.
4. Implementing the software that communicates with the hardware.
5. The system achieved good robustness against photometric and geometric changes.

6.2 Challenges

The following are the main challenges that we faced while building the project:

1. The parallel port problem and how to control it through java, we solved it through C++ language.
2. Webcam connection through java, we solve it through java media framework.
3. Scale changing problem, we solved it by taking the ratio of each bin of the histogram to the total summation of all histogram bins.

6.3 Future work

There are some suggestions to improve this system, such as:

1. Dealing with complex environments.
2. Increasing the number of classes that the user can use to control TV.
3. Integrating voice and gesture recognition to control the TV.
4. Building the whole system on a microcontroller.

References

[1] T.Freeman, CraigD. Weissman, *"Television Control by Hand Gestures,"* 24 December 1994, Mitsubishi Electric Research Labs.

[2] Alaa Halawani, *"Recognition of Gestures in Arabic Sign Language using Neuro-fuzzy Systems,"* Master Thesis, Jordan University of Science and Technology, Irbid, Jordan, January 2000.

[3] Vladimir I. Pavlovic, Rajeev Sharma, Thomas S. Huang, "Visual Interpretation of Hand Gestures for Human-Computer Interaction,"reported in Telecommunications Policy Review, July 31 1994.Porter/Novellisurvey.

[4] M. Kass and A. P. Witkin. "Analyzing oriented patterns" .In Proc. Ninth IJCAI, Los Angeles, CA, August 1985.

[5] J. Davis and M. Shah." Gesture recognition". Technical Report CS-TR-93-11, University of Central Florida, Orlando, FL 32816, 1993.

[6] J. M. Rehg and T. Kanade. Digiteyes: "vision-based human hand tracking". Technical Report CMU-CS-93-220, Carnegie Mellon School of Computer Science, Pittsburgh, PA 15213,1993.

[7] J. Segen. Gest: "a learning computer vision system that recognizes gestures". In Machine Learning IV. Morgan Kau_man, 1992. edited by Michalski et. al.

[8] Giachetti, A., Matching Techniques to Compute Image Motion," Image and Vision Computing",18(3), pp. 247–260, 2000.

[9] Bishop, C. M., "Neural Networks for Pattern Recognition", Oxford University Press, Oxford UK, 1995.

[10] Mark S. Nixon and Alberto S. Aguado, "Feature Extraction and Image Processing", 2002.

[11] http://en.wikipedia.org/wiki/Machine_vision

[12] http://en.wikipedia.org/wiki/Adaptive_thresholding

[13] http://www.freepatentsonline.com/EP0771101.html

[14] http://en.wikipedia.org/wiki/Feature_extraction

[15] http://en.wikipedia.org/wiki/C%2B%2B

[16] http://www.kpsec.freeuk.com/components/relay.htm

Appendix A:

Class Description

System_Class

Class Main

java.lang.Object

System_Class.Main

public class **Main**
extends java.lang.Object

Author:

Ahmed Shawki Tamimi

Field Summary

boolean		**Ch Vol Flage**
(package private)	double[]	**Down**
(package private)	Image Analyzer	**image analyzer**
(package private)	JWebCam	**myWebCam**
(package private)	NeuralNetwork	**neural network**
(package private)	double[]	**Off**
(package private)	double[]	**On**
(package private)	PortControl	**P C**
(package private)	double[]	**Result**
(package private)	double[]	**Switch**
(package private)	double[]	**Up**
(package private)	long	**x**

Constructor Summary

Main()

Method Summary

static void	**main**(java.lang.String[] args)
private boolean	**MatIsEqual**(double[] matrix1, double[] matrix2)

Field Detail

Result

double[] **Result**

On

double[] **On**

Off

double[] **Off**

Up

double[] **Up**

Down

double[] **Down**

Switch

double[] **Switch**

image_analyzer

Image_Analyzer **image_analyzer**

neural_network

NeuralNetwork **neural_network**

myWebCam

```
JWebCam myWebCam
```

P_C

```
PortControl P_C
```

---

x

```
long x
```

Ch_Vol_Flage

```
public boolean Ch_Vol_Flage
```

Constructor Detail

Main

```
public Main()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

MatIsEqual

