Fayoum University

Faculty of Computers and Information

Computer Science Department

Graduation project Documentation

Virtual Mouse Controller

Abd-Elrahman Ahmed Abd-Elalaim

Ahmed Hammad Ahmed

Ahmed Hamouda Ramadan

Supervisor: Dr. Mohammed Hassan

May 2015

ACKNOWLEDGMENT

We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We would like to express our gratitude to our advisor, Dr. Mohamed Hassan, for his support, patience, and encouragement throughout my graduate studies. He always finds the time for listening to the little problems and roadblocks that unavoidably. His technical and editorial advice was essential to the completion of this project,

Our greatly appreciate to the contribution of our faculty, Faculty of Computer and Information (FCI).

Especially Prof. Nabila Hassan, listen to our problems and did her best to satisfy all groups.

Our grateful thanks also go to both TA.Hussein shehata, TA.Rasha El-Badry a big contribution and hard worked from both of you during the last ten months. The project would be nothing without the enthusiasm, helping and support from both of you. Besides, this project makes us realized the value of working together as a team and as a new experience in working environment, which challenges us every minute.

We don’t forget help from Dr.James Gips we contact with him via Gmail and give us advises to accomplish this project

Table Of CONTENTS

Contents:

ABSTRACT

Virtual Mouse Controller (VMC) is a software application that help arm Disabilities (such as Parkinson's disease, Arm amputations or they born With Congenital disorder in their arms) to control mouse operation using face not physical mouse in suitable manner.

VMC is alternative source for physical mouse (traditional mouse) user can make click by one of three option

1) by speech recognition (user say left for left click and right for right click)
2) by time limit (stay in same position for one second means left click and for three seconds means double click)
3) by eye blink (user blink left eye means left click and blink right eye means right click)

VMC use only webcam of laptop or any USB webcam (this to reduce cost as possible as we can)

Objectives of our Project VMC:

1) help disabled people to control computer by themselves
2) spread virtual mouse technology
3) reduce cost (almost no additional cost)

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
ASR	automatic speech recognition
BP	back probagetion
CDF	cumulative distributive function
CV	computer vision
DFD	data flow diagram
DIP	digital image processing
GIC	Gamma Intensity Correction
GUI	graphical user interface
HD	High-definition
HE	Histogram Equalization
JSGF	JSpeech Grammar Format
RGB	Red, Green and Blue
RNN	Recurrent neural network
SR	speech recognition
SSD	System Sequence Diagram
STT	speech to text

Chapter 1

Introduction

1.1 Overview

1.2 Problem Definition

1.3 Project Objectives / goals

1.4 Documentation Structure

1.1 Overview

Nowadays Computers have become more popular and important to our society and there are millions of users for it. Then I can said, how many people have arm disabilities hence they can't use computer in suitable manner because they can't use physical mouse. So, we have to find solution for this

The Virtual Mouse Controller (VMC), is represented as an intelligent assessment system for disabled people.

It has different characteristics which are the following:

1. Accessibility: the system should be easy to access.

The application will be desktop-based and can be accessed from any device which the system will be installed on it.

The Application is developed by java programming language so it work in any platform (such as windows, mac, Linux …)

2. Availability: the system should be in a specified operable state at the start of the mission as NN Files replicated.

3. Accuracy: The system should provide accurate e results

4. Usability: The system should be easy to use. (As GUI need limit interaction from user as he have arm disabilities)

And it has different benefits such as:

1. Help arm disabilities with needing limit interaction from them (

Almost no interaction)

2. No additional H/W required (require only laptop webcam or any USB camera)

3. Reduce Cost

4. No need for traditional mouse

1.2 Problem Definition

Some people have diseases such as Parkinson's disease, Arm amputations or they born with congenital disorder in their arms .so, computer control will be a difficult task for those people

Also, there is no intelligent system that help the user to overcome these problems in suitable manner.

But there is hardware version that this is innovation and used to help user to overcome these problems but this hardware is most expensive and has industrial faults.

So we developed this system that overcome these problems caused by the hardware version of the system (such as can't make right click ,must open his eye as large as he can and this exhausts user and device have rays that have bad effect on user)

1.3 Project Objectives / goals

Our goal, developing intelligent software system that is less expensive (almost no cost) help disabled people.

Then we can list some of the system benefits:

1) Help arm disabilities with needing limit interaction from them (Almost no interaction)
2) No additional H/W required (require only laptop webcam or any USB camera)
3) Reduce Cost
4) No need for traditional mouse

1.2 Documentation Structure

Chapter one Introduction: Contains of an Overview, Problem Definition, Objectives and Documentation Structure.

Chapter two introduce the project Technologies and the related systems which are used before in this field, and description of the programming language which we use in the project ' implementation

Chapter three contains a background of the analysis and design requirements .Then a brief literature review about analysis use case diagram, and the system sequence diagram of the project, the Data Flow Diagram (DFD) of the system and flow chart diagram of the system

Chapter four_contains more details about the implementation of the project and the algorithms which are used to develop the system.

Chapter five_display snap shots of the system in the test level of the life cycle of the system.

Chapter six_contains conclusion about the documentation that represent what will we use in this intelligent system and the feature of the system then refer to the future work of the system.

Chapter 2

Background

2.1 Mouse overview

2.2 Image processing overview

2.3 Artificial Neural Network

2.4 Computer vision

2.5 Speech Recognition

2.6 Related work

2.7 Virtual mouse control (VMC)

2.1 Mouse overview

A computer mouse with the most common standard features: two buttons and a scroll wheel, which can also act as a third button.

Figure 2.1: A computer mouse

In computing, a mouse is a pointing device that detects two-dimensional motion relative to a surface. This motion is typically translated into the motion of a pointer on a display, which allows for fine control of a graphical user interface.

Physically, a mouse consists of an object held in one's hand, with one or more buttons. Mice often also feature other elements, such as touch surfaces and "wheels", which enable additional control and dimensional input.

2.1.1 Naming

The earliest known publication of the term mouse as a computer pointing device is in Bill English's 1965 publication "Computer-Aided Display Control".

The online Oxford Dictionaries entry for mouse states the plural for the small rodent is mice, while the plural for the small computer connected device is either mice or mouse. However, in the use section of the entry it states that the more common plural is mice, and that the first recorded use of the term

in the plural is mice as well (though it cites a 1984 use of mice when there were actually several earlier ones, such as J. C. R. Licklider's "The Computer as a Communication Device" of 1968). According to the fifth edition of The American Heritage Dictionary of the English Language the plural can be either "mice" or "mouse".

2.1.2 Operation

A mouse typically controls the motion of a pointer in two dimensions in a graphical user interface (GUI). The mouse turns movements of the hand backward and forward, left and right into equivalent electronic signals that in turn are used to move the pointer.

The relative movements of the mouse on the surface are applied to the position of the pointer on the screen, which signals the point where actions of the user take place, so that the hand movements are replicated by the pointer. Clicking or hovering (stopping movement while the cursor is within the bounds of an area) can select files, programs or actions from a list of names, or (in graphical interfaces) through small images called "icons" and other elements. For example, a text file might be represented by a picture of a paper notebook, and clicking while the cursor hovers this icon might cause a text editing program to open the file in a window.

Different ways of operating the mouse cause specific things to happen in the GUI:

- Click: pressing and releasing a button.
    - (left) Single-click: clicking the main button.
    - (left) Double-click: clicking the button two times in quick succession counts as a different gesture than two separate single clicks.

  - (left) Triple-click: clicking the button three times in quick succession.
  - Right-click: clicking the secondary button.
  - Middle-click: clicking the tertiary button.
- Drag: pressing and holding a button, then moving the mouse without releasing. (Using the command "drag with the right mouse button" instead of just "drag" when one instructs a user to drag an object while holding the right mouse button down instead of the more commonly used left mouse button.)
- Button chording (a.k.a. Rocker navigation).
  - Combination of right-click then left-click.
  - Combination of left-click then right-click or keyboard letter.
  - Combination of left or right-click and the mouse wheel.
- Clicking while holding down a modifier key.
- Moving the pointer a long distance: When a practical limit of mouse movement is reached, one lifts up the mouse, brings it to the opposite edge of the working area while it is held above the surface, and then replaces it down onto the working surface. This is often not necessary, because acceleration software detects fast movement, and moves the pointer significantly faster in proportion than for slow mouse motion.
- Multi-touch: this method is similar to a multi-touch trackpad on a laptop with support for tap input for multiple fingers, the most famous example being the Apple Magic Mouse.

2.1.2.1 Mouse gestures

Users can also employ mice gesturally; meaning that a stylized motion of the mouse cursor itself, called a "gesture", can issue a command or map to

a specific action. For example, in a drawing program, moving the mouse in a rapid "x" motion over a shape might delete the shape.

Gestural interfaces occur more rarely than plain pointing-and-clicking; and people often find them more difficult to use, because they require finer motor-control from the user. However, a few gestural conventions have become widespread, including the drag and drop gesture, in which:

1. The user presses the mouse button while the mouse cursor hovers over an interface object
2. The user moves the cursor to a different location while holding the button down
3. The user releases the mouse button

For example, a user might drag-and-drop a picture representing a file onto a picture of a trash can, thus instructing the system to delete the file.

Standard semantic gestures include:

- Crossing-based goal
- Drag and drop
- Menu traversal
- Pointing
- Rollover (Mouse-over)
- Selection

2.1.2.2 Specific uses

Other uses of the mouse's input occur commonly in special application-domains. In interactive three-dimensional graphics, the mouse's motion often translates directly into changes in the virtual camera's orientation. For example, in the first-person shooter genre of games (see below), players

usually employ the mouse to control the direction in which the virtual player's "head" faces: moving the mouse up will cause the player to look up, revealing the view above the player's head. A related function makes an image of an object rotate, so that all sides can be examined.

When mice have more than one button, software may assign different functions to each button. Often, the primary (leftmost in a right-handed configuration) button on the mouse will select items, and the secondary (rightmost in a right-handed) button will bring up a menu of alternative actions applicable to that item. For example, on platforms with more than one button, the Mozilla web browser will follow a link in response to a primary button click, will bring up a contextual menu of alternative actions for that link in response to a secondary-button click, and will often open the link in a new tab or window in response to a click with the tertiary (middle) mouse button.

2.1.3 Buttons

Mouse buttons are micro-switches which can be pressed to select or interact with an element of a graphical user interface, producing a distinctive clicking sound.

The three-button scroll-
mouse has become the most commonly available design. As of 2007 (and roughly since the late 1990s), users most commonly employ the second button to invoke a contextual menu in the computer's software user interface, which contains options specifically tailored to the interface element over which the mouse cursor currently sits. By default, the primary mouse button sits located on the left-hand side of the mouse, for the benefit of right-handed users; left-handed users can usually reverse this configuration via software.

2.1.4 Mouse speed

Mickeys per second is a unit of measurement for the speed and movement direction of a computer mouse. But speed can also refer to the ratio between how many pixels the cursor moves on the screen and how far the mouse moves on the mouse pad, which may be expressed as pixels per Mickey, or pixels per inch, or pixels per cm. The directional movement is called the horizontal mickey count and the vertical mickey count.

The computer industry often measures mouse sensitivity in terms of counts per inch (CPI), commonly expressed as dots per inch (DPI) – the number of steps the mouse will report when it moves one inch. In early mice, this specification was called pulses per inch (ppi). The Mickey originally referred to one of these counts, or one resolvable step of motion. If the default mouse-tracking condition involves moving the cursor by one screen-pixel or dot on-screen per reported step, then the CPI does equate to DPI: dots of cursor motion per inch of mouse motion. The CPI or DPI as reported by manufacturers depends on how they make the mouse; the higher the CPI, the faster the cursor moves with mouse movement. However, software can adjust the mouse sensitivity, making the cursor move faster or slower than its CPI. Current software can change the speed of the cursor dynamically, taking into account the mouse's absolute speed and the movement from the last stop-point. In most software, an example being the Windows platforms, this setting is named "speed" referring to "cursor precision". However, some operating systems name this setting "acceleration", the typical Apple OS designation. This term is in fact incorrect. The mouse acceleration, in the majority of mouse software, refers to the setting allowing the user to modify the cursor acceleration: the change in speed of the cursor over time while the mouse movement is constant.

For simple software, when the mouse starts to move, the software will count the number of "counts" or "mickeys" received from the mouse and will move the cursor across the screen by that number of pixels (or multiplied by a rate factor, typically less than 1). The cursor will move slowly on the screen, having a good precision. When the movement of the mouse passes the value set for "threshold", the software will start to move the cursor more quickly, with a greater rate factor. Usually, the user can set the value of the second rate factor by changing the "acceleration" setting.