```
private boolean MatIsEqual(double[] matrix1,
                           double[] matrix2)
```

Class MyCaptureDeviceInfo

java.lang.Object

    System_Class.MyCaptureDeviceInfo

class **MyCaptureDeviceInfo**
extends java.lang.Object

Field Summary

CaptureDeviceInfo	**capDevInfo**

Constructor Summary

MyCaptureDeviceInfo(CaptureDeviceInfo devInfo)

Method Summary

java.lang.String	**toString**()

Field Detail

capDevInfo

public CaptureDeviceInfo **capDevInfo**

Constructor Detail

MyCaptureDeviceInfo

public **MyCaptureDeviceInfo**(CaptureDeviceInfo devInfo)

Method Detail

toString

public java.lang.String **toString**()

Overrides:
toString in class java.lang.Object

System_Class

Class MyVideoFormat

java.lang.Object

  System_Class.MyVideoFormat

class **MyVideoFormat**
extends java.lang.Object

Field Summary

VideoFormat	**format**

Constructor Summary

MyVideoFormat(VideoFormat format)

Method Summary

java.lang.String	**toString**()

Field Detail

format

public VideoFormat **format**

Constructor Detail

MyVideoFormat

public **MyVideoFormat**(VideoFormat format)

Method Detail

toString

public java.lang.String **toString**()

Overrides:

toString in class java.lang.Object

System_Class

Class MySnapshot

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        System_Class.MySnapshot
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
class MySnapshot
extends javax.swing.JFrame
implements java.awt.image.ImageObserver
```

Nested Class Summary

Field Summary

(package private) int	**imageHeight**
(package private) int	**imageWidth**
protected java.awt.Image	**photo**

Constructor Summary

MySnapshot()

Method Summary

void	**MySnapshot**(java.awt.Image grabbedFrame, java.awt.Dimension imageSize, int shotCounter)
void	**paint**(java.awt.Graphics g)

Field Detail

photo

```
protected java.awt.Image photo
```

imageHeight

```
int imageHeight
```

imageWidth

```
int imageWidth
```

Constructor Detail

MySnapshot

```
public MySnapshot()
```

Method Detail

MySnapshot

```
public void MySnapshot(java.awt.Image grabbedFrame,
                       java.awt.Dimension imageSize,
                       int shotCounter)
```

paint

```
public void paint(java.awt.Graphics g)
```

Overrides:

`paint` in class `java.awt.Container`

System_Class

Class JWebCam

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Window
                java.awt.Frame
                    javax.swing.JFrame
                        System_Class.JWebCam
```

All Implemented Interfaces:

java.awt.event.ComponentListener, java.awt.event.WindowListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class JWebCam
extends javax.swing.JFrame
implements java.awt.event.WindowListener,
java.awt.event.ComponentListener
```

See Also:

Serialized Form

Nested Class Summary

Field Summary

static javax.swing.JButton	**b**
protected VideoFormat	**currentFormat**
protected FormatControl	**formatControl**
protected static java.awt.Dimension	**imageSize**
protected boolean	**initialised**
static java.awt.image.BufferedImage	**input**

protected MediaLocator	**ml**
protected MyCaptureDeviceInfo[]	**myCaptureDevices**
protected MyVideoFormat[]	**myFormatList**
protected static Player	**player**
protected static int	**shotCounter**
(package private) static MySnapshot	**snapshot**
protected javax.swing.JLabel	**statusBar**
static boolean	**use**
protected Format[]	**videoFormats**
protected java.awt.Component	**visualComponent**
protected javax.swing.JPanel	**visualContainer**
protected CaptureDeviceInfo	**webCamDeviceInfo**

Constructor Summary

JWebCam(java.lang.String frameTitle)

Method Summary

MyCaptureDeviceInfo[]	**autoDetect**()
void	**componentHidden**(java.awt.event.ComponentEvent e)
void	**componentMoved**(java.awt.event.ComponentEvent e)
void	**componentResized**(java.awt.event.ComponentEvent e)
void	**componentShown**(java.awt.event.ComponentEvent e)

static java.awt.image.BufferedImage	**createBufferedImage**(java.awt.Image image)
void	**deviceInfo**()
protected void	**finalize**()
VideoFormat	**getFormat**()
static Buffer	**grabFrameBuffer**()
static java.awt.Image	**grabFrameImage**()
boolean	**initialise**()
boolean	**initialise**(CaptureDeviceInfo _deviceInfo)
void	**playerClose**()
void	**setFormat**(VideoFormat selectedFormat)
protected void	**setUpToolBar**()
void	**windowActivated**(java.awt.event.WindowEvent e)
void	**windowClosed**(java.awt.event.WindowEvent e)
void	**windowClosing**(java.awt.event.WindowEvent e)
void	**windowDeactivated**(java.awt.event.WindowEvent e)
void	**windowDeiconified**(java.awt.event.WindowEvent e)
void	**windowIconified**(java.awt.event.WindowEvent e)
void	**windowOpened**(java.awt.event.WindowEvent e)

Field Detail

shotCounter

```
protected static int shotCounter
```

statusBar

```
protected javax.swing.JLabel statusBar
```

visualContainer

```
protected javax.swing.JPanel visualContainer
```

visualComponent

```
protected java.awt.Component visualComponent
```

player

```
protected static Player player
```

webCamDeviceInfo

```
protected CaptureDeviceInfo webCamDeviceInfo
```

ml

```
protected MediaLocator ml
```

imageSize

```
protected static java.awt.Dimension imageSize
```

formatControl

```
protected FormatControl formatControl
```

currentFormat

```
protected VideoFormat currentFormat
```

videoFormats

```
protected Format[] videoFormats
```

myFormatList

protected MyVideoFormat[] **myFormatList**

myCaptureDevices

protected MyCaptureDeviceInfo[] **myCaptureDevices**

use

public static boolean **use**

initialised

protected boolean **initialised**

snapshot

static MySnapshot **snapshot**

b

public static javax.swing.JButton **b**

input

public static java.awt.image.BufferedImage **input**

Constructor Detail

JWebCam

public **JWebCam**(java.lang.String frameTitle)

Method Detail

initialise

```
public boolean initialise()
                    throws java.lang.Exception
```

Throws:
java.lang.Exception

---

initialise

```
public boolean initialise(CaptureDeviceInfo _deviceInfo)
                    throws java.lang.Exception
```

Throws:
java.lang.Exception

---

setFormat

```
public void setFormat(VideoFormat selectedFormat)
```

---

getFormat

```
public VideoFormat getFormat()
```

---

setUpToolBar

```
protected void setUpToolBar()
```

---

autoDetect

```
public MyCaptureDeviceInfo[] autoDetect()
```

---

deviceInfo

```
public void deviceInfo()
```

---

grabFrameBuffer

```
public static Buffer grabFrameBuffer()
```

---

grabFrameImage

```
public static java.awt.Image grabFrameImage()
```

---

playerClose

```
public void playerClose()
```

---

windowClosing

```
public void windowClosing(java.awt.event.WindowEvent e)
```

Specified by:

windowClosing in interface java.awt.event.WindowListener

---

componentResized

```
public void componentResized(java.awt.event.ComponentEvent e)
```

Specified by:

componentResized in interface java.awt.event.ComponentListener

---

windowActivated

```
public void windowActivated(java.awt.event.WindowEvent e)
```

Specified by:

windowActivated in interface java.awt.event.WindowListener

---

windowClosed

```
public void windowClosed(java.awt.event.WindowEvent e)
```

Specified by:

windowClosed in interface java.awt.event.WindowListener

---

windowDeactivated

```
public void windowDeactivated(java.awt.event.WindowEvent e)
```

Specified by:

windowDeactivated in interface java.awt.event.WindowListener

---

windowDeiconified

```
public void windowDeiconified(java.awt.event.WindowEvent e)
```

Specified by:

windowDeiconified in interface java.awt.event.WindowListener

---

windowIconified

```
public void windowIconified(java.awt.event.WindowEvent e)
```

Specified by:

windowIconified in interface java.awt.event.WindowListener

---

windowOpened

```
public void windowOpened(java.awt.event.WindowEvent e)
```

Specified by:

windowOpened in interface java.awt.event.WindowListener

---

componentHidden

```
public void componentHidden(java.awt.event.ComponentEvent e)
```

Specified by:

componentHidden in interface java.awt.event.ComponentListener

---

componentMoved

```
public void componentMoved(java.awt.event.ComponentEvent e)
```

Specified by:

componentMoved in interface java.awt.event.ComponentListener

---

componentShown

```
public void componentShown(java.awt.event.ComponentEvent e)
```

Specified by:

componentShown in interface java.awt.event.ComponentListener

---

finalize

```
protected void finalize()
                throws java.lang.Throwable
```