Operating systems sometimes apply acceleration, referred to as "ballistics", to the motion reported by the mouse. For example, versions of Windows prior to Windows XP doubled reported values above a configurable threshold, and then optionally doubled them again above a second configurable threshold. These doublings applied separately in the X and Y directions, resulting in very nonlinear response.

2.1.5 Mousepads

Engelbart's original mouse did not require a mouse-pad; the mouse had two large wheels which could roll on virtually any surface. However, most subsequent mechanical mice starting with the steel roller ball mouse have required a mouse-pad for optimal performance.

The mouse-pad, the most common mouse accessory, appears most commonly in conjunction with mechanical mice, because to roll smoothly the ball requires more friction than common desk surfaces usually provide. So-called "hard mouse-pads" for gamers or optical/laser mice also exist.

Most optical and laser mice do not require a pad. Whether to use a hard or soft mouse-pad with an optical mouse is largely a matter of personal preference. One exception occurs when the desk surface creates problems

for the optical or laser tracking, for example, a transparent or reflective surface.

2.1.6 Use in games

Figure 2.2: Logitech G5 laser mouse designed for gaming

The Mac OS Desk Accessory Puzzle in 1984 was the first game designed specifically for a mouse. The device often functions as an interface for PC-based computer games and sometimes for video game consoles.[1]

2.2 Image Processing overview

Digital image processing deals with manipulation of digital images through a digital computer. It is a subfield of signals and systems but focus particularly on images. DIP focuses on developing a computer system that is able to perform processing on an image. The input of that system is a digital image and the system process that image using efficient algorithms, and gives an image as an output. The most common example is Adobe Photoshop. It is one of the widely used application for processing digital images.

Digital image is a representation of a two-dimensional image as a finite set of digital values, called picture elements or pixels in

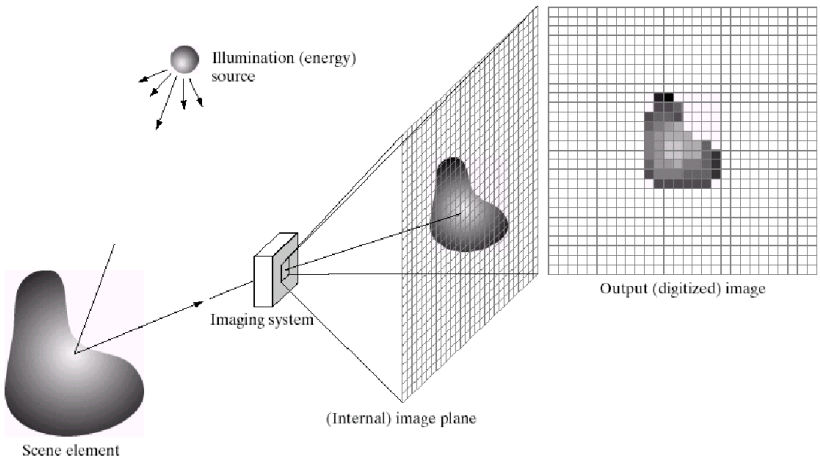
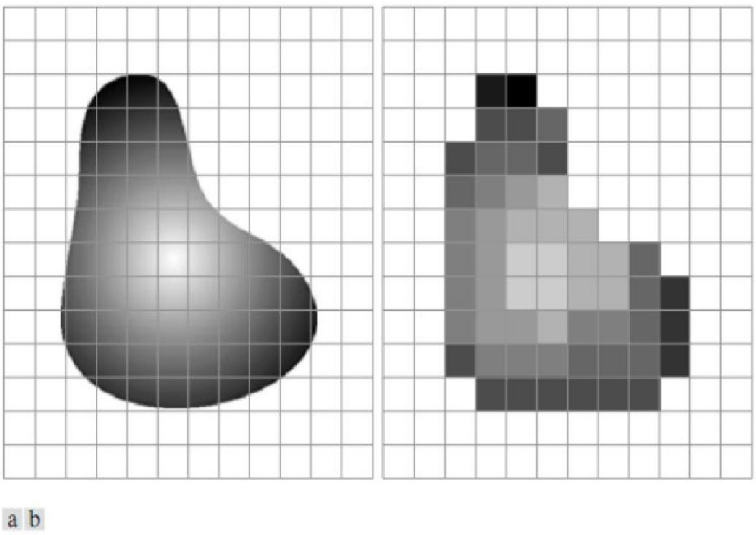


Figure 2.3 Digital image processing

Remember DIP representation



a b

Figure 2.4 DIP representation

2.2.1 How DIP works

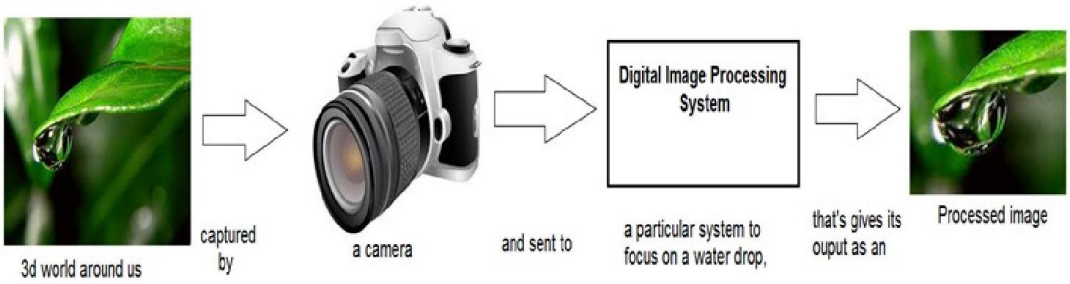


Figure 2.5 how DIP work

In the above figure, an image has been captured by a camera and has been sent to a digital system to remove all the other details, and just focus on the water drop by zooming it in such a way that the quality of the image remains the same.

2.2.2 Concept of Pixel

2..2.2.1 Pixel

Pixel is the smallest element of an image. Each pixel correspond to any one value. In an 8-bit gray scale image, the value of the pixel between 0 and 255. The value of a pixel at any point correspond to the intensity of the light photons striking at that point. Each pixel store a value proportional to the light intensity at that particular location.

2.2.2.2 Gray level

The value of the pixel at any point denotes the intensity of image at that location, and that is also known as gray level.

We will see in more detail about the value of the pixels in the image storage and bits per pixel tutorial, but for now we will just look at the concept of only one pixel value.

Pixel value.(0)

As it has already been define in the beginning of this tutorial that each pixel can have only one value and each value denotes the intensity of light at that point of the image.

We will now look at a very unique value 0. The value 0 means absence of light. It means that 0 denotes dark, and it further means that whenever a pixel has a value of 0, it means at that point, black color would be formed.

Have a look at this image matrix

0	0	0
0	0	0
0	0	0

Figure 2.6 image matrix

Now this image matrix has all filled up with 0. All the pixels have a value of 0. If we were to calculate the total number of pixels form this matrix , this is how we are going to do it.

Total no of pixels = total no. of rows X total no. of columns

= 3 X 3

= 9.

It means that an image would be formed with 9 pixels, and that image would have a dimension of 3 rows and 3 column and most importantly that image would be black.

The resulting image that would be made would be something like this

Figure 2.7 resulting image from image matrix

Now why is this image all black? Because all the pixels in the image had a value of 0.

2.2.3 Perspective Transformation

When human eyes see near things they look bigger as compare to those who are far away. This is called perspective in a general way. Whereas transformation is the transfer of an object etc. from one state to another.

So overall, the perspective transformation deals with the conversion of 3d world into 2d image. The same principle on which human vision works and the same principle on which the camera works.

We will see in detail about why this happens, that those objects which are near to you look bigger, while those who are far away, look smaller even though they look bigger when you reach them.

We will start this discussion by the concept of frame of reference:

2.2.3.1 Frame of reference:

Frame of reference is basically a set of values in relation to which we measure something.

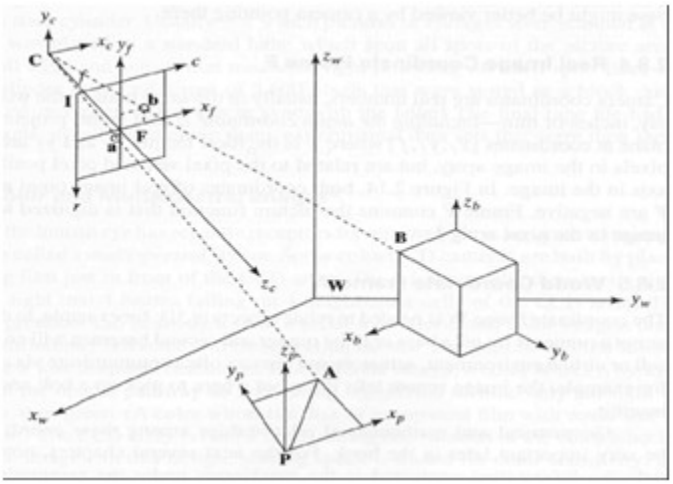


Figure 2.8 image transformation

5 frames of reference

In order to analyze a 3d world/image/scene, 5 different frame of references are required.

- Object
- World
- Camera
- Image
- Pixel

Object coordinate frame

Object coordinate frame is used for modeling objects. For example , checking if a particular object is in a proper place with respect to the other object. It is a 3d coordinate system.

World coordinate frame

World coordinate frame is used for co-relating objects in a 3 dimensional world. It is a 3d coordinate system.

Camera coordinate frame

Camera co-ordinate frame is used to relate objects with respect of the camera. It is a 3d coordinate system.

Image coordinate frame

It is not a 3d coordinate system, rather it is a 2d system. It is used to describe how 3d points are mapped in a 2d image plane.

Pixel coordinate frame

It is also a 2d coordinate system. Each pixel has a value of pixel coordinates.

2.2.3.2 Transformation between these 5 frames

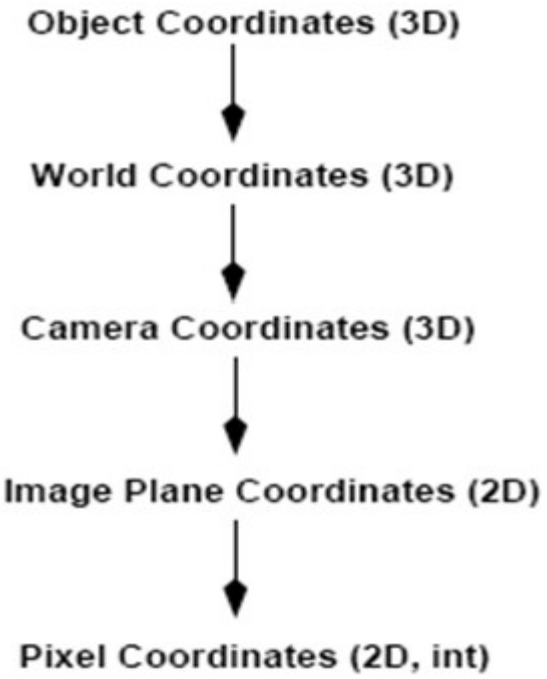


Figure 2.9 Transformation between the 5 frames

That is how a 3d scene is transformed into 2d, with image of pixels.

Now we will explain this concept mathematically.

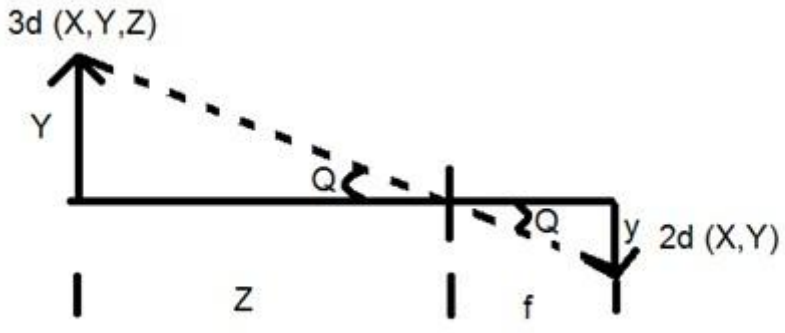


Figure 2.10 image inversion

Where

Y = 3d object

y = 2d Image

f = focal length of the camera

Z = distance between image and the camera

Now there are two different angles formed in this transform which are represented by Q.

The first angle is

$$\tan\theta = -\frac{y}{f}$$

Where minus denotes that image is inverted. The second angle that is formed is:

$$\tan\theta = \frac{Y}{Z}$$

Comparing these two equations we get

$$Y = -f\frac{Y}{Z}$$

From this equation, we can see that when the rays of light reflect back after striking from the object, passed from the camera, an invert image is formed.

We can better understand this, with this example.

2.2.3.2 .1Calculating the size of image formed

Suppose an image has been taken of a person 5m tall, and standing at a distance of 50m from the camera, and we have to tell that what is the size of the image of the person, with a camera of focal length is 50mm.

Solution:

Since the focal length is in millimeter, so we have to convert everything in millimeter in order to calculate it.

So,

Y = 5000 mm.

f = 50 mm.

Z = 50000 mm.

Putting the values in the formula, we get

$$Y = -f\frac{Y}{Z} = -50 \times 5000 / 50000$$

= -5 mm.

Again, the minus sign indicates that the image is inverted.

2.2.4Types of Images

There are many type of images, and we will look in detail about different types of images, and the color distribution in them.

2.2.4.1 The binary image

The binary image as it name states, contain only two pixel values.

0 and 1.

In our previous tutorial of bits per pixel, we have explained this in detail about the representation of pixel values to their respective colors.

Here 0 refers to black color and 1 refers to white color. It is also known as Monochrome.

2 .2.4.1.1 Black and white image

The resulting image that is formed hence consist of only black and white color and thus can also be called as Black and White image.

Figure 2.11 binary image

.

2.2.4.2 Format

Binary images have a format of PBM (Portable bit map)

2.2.4.2.1 2 , 3 , 4 ,5 ,6 bit color format

The images with a color format of 2, 3, 4, 5 and 6 bit are not widely used today. They were used in old times for old TV displays, or monitor displays.

But each of these colors have more than two gray levels, and hence has gray color unlike the binary image.

In a 2 bit 4, in a 3 bit 8, in a 4 bit 16, in a 5 bit 32, in a 6 bit 64 different colors are present.

2.2.4.2.2 8 bit color format

8 bit color format is one of the most famous image format. It has 256 different shades of colors in it. It is commonly known as Grayscale image.

The range of the colors in 8 bit vary from 0-255. Where 0 stands for black, and 255 stands for white, and 127 stands for gray color.

A grayscale image of Einstein is shown below:

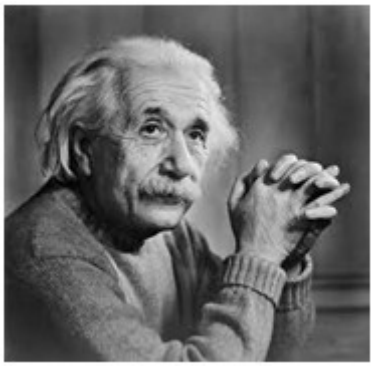


Figure 2.12 gray image

.

2.2.4.2.3 16 bit color format

It is a color image format. It has 65,536 different colors in it. It is also known as High color format.

It has been used by Microsoft in their systems that support more than 8 bit color format. Now in this 16 bit format and the next format we are going to discuss which is a 24 bit format are both color format.

The distribution of color in a color image is not as simple as it was in grayscale image.

A 16 bit format is actually divided into three further formats which are Red, Green and Blue. The famous (RGB) format.

It is pictorially represented in the image below.

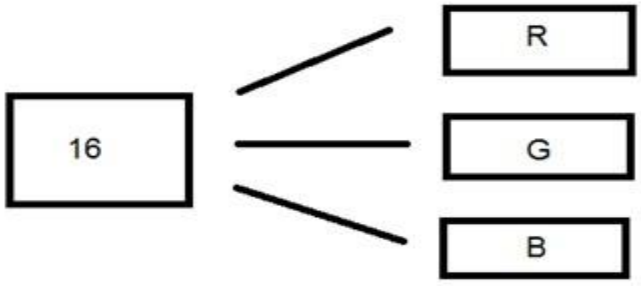


Figure 2.13 16 bit image format

Now the question arises, that how would you distribute 16 into three. If you do it like this,

5 bits for R, 5 bits for G, 5 bits for B

Then there is one bit remains in the end.

So the distribution of 16 bit has been done like this.

5 bits for R, 6 bits for G, 5 bits for B.

The additional bit that was left behind is added into the green bit. Because green is the color which is most soothing to eyes in all of these three colors.

Note this is distribution is not followed by all the systems. Some have introduced an alpha channel in the 16 bit.

Another distribution of 16 bit format is like this:

4 bits for R, 4 bits for G, 4 bits for B, 4 bits for alpha channel.

Or some distribute it like this

5 Bits for R, 5 bits for G, 5 bits for B, 1 bits for alpha channel.

2.2.3.4.4 24 bit color format

24 bit color format also known as true color format. Like 16 bit color format, in a 24 bit color format, the 24 bits are again distributed in three different formats of Red, Green and Blue.

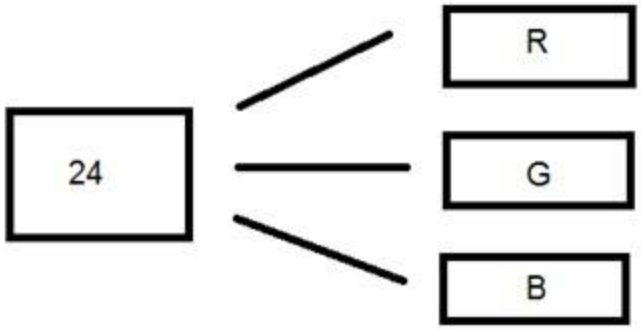


Figure 2.14 24 bit image format

Since 24 is equally divided on 8 , so it has been distributed equally between three different color channels.

Their distribution is like this.

8 bits for R, 8 bits for G, 8 bits for B.

Behind a 24 bit image.

Unlike a 8 bit gray scale image, which has one matrix behind it, a 24 bit image has three different matrices of R, G, B.

Figure 2.15 24 bit image

22.5 Histograms Introduction

Before discussing the use of Histograms in image processing , we will first look at what histogram is, how it is used and then an example of histograms to have more understanding of histogram.

2.2.5.1 Histograms

A histogram is a graph. A graph that shows frequency of anything. Usually histogram have bars that represent frequency of occurring of data in the whole data set.

A Histogram has two axis the x axis and the y axis.

The x axis contains event whose frequency you have to count.

The y axis contains frequency.

The different heights of bar shows different frequency of occurrence of data.

Usually a histogram looks like this.

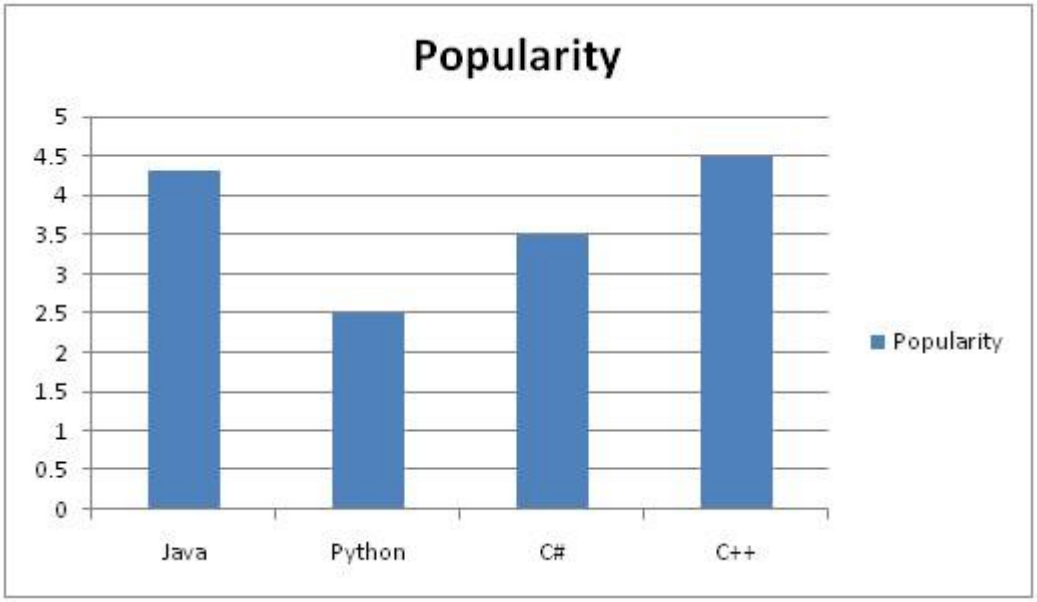


Figure 2.16 illustration of histogram

Now we will see an example of this histogram is build

Example:

Consider a class of programming students and you are teaching python to them.

At the end of the semester, you got this result that is shown in table. But it is very messy and does not show your overall result of class. So you have to make a histogram of your result, showing the overall frequency of occurrence of grades in your class. Here how you are going to do it.

Result sheet:

Name	Grade
John	A
Jack	D
Carter	B
Tommy	A
Lisa	C+
Derek	A-
Tom	B+

Table 2.1 illustitive example of histogram

Histogram of result sheet:

Now what you are going to do is, that you have to find what comes on the x and the y axis.

There is one thing to be sure, that y axis contains the frequency, so what comes on the x axis. X axis contains the event whose frequency has to be calculated. In this case x axis contains grades.

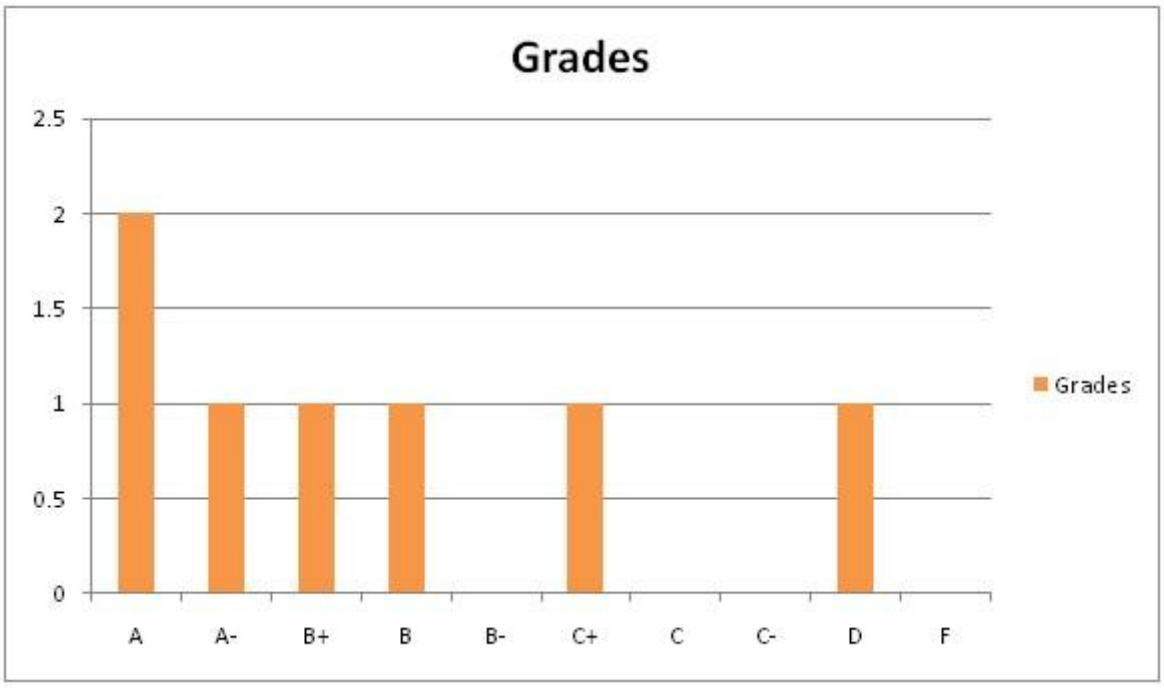


Figure 2.17 Histogram of result sheet

Now we will how we use a histogram in an image.

Histogram of an image

Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels intensity values. In an image histogram, the x axis shows the gray level intensities and the y axis shows the frequency of these intensities.