Overrides:

finalize in class java.lang.Object

Throws:

java.lang.Throwable

---

createBufferedImage

```
public static java.awt.image.BufferedImage
createBufferedImage(java.awt.Image image)
```

System_Class

Class Image_Analyzer

java.lang.Object

System_Class.Image_Analyzer

public class **Image_Analyzer**
extends java.lang.Object

Author:
Ahmed

Field Summary

private int	**b**
private double[][]	**Degrees Orient**
private double[][]	**Final Orient Matrix0**
private double[][]	**Final Orient Matrix1**
private double[]	**Final Orient Matrix3**
private double[]	**Final Orient Matrix4**
private int	**g**
private double[][]	**Hconv**
private int	**Heigh**
private double[]	**Hostogram Matrix**
private double[][]	**image edge**
private float[]	**kernelMatrix1**
private float[]	**kernelMatrix2**
private double[][]	**mag**
private int	**no of nonzeros**
private double[][]	**Orient Matrix**

private int	**pixel**
private int	**r**
private double[][]	**Result**
private int	**temp**
private double[][]	**temp Result**
private double	**Thresholding**
private double[][]	**Vconv**
private int	**Width**

Constructor Summary

Image Analyzer(java.awt.image.BufferedImage input, int height, int width)

Creates a new instance of Main

Method Summary

private double[][]	**Add Matrix**(double[][] Matrix1, double[][] Matrix2, int Hight, int Width)
private double[][]	**Convolution**(double[][] Matrix, int Hight, int Width, float[] Kernel)
private double[][]	**Get Degree Orient**(double[][] Matrix, int Hight, int Width)
private double[][]	**Get Edges**(double[][] Matrix, int Hight, int Width, double thresholding)
double[]	**Get Histogram**()
private double[]	**Get Nonzeros Elements**(double[][] Matrix, int Hight, int Width, int size)
private double[][]	**Get Orient**(double[][] Matrix1, double[][] Matrix2, int Hight, int Width)
private double[]	**Histogram**(double[] Matrix, int size)

private double[][]	**Magnitude**(double[][] Matrix1, double[][] Matrix2, int Hight, int Width)
private double	**Mean**(double[][] Matrix, int Hight, int Width)
private double[][]	**Mul Matrix Per Element**(double[][] Matrix1, double[][] Matrix2, int Hight, int Width)
private int	**No NonZeros Element**(double[][] Matrix, int Hight, int Width)
double[]	**Normc**(double[] Matrix)
private double[][]	**RGB2GRAY**(java.awt.image.BufferedImage input, int Hight, int Width)
private double[]	**Sub Matrix By One**(double[] Matrix, int size)
private double	**Sub Normalize**(double[] Matrix)
private double[][]	**Transpose**(double[][] Matrix, int Hight, int Width)

Field Detail

Heigh

private int **Heigh**

Width

private int **Width**

---

pixel

private int **pixel**

---

r

private int **r**

---

g

```
private int g
```

---

b

```
private int b
```

---

temp

```
private int temp
```

---

no_of_nonzeros

```
private int no_of_nonzeros
```

Hconv

```
private double[][] Hconv
```

Vconv

```
private double[][] Vconv
```

---

mag

```
private double[][] mag
```

---

image_edge

```
private double[][] image_edge
```

Orient_Matrix

```
private double[][] Orient_Matrix
```

Degrees_Orient

```
private double[][] Degrees_Orient
```

Final_Orient_Matrix0

```
private double[][] Final_Orient_Matrix0
```

Final_Orient_Matrix1

```
private double[][] Final_Orient_Matrix1
```

Result

```
private double[][] Result
```

---

temp_Result

```
private double[][] temp_Result
```

Thresholding

```
private double Thresholding
```

Final_Orient_Matrix3

```
private double[] Final_Orient_Matrix3
```

Final_Orient_Matrix4

```
private double[] Final_Orient_Matrix4
```

Hostogram_Matrix

```
private double[] Hostogram_Matrix
```

---

kernelMatrix1

```
private float[] kernelMatrix1
```

---

kernelMatrix2

```
private float[] kernelMatrix2
```

Constructor Detail

Image_Analyzer

```
public Image_Analyzer(java.awt.image.BufferedImage input,
                      int height,
                      int width)
```

Creates a new instance of Main

Method Detail

RGB2GRAY

```
private double[][] RGB2GRAY(java.awt.image.BufferedImage input,
                            int Hight,
                            int Width)
```

Convolution

```
private double[][] Convolution(double[][] Matrix,
                               int Hight,
                               int Width,
                               float[] Kernel)
```

Magnitude

```
private double[][] Magnitude(double[][] Matrix1,
                             double[][] Matrix2,
                             int Hight,
                             int Width)
```

Mean

```
private double Mean(double[][] Matrix,
                   int Hight,
                   int Width)
```

Get_Edges

```
private double[][] Get_Edges(double[][] Matrix,
                             int Hight,
```

```
                                int Width,
                                double thresholding)
```

Get_Orient

```
private double[][] Get_Orient(double[][] Matrix1,
                              double[][] Matrix2,
                              int Hight,
                              int Width)
```

Get_Degree_Orient

```
private double[][] Get_Degree_Orient(double[][] Matrix,
                                     int Hight,
                                     int Width)
```

Add_Matrix

```
private double[][] Add_Matrix(double[][] Matrix1,
                              double[][] Matrix2,
                              int Hight,
                              int Width)
```

Mul_Matrix_Per_Element

```
private double[][] Mul_Matrix_Per_Element(double[][] Matrix1,
                                          double[][] Matrix2,
                                          int Hight,
                                          int Width)
```

Transpose

```
private double[][] Transpose(double[][] Matrix,
                             int Hight,
                             int Width)
```

No_NonZeros_Element

```
private int No_NonZeros_Element(double[][] Matrix,
                                int Hight,
                                int Width)
```

Get_Nonzeros_Elements

```
private double[] Get_Nonzeros_Elements(double[][] Matrix,
                                       int Hight,
                                       int Width,
```

int size)

Sub_Matrix_By_One

```
private double[] Sub_Matrix_By_One(double[] Matrix,
                                   int size)
```

Histogram

```
private double[] Histogram(double[] Matrix,
                           int size)
```

Get_Histogram

```
public double[] Get_Histogram()
```

Normc

```
public double[] Normc(double[] Matrix)
```

Sub_Normalize

```
private double Sub_Normalize(double[] Matrix)
```

System_Class

Class NeuralNetwork

java.lang.Object

System_Class.NeuralNetwork

public class **NeuralNetwork**
extends java.lang.Object

Field Summary

private static double[]	**Hidden b**
private static double[][]	**Hidden Output w**
private static double[]	**Hidden Temp Result**
private static double[]	**Input Data**
private static double[][]	**Input Hidden w**
private static int	**No Hidden**
private static int	**No Input**
private static int	**No Output**
private static double[]	**Output b**
private static double[]	**Output Result**

Constructor Summary

NeuralNetwork(int No_Input, int No_Hidden, int No_Output)

Method Summary

private static void	**forward pass**()
private static int	**getNo Hidden**()
private static int	**getNo Input**()

private static int	**getNo_Output**()
double[]	**getOutput_Result**()
private static void	**initialize**()
private static void	**LoadBais**()
private static void	**LoadWaights**()
double[]	**recognize**(double[] data)
void	**setInput_Data**(double[] input_Data)
private void	**setNo_Hidden**(int no_Hidden)
private void	**setNo_Input**(int no_Input)
private void	**setNo_Output**(int no_Output)
private static void	**setOutput_Result**(double[] output_Result)
private static double	**Threshold**(double sum)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

No_Input

private static int **No_Input**

No_Hidden

private static int **No_Hidden**

No_Output

private static int **No_Output**

Input_Hidden_w

```
private static double[][] Input_Hidden_w
```

Hidden_Output_w

```
private static double[][] Hidden_Output_w
```

Input_Data

```
private static double[] Input_Data
```

Hidden_Temp_Result

```
private static double[] Hidden_Temp_Result
```

Output_Result

```
private static double[] Output_Result
```

Hidden_b

```
private static double[] Hidden_b
```

Output_b

```
private static double[] Output_b
```

Constructor Detail

NeuralNetwork

```
public NeuralNetwork(int No_Input,
                     int No_Hidden,
                     int No_Output)
```