For example:

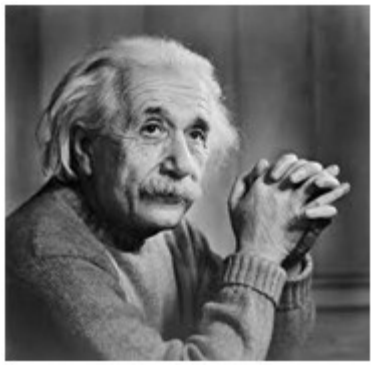


Figure 2.18 gray image

The histogram of the above picture of the Einstein would be something like this

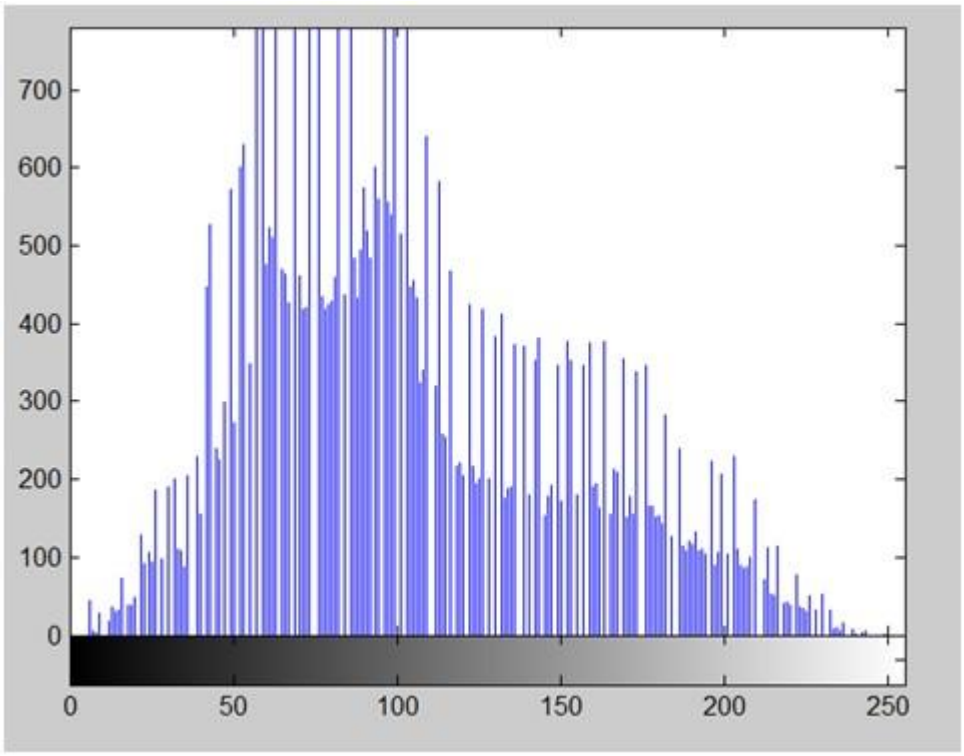


Figure 2.19 histogram of a gray image

The x axis of the histogram shows the range of pixel values. Since its an 8 bpp image, that means it has 256 levels of gray or shades of gray

in it. That's why the range of x axis starts from 0 and end at 255 with a gap of 50. Whereas on the y axis, is the count of these intensities.

As you can see from the graph, that most of the bars that have high frequency lies in the first half portion which is the darker portion. That means that the image we have got is darker. And this can be proved from the image too.

2.2.5.2 Applications of Histograms:

Histograms has many uses in image processing. The first use as it has also been discussed above is the analysis of the image. We can predict about an image by just looking at its histogram. It's like looking an x ray of a bone of a body.

The second use of histogram is for brightness purposes. The histograms has wide application in image brightness. Not only in brightness, but histograms are also used in adjusting contrast of an image.

Another important use of histogram is to equalize an image.

And last but not the least, histogram has wide use in thresholding. This is mostly used in computer vision.

2.2.5.2.1Histogram stretching

One of the other advantage of Histogram s that we discussed in our tutorial of introduction to histograms is contrast enhancement.

There are two methods of enhancing contrast. The first one is called Histogram stretching that increase contrast. The second one is called Histogram equalization that enhance contrast and it has been discussed in our tutorial of histogram equalization.

Before we `will discuss the histogram stretching to increase contrast, we will briefly define contrast.

Contrast.

Contrast is the difference between maximum and minimum pixel intensity.

Consider this image.

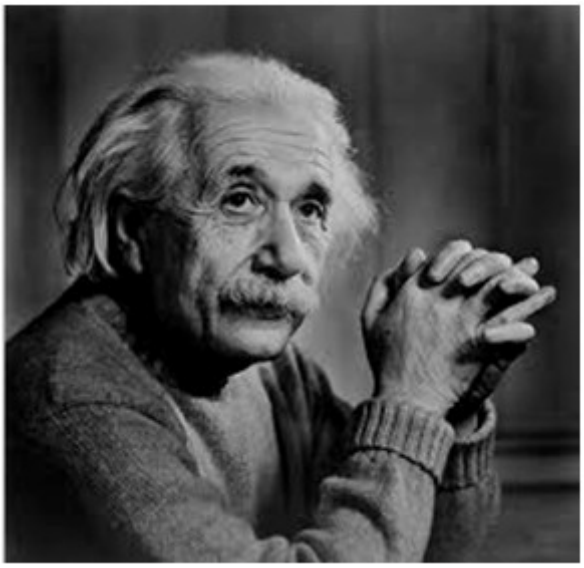


Figure 2.20 illustrative image for contrast

The histogram of this image is shown below.

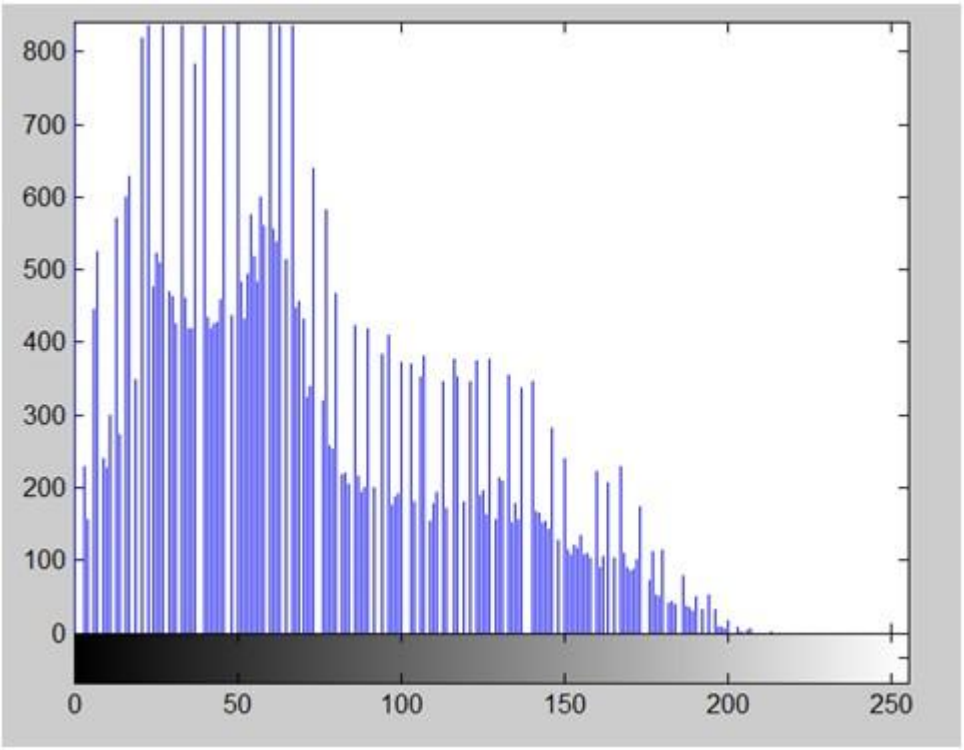


Figure 2.21 histogram of the illustrative image

Now we calculate contrast from this image.

Contrast = 225.

Now we will increase the contrast of the image.

Increasing the contrast of the image:

The formula for stretching the histogram of the image to increase the

$$g(x,y) = \frac{f(x,y)-fmin}{fmax- fmin} * 2^{bpp}$$

contrast is

The formula requires finding the minimum and maximum pixel intensity multiply by levels of gray. In our case the image is 8bpp, so levels of gray are 256.

The minimum value is 0 and the maximum value is 225. So the formula in our case is

$$g(x,y) = \frac{f(x,y)- 0}{225 - 0} * 255$$

Where f(x,y) denotes the value of each pixel intensity. For each f(x,y) in an image , we will calculate this formula.

After doing this, we will be able to enhance our contrast.

The following image appear after applying histogram stretching.

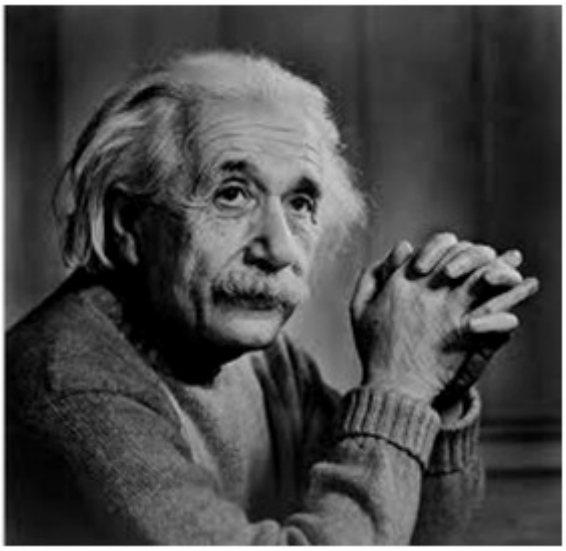


Figure 2.22 image of stretched histogram

The stretched histogram of this image has been shown below.

Note the shape and symmetry of histogram. The histogram is now stretched or in other means expand. Have a look at it.

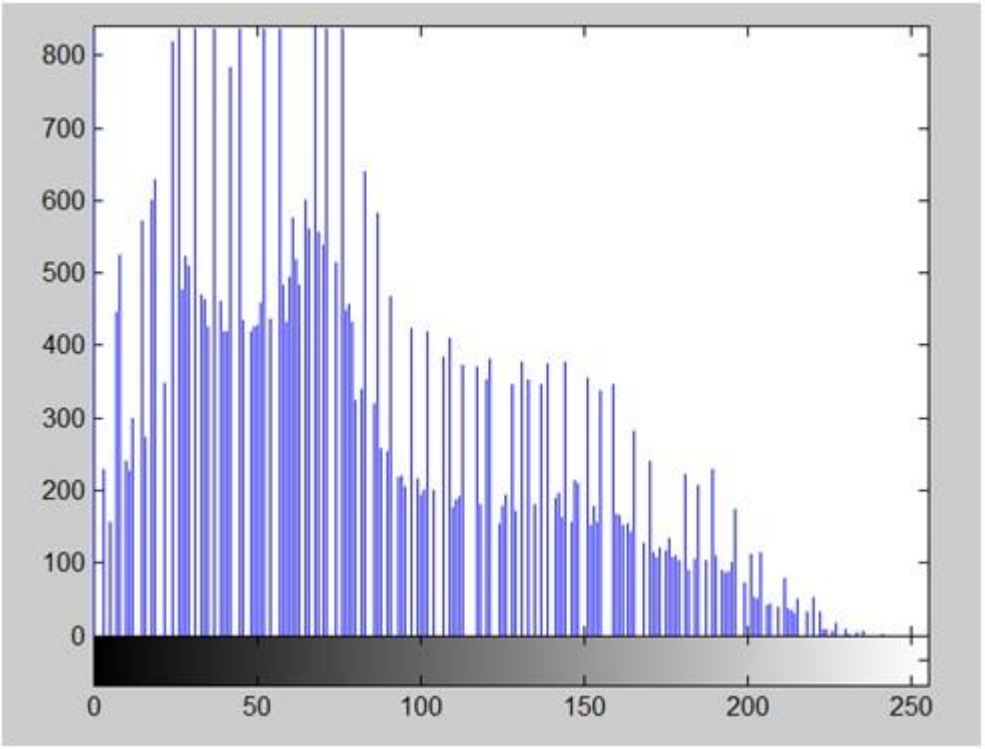


Figure 2.23 stretched histogram

In this case the contrast of the image can be calculated as

Contrast = 240

Hence we can say that the contrast of the image is increased.

Note: this method of increasing contrast doesnot work always, but it fails on some cases.

Failing of histogram stretching

As we have discussed, that the algorithm fails on some cases. Those cases include images with when there is pixel intensity 0 and 255 are present in the image

Because when pixel intensities 0 and 255 are present in an image, then in that case they become the minimum and maximum pixel intensity which ruins the formula like this.

Original Formula

$$g(x,y) = \frac{f(x,y)-fmin}{fmax- fmin} * 2^{bpp}$$

Putting fail case values in the formula:

$$g(x,y) = \frac{f(x,y)- 0}{255 - 0} * 255$$

Simplify that expression gives

$$g(x,y) = \frac{f(x,y)}{255} * 255$$

$$g(x,y) = f(x,y)$$

That means the output image is equal to the processed image. That means there is no effect of histogram stretching has been done at this image.