Method Detail

initialize

```
private static void initialize()
```

LoadWaights

```
private static void LoadWaights()
```

LoadBais

```
private static void LoadBais()
```

recognize

```
public double[] recognize(double[] data)
```

forward_pass

```
private static void forward_pass()
```

Threshold

```
private static double Threshold(double sum)
```

getNo_Hidden

```
private static int getNo_Hidden()
```

setNo_Hidden

```
private void setNo_Hidden(int no_Hidden)
```

getNo_Input

```
private static int getNo_Input()
```

setNo_Input

```
private void setNo_Input(int no_Input)
```

getNo_Output

```
Private static int getNo_Output()
```

setNo_Output

```
private void setNo_Output(int no_Output)
```

getOutput_Result

```
public double[] getOutput_Result()
```

setInput_Data

```
public void setInput_Data(double[] input_Data)
```

setOutput_Result

```
private static void setOutput_Result(double[] output_Result)
```

System_Class

Class PortControl

java.lang.Object

    System_Class.PortControl

public class **PortControl**
extends java.lang.Object

Constructor Summary

PortControl()

Method Summary

void	**UsePort**(int Signal)

Constructor Detail

PortControl

public **PortControl**()

Method Detail

UsePort

public void **UsePort**(int Signal)
            throws java.lang.Exception

Throws:
java.lang.Exception

Appendix B:

Over all System experiments

target error	1.00E-05	1.00E-05	1.00E-05
epochs	3000	3000	3000
layer #	3	3	3
# neurons	36-72-5	36-72-5	36-72-5
func.	logsig / purelin	logsig / purelin	satlin / purelin
traning method	traingd	traingdm	traingdm
using normc	no	no	no
traning samples #	30	30	30
testing samples #	10	10	10
using scale sol.	y	y	y

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**	***traning samples Test***	***test sample Test***
On	90	86.6	86.6667	86.6667	*100*	*93.3333*
Off	90	10	80	30	*83.3333*	*10*
up	86.6	60	80	80	*76.6667*	*80*
down	70	10	90	10	*90*	*40*
switch	90	100	90	70	*96.6667*	*100*
average	85.32	53.32	85.33334	55.33334	*89.33334*	*64.66666*

target error	1.00E-05	1.00E-05	1.00E-05
epochs	3000	30000	5000
layer #	3	3	3
# neurons	36-72-5	36-72-5	36-72-5
func.	satlin / purelin	satlin / purelin	satlin / purelin
traning method	traingd	traingdm	traingdm
using normc	no	no	yes
traning samples #	30	30	30
testing samples #	10	10	10
using scale sol.	y	y	yes

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**	**traning samples Test**	***test sample Test***
On	90	93.33	96.6667	86.6667	*100*	*93.3333*
Off	90	20	100	50	*100*	*70*
up	90	10	93.3333	90	*96.66*	*90*
down	76.66	80	86.6667	50	*90*	*60*
switch	100	60	100	100	*100*	*80*
average	89.332	52.666	95.33334	75.33334	*97.332*	*78.66666*

target error	1.00E-05	1.00E-05	1.00E-05
epochs	30000	3000	3000
layer #	3	3	3
# neurons	36-72-5	36-72-5	36-72-5
func.	satlin / purelin	satlin / purelin	satlin / purelin
traning method	traingdm	traingdm	traingdm
using normc	yes	yes	no
traning samples #	30	30	30
testing samples #	10	10	10
using scale sol.	yes	no	no

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**	**traning samples Test**	***test sample Test***
On	100	93.3333	96.6667	93.3333	*0*	*33.33*
Off	100	40	93.333	30	*0*	*0*
up	100	90	96.6667	90	*0*	*0*
down	100	40	86.6667	50	*0*	*0*
switch	100	100	100	100	*86.66*	*40*
average	100	72.66666	94.66662	72.66666	*17.332*	*14.666*

target error	1.00E-05	1.00E-05	1.00E-05
epochs	3000	3000	800
layer #	3	3	3
# neurons	36-100-5	36-50-5	36-72-5
func.	satlin / purelin	satlin / purelin	satlin / purelin
traning method	traingdm	traingdm	trainscg
using normc	y	y	y
traning samples #	30	30	30
testing samples #	10	10	10
using scale sol.	y	y	y