2.2.5.2.2 Histogram Equalization

We have already seen that contrast can be increased using histogram stretching. In this tutorial we will see that how histogram equalization can be used to enhance contrast.

Before performing histogram equalization, you must know two important concepts used in equalizing histograms. These two concepts are known as PMF and CDF.

They are discussed in our tutorial of PMF and CDF. Please visit them in order to successfully grasp the concept of histogram equalization.

Histogram Equalization:

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram equalization can be worse. In that cases the contrast is decreased.

Lets start histogram equalization by taking this image below as a simple image.

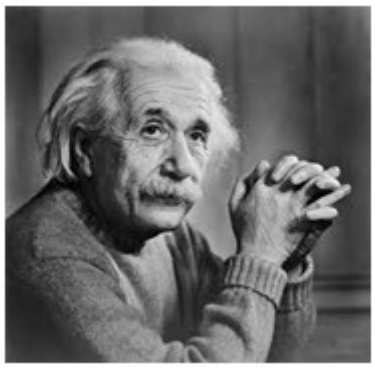


Figure 2.24 simple image before histogram equalization

Histogram of this image:

The histogram of this image has been shown below.

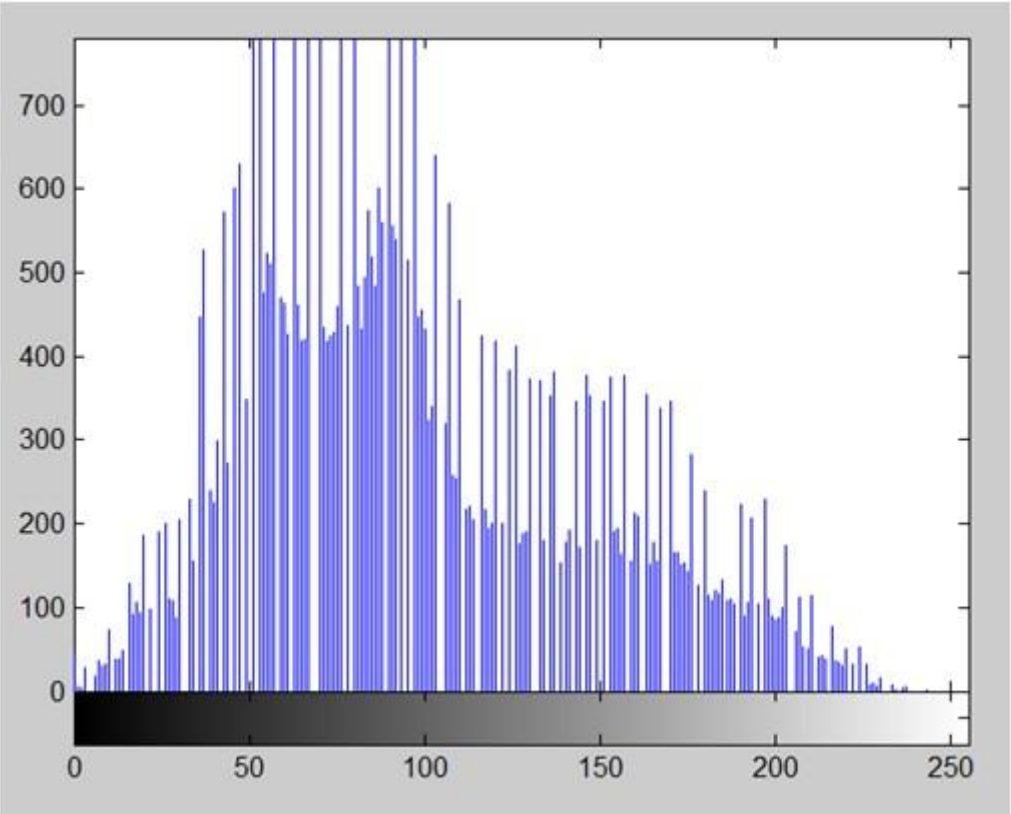


Figure 2.25 the histogram before histogram equalization

Now we will perform histogram equalization to it.

PMF:

First we have to calculate the PMF (probability mass function) of all the pixels in this image. If you don't know how to calculate PMF, please visit our tutorial of PMF calculation.

CDF:

Our next step involves calculation of CDF (cumulative distributive function). Again if you don't know how to calculate CDF, please visit our tutorial of CDF calculation.

Calculate CDF according to gray levels

Let's for instance consider this that the CDF calculated in the second step looks like this.

Gray Level Value	CDF
0	0.11
1	0.22
2	0.55
3	0.66
4	0.77
5	0.88
6	0.99
7	1

Table 2.2 Calculate CDF according to gray

Then in this step you will multiply the CDF value with (Gray levels (minus) 1).

Considering we have a 3 bpp image. Then number of levels we have are 8. And 1 subtracts 8 is 7. So we multiply CDF by 7. Here what we got after multiplying.

Gray Level Value	CDF	CDF * (Levels-1)
0	0.11	0
1	0.22	1
2	0.55	3
3	0.66	4
4	0.77	5
5	0.88	6
6	0.99	6
7	1	7

Table 2.3 CDF multiplying by(Levels-1)

Now we have is the last step, in which we have to map the new gray level values into number of pixels.

Lets assume our old gray levels values has these number of pixels.

Gray Level Value	Frequency
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16

Table 2.4 frequency of gray level values

Now if we map our new values to, then this is what we got.

Gray Level Value	New Gray Level Value	Frequency
0	0	2
1	1	4
2	3	6
3	4	8
4	5	10
5	6	12
6	6	14
7	7	16

Table 2.5 mapping values after histogram equalization

Now map these new values you are onto histogram, and you are done.

Let's apply this technique to our original image. After applying we got the following image and its following histogram.

Histogram Equalization Image

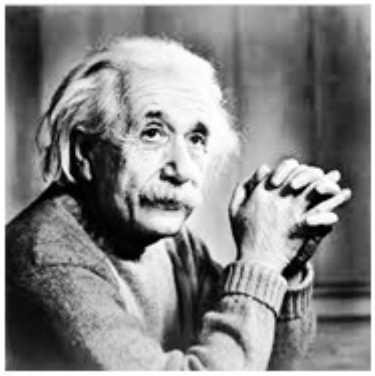


Figure 2.26 Histogram Equalization Image

Cumulative Distributive function of this image

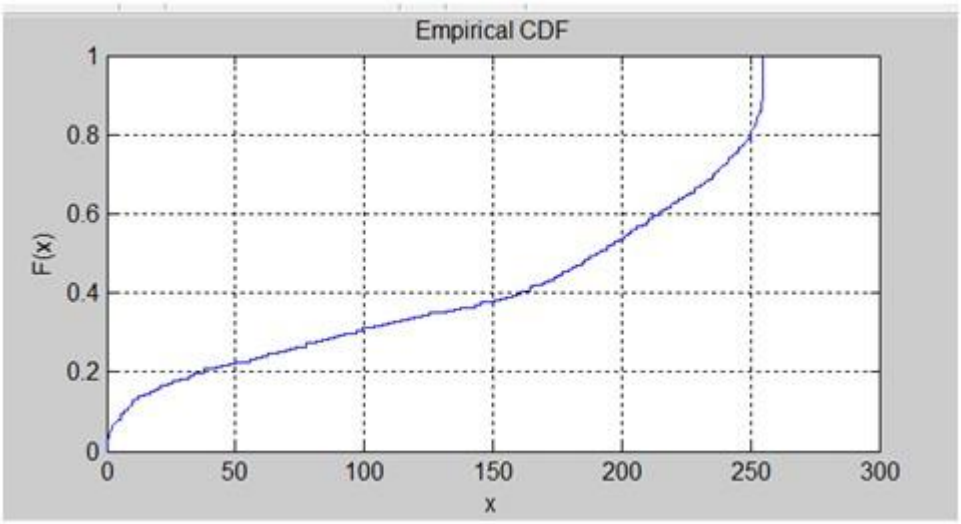


Figure 2.27 Cumulative Distributive function of this image

Histogram Equalization histogram

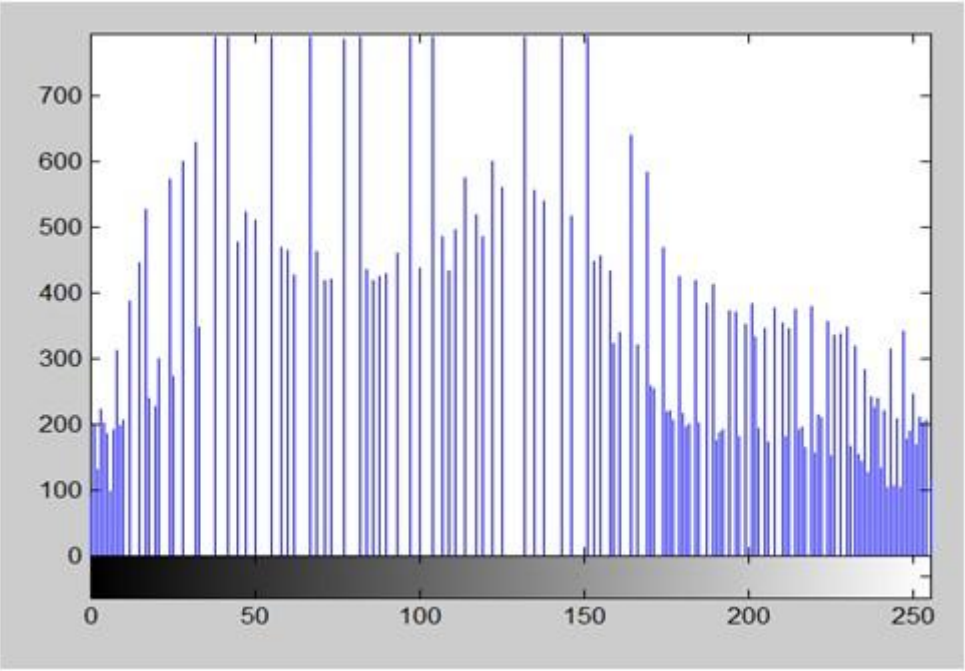


Figure 2.28 Histogram Equalization histogram

Comparing both the histograms and images

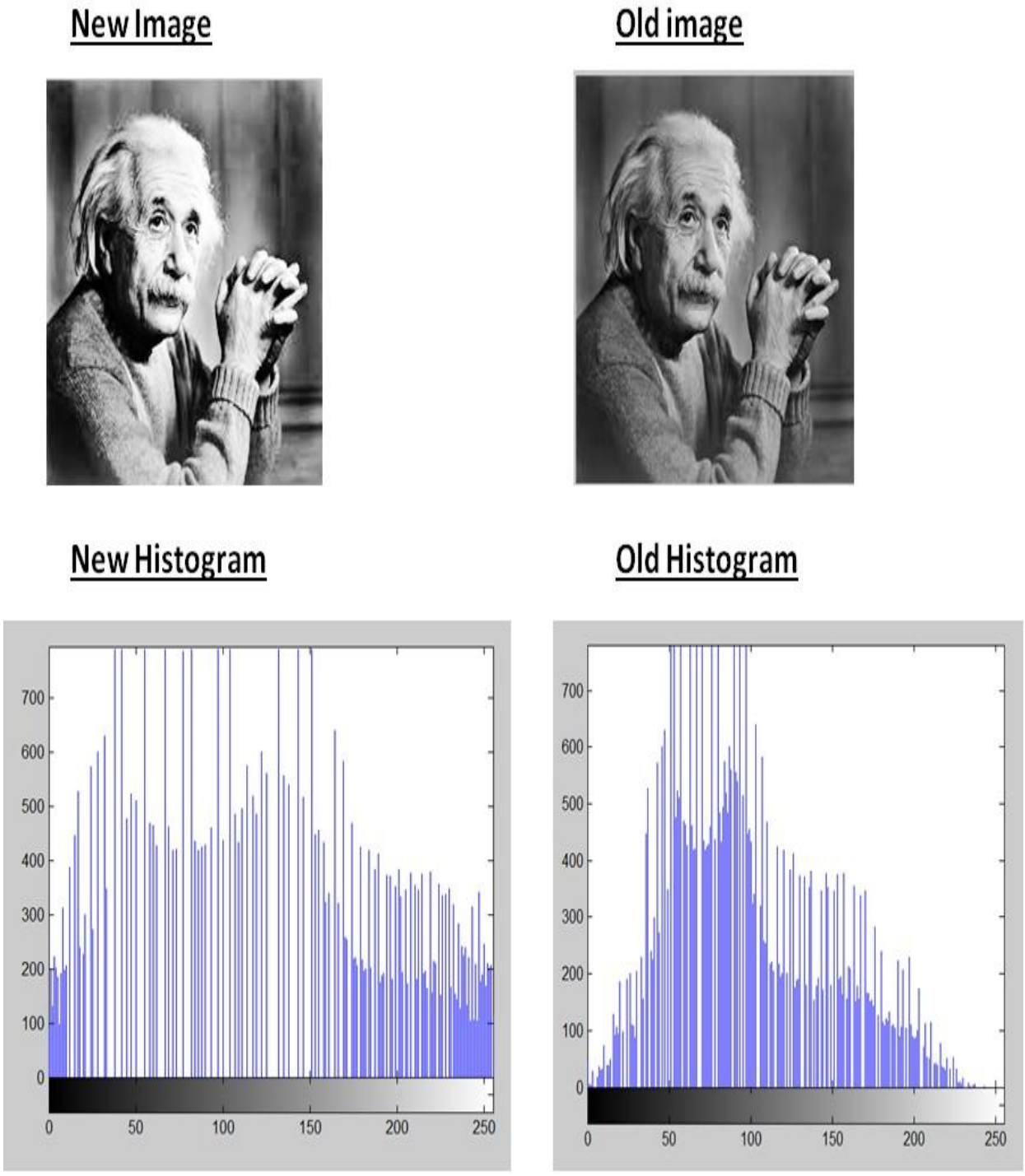


Figure 2.29 Comparing both the histograms and images

Conclusion

As you can clearly see from the images that the new image contrast has been enhanced and its histogram has also been equalized. There is also one

important thing to be note here that during histogram equalization the overall shape of the histogram changes, where as in histogram stretching the overall shape of histogram remains same.[15]

2.2.6 Digital Filters

In image processing filters are mainly used to suppress either the high frequencies in the image, *i.e.* smoothing the image, or the low frequencies, *i.e.* enhancing or detecting edges in the image.

An image can be filtered either in the frequency or in the spatial domain.

The first involves transforming the image into the frequency domain, multiplying it with the frequency filter function and re-transforming the result into the spatial domain. The filter function is shaped so as to attenuate some frequencies and enhance others. For example, a simple low pass function is *1* for frequencies smaller than the *cut-off frequency* and *0* for all others.

The corresponding process in the spatial domain is to convolve the input image $f(i,j)$ with the filter function $h(i,j)$. This can be written as

$$g(i,j) = h(i,j) \odot f(i,j)$$

The mathematical operation is identical to the multiplication in the frequency space, but the results of the digital implementations vary, since we have to approximate the filter function with a discrete and finite kernel.

The discrete convolution can be defined as a *`shift and multiply'* operation, where we shift the kernel over the image and multiply its value with the corresponding pixel values of the image. For a square kernel with size $M \times M$, we can calculate the output image with the following formula:

$$g(i,j) = \sum_{m=-\frac{M}{2}}^{\frac{M}{2}} \sum_{n-=\frac{M}{2}}^{\frac{M}{2}} h(m,n)\, f(i-m, j-n)$$

Various standard kernels exist for specific applications, where the size and the form of the kernel determine the characteristics of the operation. The most important of them are discussed in this chapter. The kernels for two examples, the mean and the Laplacian operator, can be seen in Figure 1.

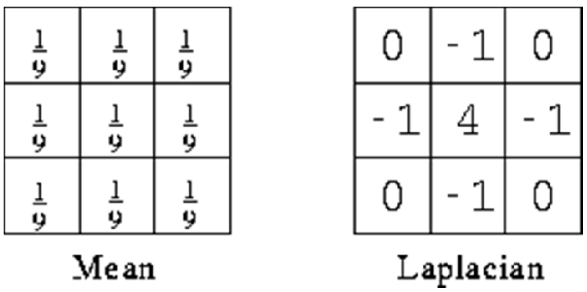


Figure 2.30 Convolution kernel for a mean filter and one form of the discrete Laplacian.

In contrast to the frequency domain, it is possible to implement non-linear filters in the spatial domain. In this case, the summations in the convolution function are replaced with some kind of non-linear operator:

$$g(i,j) = O_{m,n}[h(m,n)\ f(i-m, j-n)]$$

For most non-linear filters the elements of $h(i,j)$ are all 1. A commonly used non-linear operator is the median, which returns the `middle' of the input values.

2.2.6.1Median Filter

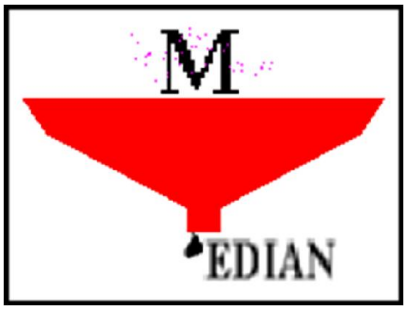


Figure 2.31 median filter. Symbol

2.2.6.1.1Brief Description

The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image.

2.2.6.1.2How It Works

Like the mean filter, the median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the *mean* of neighboring pixel values, it replaces it with the *median* of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value. (If the neighborhood under consideration contains an even number of pixels, the average of the two middle pixel values is used.) Figure 1 illustrates an example calculation.

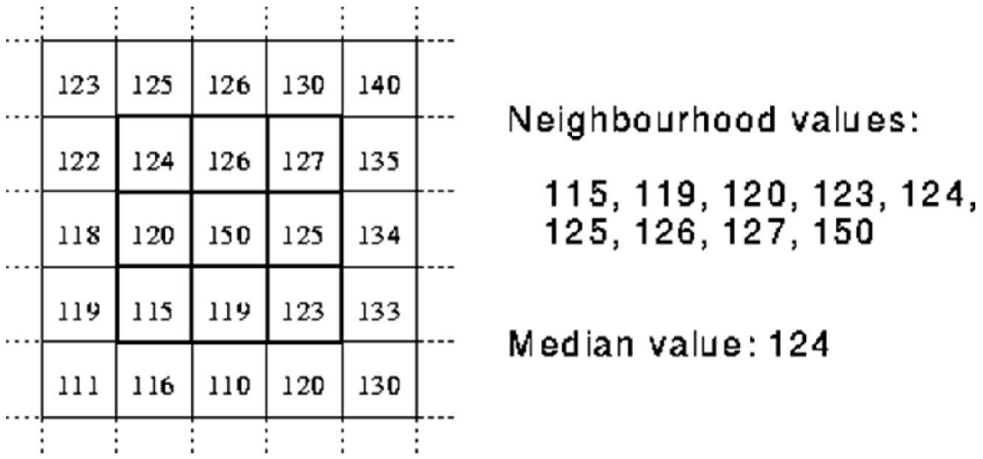


Figure 2.32 calculating the median value of a pixel neighborhood.

As can be seen, the central pixel value of 150 is rather unrepresentative of the surrounding Pixels and is replaced with the median value: 124. A 3×3 square neighborhood is used here --- larger neighborhoods will produce more severe smoothing.

2.2.6.1.3 Guidelines for Use

By calculating the median value of a neighborhood rather than the mean filter, the median filter has two main advantages over the mean filter:

- The median is a more robust average than the mean and so a single very unrepresentative pixel in a neighborhood will not affect the median value significantly.
- Since the median value must actually be the value of one of the pixels in the neighborhood, the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter.

The image

Figure 2.33 image corrupted by Gaussian noise

Shows an image that has been corrupted by Gaussian noise with mean 0 and standard deviation (σ) 8. The original image is

Figure 2.34 the original image before Gaussian noise

For comparison. Applying a 3×3 median filter produces

Figure 2.35 image with applying a 3×3 median filter

Note how the noise has been reduced at the expense of a slight degradation in image quality. The image

Figure 2.36 image with noise

Has been corrupted by even more noise (Gaussian noise with mean 0 and σ 13), and

Figure 2.37 image corrupted by even more noise

Is the result of 3×3 median filtering? The median filter is sometimes not as subjectively good at dealing with large amounts of Gaussian noise as the mean filter.

Where median filtering really comes into its own is when the noise produces extreme `outlier' pixel values, as for instance in

Figure 2.38 image with noise produces extreme `outlier' pixel values

Which has been corrupted with `salt and pepper' noise, *i.e.* bits have been flipped with probability 1%. Median filtering this with a 3×3 neighborhood produces

Figure 2.39 image with 3×3 neighborhood Median filtering

in which the noise has been entirely eliminated with almost no degradation to the underlying image. Compare this with the similar test on the mean filter.

Consider another example wherein the original image

Figure 2.40 original image before salt and pepper noise

Has been corrupted with higher levels (*i.e.* p=5% that a bit is flipped) of salt and pepper noise

Figure 2.41 image after corrupting by salt and pepper noise

After smoothing with a 3×3 filter, most of the noise has been eliminated

Figure 2.42 image after smoothing with a 3×3 filter

If we smooth the noisy image with a larger median filter, *e.g.* 7×7, all the noisy pixels disappear, as shown in

Figure 2.43 image with smoothing by 7×7 median filter

Note that the image is beginning to look a bit `blotchy', as graylevel regions are mapped together. Alternatively, we can pass a 3×3 median filter over the image three times in order to remove all the noise with less loss of detail

Figure 2.44 image with applying 3×3 Median filtering for 3 times

In general, the median filter allows a great deal of high spatial frequency detail to pass while remaining very effective at removing noise on images where less than half of the pixels in a smoothing neighborhood have been effected. (As a consequence of this, median filtering can be less effective at removing noise from images corrupted with Gaussian noise.)

One of the major problems with the median filter is that it is relatively expensive and complex to compute. To find the median it is necessary to sort all the values in the neighborhood into numerical order and this is relatively slow, even with fast sorting algorithms such as *quicksort*. The basic algorithm can, however, be enhanced somewhat for speed. A

common technique is to notice that when the neighborhood window is slid across the image, many of the pixels in the window are the same from one step to the next, and the relative ordering of these with each other will obviously not have changed. Clever algorithms make use of this to improve performance.[6]

2.2.7 Converting color to grayscale

Conversion of a color image to grayscale is not unique; different weighting of the color channels effectively represent the effect of shooting black-and-white film with different-colored photographic filters on the cameras.

2.2.7.1 Colorimetric (luminance-preserving) conversion to grayscale

A common strategy is to use the principles of photometry or, more broadly, colorimetry to match the luminance of the grayscale image to the luminance of the original color image.[2][3] This also ensures that both images will have the same absolute luminance, as can be measured in its SI units of candelas per square meter, in any given area of the image, given equal whitepoints. In addition, matching luminance provides matching perceptual lightness measures, such as L^* (as in the 1976 CIE *L*ab color space) which is determined by the luminance Y (as in the CIE 1931 *XYZ* color space) .

To convert a color from a colorspace based on an RGB color model to a grayscale representation of its luminance, weighted sums must be calculated in a linear RGB space, that is, after the gamma compression function has been removed first via gamma expansion.[4]

For the sRGB color space, gamma expansion is defined as

$$C_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \leq 0.04045 \\ \left(\frac{C_{\text{srgb}}+0.055}{1.055}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

where C_{srgb} represents any of the three gamma-compressed sRGB primaries (R_{srgb}, G_{srgb}, and B_{srgb}, each in range [0,1]) and C_{linear} is the corresponding linear-intensity value (R, G, and B, also in range [0,1]). Then, luminance is calculated as a weighted sum of the three linear-intensity values. The sRGB color space is defined in terms of the CIE 1931 linear luminance Y, which is given by

$$Y = 0.2126R + 0.7152G + 0.0722B.$$ [5]

The coefficients represent the measured intensity perception of typical trichromat humans, depending on the primaries being used; in particular, human vision is most sensitive to green and least sensitive to blue. To encode grayscale intensity in linear RGB, each of the three primaries can be set to equal the calculated linear luminance Y (replacing R,G,B by Y,Y,Y to get this linear grayscale). Linear luminance typically needs to be gamma compressed to get back to a conventional non-linear representation. For sRGB, each of its three primaries is then set to the same gamma-compressed Y_{srgb} given by the inverse of the gamma expansion above as

$$Y_{\text{srgb}} = \begin{cases} 12.92\ Y, & Y \leq 0.0031308 \\ 1.055\ Y^{1/2.4} - 0.055, & Y > 0.0031308. \end{cases}$$

In practice, because the three sRGB components are then equal, it is only necessary to store these values once in sRGB-compatible image formats that support a single-channel representation. Web browsers and other software that recognizes sRGB images will typically produce the same rendering for a such a grayscale image as it would for an sRGB image having the same values in all three color channels.