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**	**traning samples Test**	***test sample Test***
On	100	86.6	100	86.6	*100*	*86.6667*
Off	100	40	96.6667	60	*100*	*80*
up	100	80	96.6667	90	*100*	*90*
down	93.333	20	93.333	50	*100*	*80*
switch	100	90	100	90	*100*	*100*
average	98.6666	63.32	97.33328	75.32	100	*87.33334*

target error	1.00E-05	1.00E-05	1.00E-05
epochs	1000	3000	3000
layer #	3	3	3
# neurons	36-72-5	36-72-5	36-72-5
func.	satlin / purelin	satlin / purelin	satlin / purelin
traning method	traingd	traingda	trainrp
using normc	y	y	y
traning samples #	30	30	30
testing samples #	10	10	10
using scale sol.	y	y	y

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**	***traning samples Test***	***test sample Test***
On	100	86.6667	100	93.6667	*100*	*86.6667*
Off	100	50	100	50	*100*	*50*
up	100	90	100	90	*100*	*100*
down	96.6667	50	96.6667	50	*100*	*10*
switch	100	90	100	90	*100*	*100*
average	99.33334	73.33334	99.33334	74.73334	100	69.33334

target error	1.00E-05	1.00E-05
epochs	3000	3000
layer #	3	3
# neurons	36-72-5	36-72-5
func.	satlin / purelin	satlin / purelin
traning method	traincgf	traincgp
using normc	y	y
traning samples #	30	30
testing samples #	10	10
using scale sol.	y	y

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**
On	100	86.6667	100	93.3333
Off	100	50	100	30
up	100	80	100	90
down	100	60	100	70
switch	100	100	100	100
average	100	75.33334	100	76.66666

target error	1.00E-05	1.00E-05	1.00E-05
epochs	800	800	800
layer #	3	3	3
# neurons	36-72-5	36-72-6	36-72-5
func.	satlin / purelin	satlin / purelin	logsog / purelin
traning method	trainscg	trainscg	trainscg
using normc	y	y	y
traning samples #	70	70	70
testing samples #	35	35	35
thresholding	2.5	2.5	2.5
using scale sol.	y	y	y

	traning samples Test	**test sample Test**	**traning samples Test**	**test sample Test**	**traning samples Test**	***test sample Test***
On	98.5714	100	100	100	*98.5714*	*100*
Off	100	97.1429	100	100	*97.1429*	*94.2857*
up	98.5714	97.1429	100	94.2857	*97.1429*	*100*
down	100	88.5714	100	77.1429	*100*	*85.7143*
switch	100	82.8571	100	91.4286	*97.1429*	*85.7143*
average	99.42856	93.14286	100	92.57144	*98.00002*	*93.14286*

target error	1.00E-05
epochs	800
layer #	3
# neurons	36-72-5
func.	logsog / logsog
traning method	trainscg
using normc	y
traning samples #	70
testing samples #	35
thresholding	4.5
using scale sol.	y

	traning samples Test	**test sample Test**
On	100	97.14
Off	98.57	88.57
up	100	100
down	98.57	94.28
switch	98.57	91
average	99.142	94.198

target error	1.00E-05	1.00E-05	1.00E-05
epochs	800	800	800
layer #	3	3	3
# neurons	36-72-5	36-72-5	36-72-5
func.	satlin / satlin	satlin / satlin	satlin / satlin
traning method	trainscg	trainscg	trainscg
using normc	y	y	y
traning samples #	70	70	70
testing samples #	35	35	35
thrssholding	2.5	2.5	2.5
using scale sol.	y	y	y
output range	1 --> -1	1 --> 0	1 --> 0

On	100	85.7143	100	97.1429	*100*	*97.1429*
Off	97.1429	91.4286	98.58	88.5714	*97.1429*	*88.5714*
up	98.58	82.8571	98.58	94.2857	*98.58*	*91.4286*
down	97.1429	80	100	88.5714	*100*	*85.7143*
switch	98.58	60	98.58	77.1429	*98.58*	*80*
no op	100	100	100	100	*100*	*100*
average	98.5743	83.33333333	99.29	90.95238333	*99.05048333*	*90.4762*

target error	1.00E-05
epochs	800
layer #	3
# neurons	36-72-5
func.	satlin / satlin
traning method	trainscg
using normc	y
traning samples #	70
testing samples #	35
thrssholding	4.5
using scale sol.	y
output range	1 --> 0

On	100	100
Off	98.57	88.57
up	95.57	94.28
down	98.57	91.42
switch	98.57	88.57
no op	97.14	100
average	98.07	93.80666667