2.2.7.2 Luma coding in video systems

For images in color spaces such as Y'UV and its relatives, which are used in standard color TV and video systems such as PAL, SECAM, and NTSC, a nonlinear luma component (Y') is calculated directly from gamma-compressed primary intensities as a weighted sum, which can be calculated quickly without the gamma expansion and compression used in colorimetric grayscale calculations. In the Y'UV and Y'IQ models used by PAL and NTSC, the rec601 luma (Y') component is computed as

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

where we use the prime to distinguish these gamma-compressed values from the linear R, G, B, and Y discussed above. The ITU-R BT.709 standard used for HDTV developed by the ATSC uses different color coefficients, computing the luma component as

$$Y' = 0.2126R' + 0.7152G' + 0.0722B'.$$

Although these are numerically the same coefficients used in sRGB above, the effect is different because here they are being applied directly to gamma-compressed values.

Normally these colorspaces are transformed back to R'G'B' before rendering for viewing. To the extent that enough precision remains, they can then be rendered accurately.

But if the luma component by itself is instead used directly as a grayscale representation of the color image, luminance is not preserved: two colors can have the same luma Y' but different CIE linear luminance Y (and thus different nonlinear Y_{srgb} as defined above) and therefore appear darker or lighter to a typical human than the original color. Similarly, two colors having the same luminance Y (and thus the same Y_{srgb}) will in general have different luma by either of the Y' luma definitions above.

2.3 Artificial Neural Network

2.3.1 Introduction

Computers are great at solving algorithmic and math problems, but often the world can't easily be defined with a mathematical algorithm. Facial recognition and language processing are a couple of examples of problems that can't easily be quantified into an algorithm, however these tasks are trivial to humans. The key to Artificial Neural Networks is that their design enables them to process information in a similar way to our own biological brains, by drawing inspiration from how our own nervous system functions. This makes them useful tools for solving problems like facial recognition, which our biological brains can do easily.[5]

2.3.2 How do they work?

First lets take a look at what a biological neuron looks like.

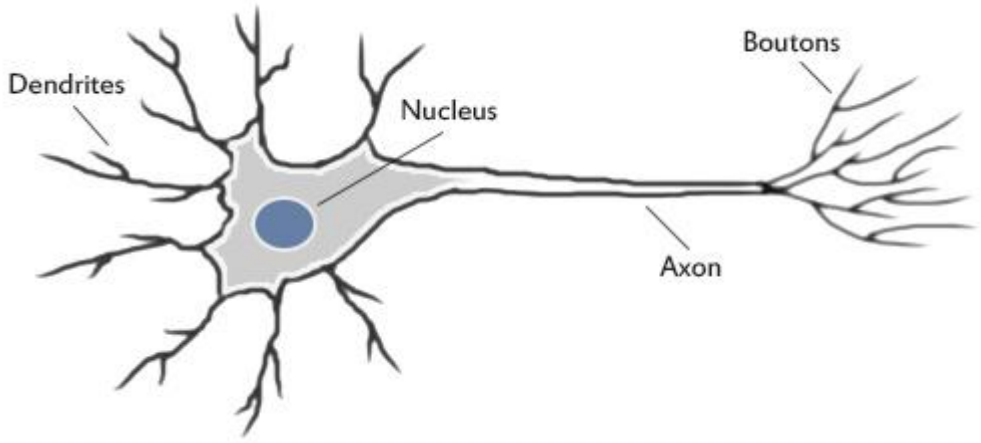


Figure 2.45 biology neuron

Our brains use extremely large interconnected networks of neurons to process information and model the world we live in. Electrical inputs are passed through this network of neurons which result in an output being produced. In the case of a biological brain this could result in contracting a muscle or signaling your sweat glands to produce sweat. A neuron collects inputs using a structure called dendrites, the neuron effectively sums all of these inputs from the dendrites and if the resulting value is greater than it's firing threshold, the neuron fires. When the neuron fires it sends an electrical impulse through the neuron's axon to it's boutons. These boutons

can then be networked to thousands of other neurons via connections called synapses. There are about one hundred billion (100,000,000,000) neurons inside the human brain each with about one thousand synaptic connections. It's effectively the way in which these synapses are wired that give our brains the ability to process information the way they do.

2.3.3 Modeling Artificial Neurons

Artificial neuron models are at their core simplified models based on biological neurons. This allows them to capture the essence of how a biological neuron functions. We usually refer to these artificial neurons as 'perceptions'. So now let's take a look at what a perceptron looks like.

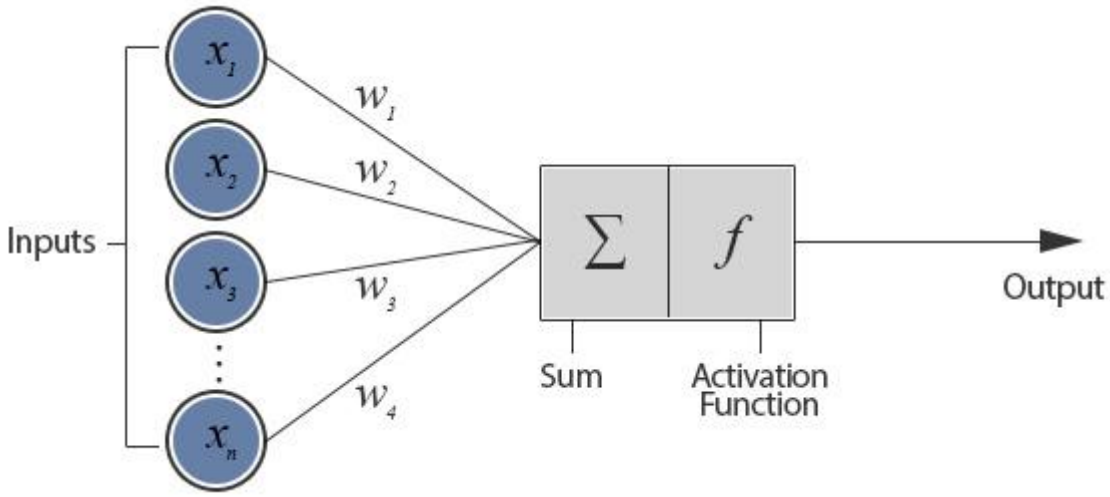


Figure 2.46 neuron mathematical calculation

As shown in the diagram above a typical perceptron will have many inputs and these inputs are all individually weighted. The perceptron weights can either amplify or deamplify the original input signal. For example, if the input is 1 and the input's weight is 0.2 the input will be decreased to 0.2. These weighted signals are then added together and passed into the activation function. The activation function is used to convert the input into a more useful output. There are many different types of activation function but one of the simplest would be step function. A step function will typically output a 1 if the input is higher than a certain threshold, otherwise

it's output will be 0.

Here's an example of how this might work:

Input 1 (x_1) = 0.6

Input 2 (x_2) = 1.0

Weight 1 (w_1) = 0.5

Weight 2 (w_2) = 0.8

Threshold = 1.0

First we multiple the inputs by their weights and sum them:

$x_1w_1 + x_2w_2 = (0.6 \times 0.5) + (1 \times 0.8) = 1.1$

Now we compare our input total to the perceptron's activation threshold. In this example the total input (1.1) is higher than the activation threshold (1.0) so the neuron would fire.

2.3.4 Implementing Artificial Neural Networks

So now you're probably wondering what an artificial neural network looks like and how it uses these artificial neurons to process information. In this tutorial we're going to be looking at feedforward networks and how their design links our perceptron together creating a functioning artificial neural network. Before we begin lets take a look at what a basic feedforward network looks like:

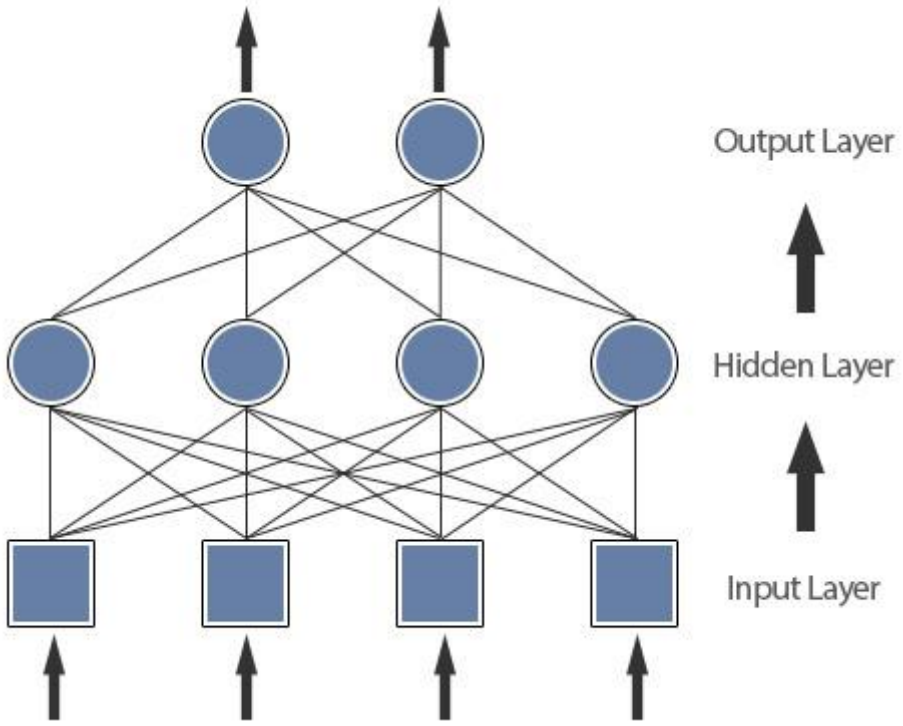


Figure 2.47 feed-forward network

Each input from the input layer is fed up to each node in the hidden layer, and from there to each node on the output layer. We should note that there can be any number of nodes per layer and there are usually multiple hidden layers to pass through before ultimately reaching the output layer. Choosing the right number of nodes and layers is important later on when optimising the neural network to work well a given problem. As you can probably tell from the diagram, it's called a feedforward network because of how the signals are passed through the layers of the neural network in a single direction. These aren't the only type of neural network though. There are also feedback networks where its architecture allows signals to travel in both directions.

2.3.5 Linear separability

To explain why we usually require a hidden layer to solve our problem, take a look at the following examples:

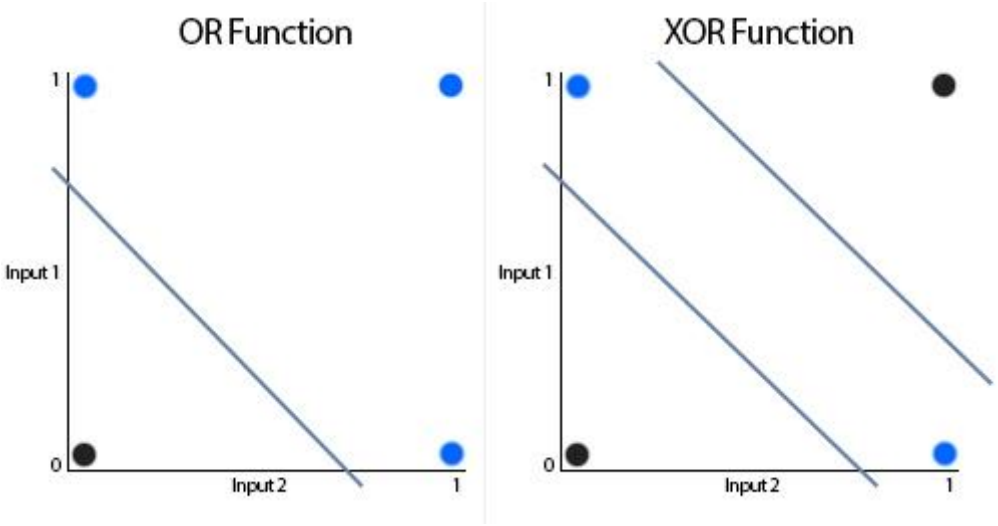


Figure 2.48 Linear separability for OR ,XOR function

Notice how the OR function can be separated on the graph with a single straight line, this means the function is "linearly separable" and can be modelled within our neural network without implementing a hidden layer, for example, the OR function can be modeled with a single perceptron like this:

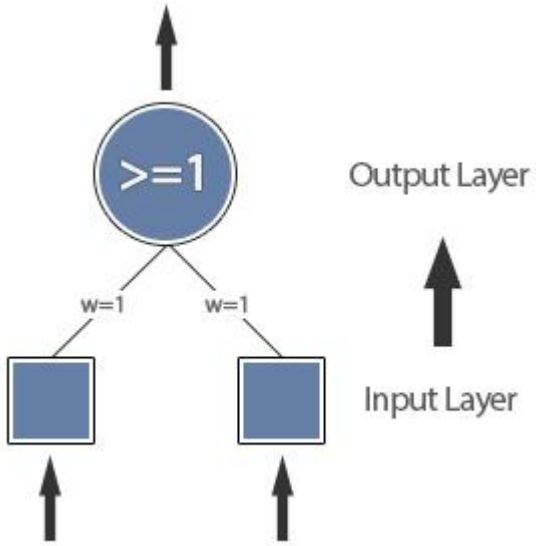


Figure 2.49 neural network of OR function

However to model the XOR function we need to use an extra layer:

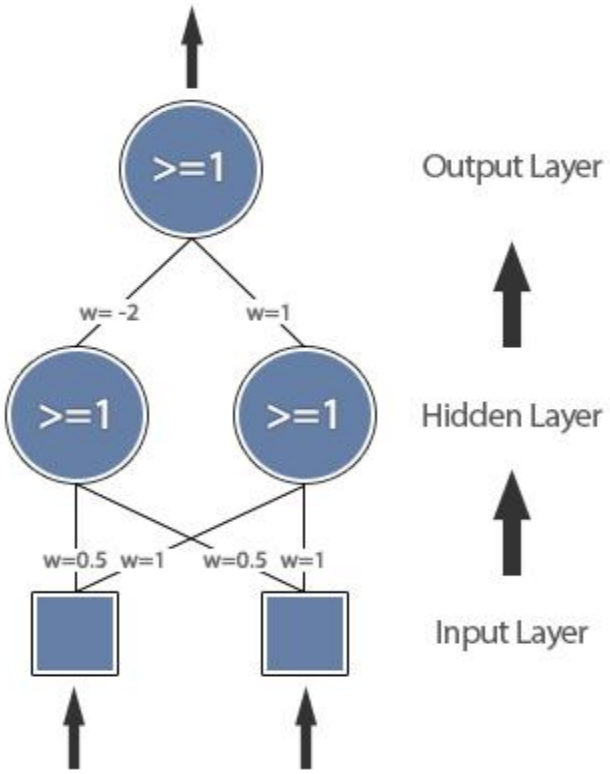


Figure 2.50 xor NN structure

We call this type of neural network a 'multi-layer perceptron'. In almost every case you should only ever need to use one or two hidden layers, however it make take more experimentation to find the optimal amount of nodes for the hidden layer(s).[8]

2.3.6 Neural Networks and Architectures

There are three fundamentally different classes of network architectures:

1. Single-layer feed-forward networks

2. Multilayer feed-forward networks

3. Recurrent networks or neural networks with feedback

2.3.6.1 Single-layer feed-forward networks

In a layered neural network, the neurons are organized in the form of layers. The simplest form of a layered network has an input layer of source nodes that project onto an output layer but not vice versa.

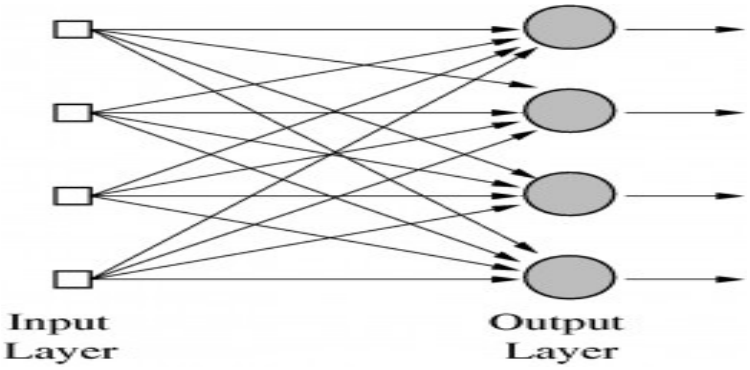


Figure 2.51 Single-layer feed-forward

The above network is feed-forward .This is also called a single-layer network. The single layer refers to the output layer as computations take place only at the output nodes.

2.3.6.2 Multilayer feed-forward networks

In this class, a neural network has one or more hidden layers, whose computation nodes are called hidden neurons or hidden units. The function of hidden neurons is to intervene between the external input and the network output in a useful manner. By adding one or more hidden layers, the network is enabled to extract higher-order statistics. This is essentially required when the size of the input layer is large. For example,

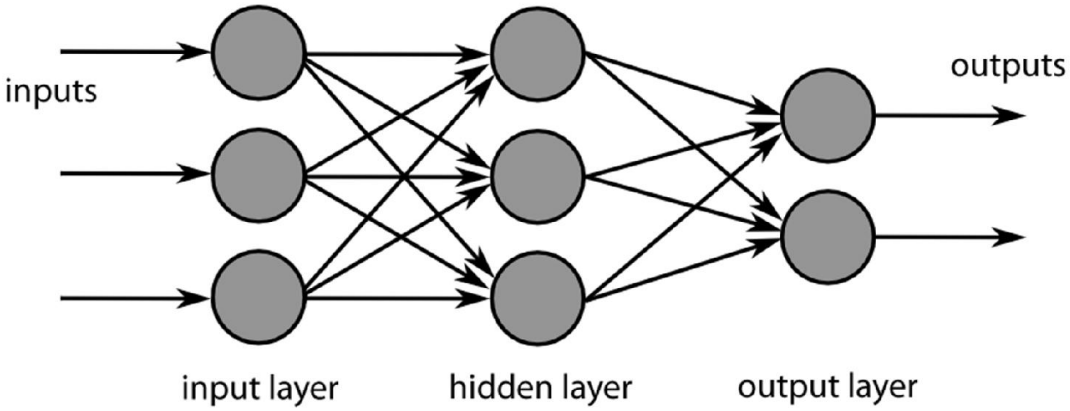


Figure 2.52 Multilayer feed-forward

2.3.6.3Recurrent neural network (RNN)

Is any network whose neurons send feedback signals to each other? This concept includes a huge number of possibilities.

Example:

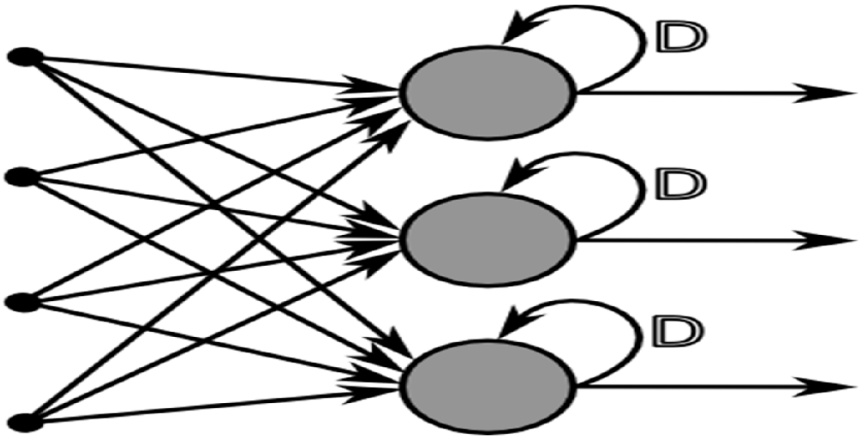


Figure 2.53 3Recurrent neural network

2.3.7 Back Propagation (BP)

The connection we're interested in is between neuron A (a hidden layer neuron) and neuron B (an output neuron) and has the weight WAB. The diagram also shows another connection, between neuron A and C, but we'll return to that later. The algorithm works like this:

1. First apply the inputs to the network and work out the output – remember this initial output could be anything, as the initial weights were random numbers.

2. Next work out the error for neuron B. The error is What you want – What you
actually get, in other words:

$$\text{ErrorB} = \text{OutputB}\ (1\text{-OutputB})(\text{TargetB} - \text{OutputB})$$

The "Output(1-Output)" term is necessary in the equation because of the Sigmoid Function – if we were only using a threshold neuron it would just be (Target –Output).

3. Change the weight. Let W+AB be the new (trained) weight and WAB be the initial weight.

$$W+AB = WAB + (\text{ErrorB} \times \text{OutputA})$$

Notice that it is the output of the connecting neuron (neuron A) we use (not B).

We update all the weights in the output layer in this way.

4. Calculate the Errors for the hidden layer neurons. Unlike the output layer we can't calculate these directly (because we don't have a Target), so we Back Propagate them from the output layer (hence the name of the algorithm). This is done by taking the Errors from the output neurons and running them back through the weights to get the hidden layer errors. For example if neuron A is connected as shown to B and C then we take the errors from B and C to generate an error for A.

$$\text{ErrorA} = \text{Output A}\ (1 - \text{Output A})(\text{ErrorB WAB} + \text{ErrorC WAC})$$

Again, the factor "Output (1 - Output)" is present because of the sigmoid squashing function.

5. Having obtained the Error for the hidden layer neurons now proceed as in stage 3 to change the hidden layer weights. By repeating this method we can train a network of any number of layers.

This may well have left some doubt in your mind about the operation, so let's clear that up by explicitly showing all the calculations for a full sized

network with 2 inputs, 3 hidden layer neurons and 2 output neurons as shown in figure 3.4. W+ represents the new, recalculated, weight, whereas W (without the superscript) represents the old weight.

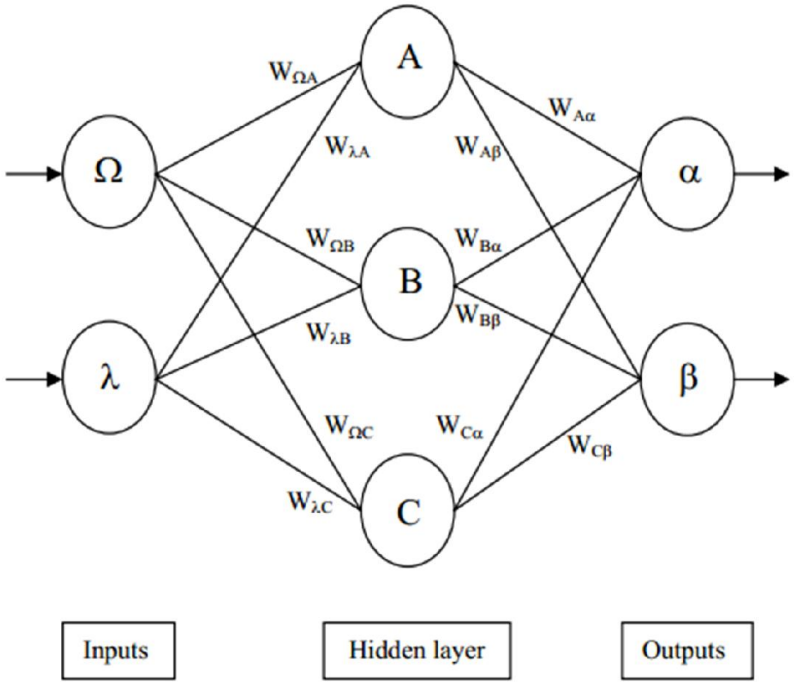


Figure 2.54 all the calculations for a reverse pass of Back Propagation

1.Calculate errors of output neurons

$$\delta\alpha = out\alpha\ (1 - out\alpha)\ (Target\alpha - out\alpha)$$

$$\delta\beta = out\beta\ (1 - out\beta)\ (Target\beta - out\beta)$$

2. Change output layer weights

$$W{+}A\alpha = WA\alpha + \eta\delta\alpha\ outA \qquad W{+}A\beta = WA\beta + \eta\delta\beta\ outA$$

$$W{+}B\alpha = WB\alpha + \eta\delta\alpha\ outB \qquad W{+}B\beta = WB\beta + \eta\delta\beta\ outB$$

$$W{+}C\alpha = WC\alpha + \eta\delta\alpha\ outC \qquad W{+}C\beta = WC\beta + \eta\delta\beta\ outC$$

3. Calculate (back-propagate) hidden layer errors

$$\delta A = outA\ (1 - outA)\ (\delta\alpha WA\alpha + \delta\beta WA\beta)$$
$$\delta B = outB\ (1 - outB)\ (\delta\alpha WB\alpha + \delta\beta WB\beta)$$
$$\delta C = outC\ (1 - outC)\ (\delta\alpha WC\alpha + \delta\beta WC\beta)$$

4. Change hidden layer weights

$$W{+}\lambda A = W\lambda A + \eta\delta A\ in\lambda \qquad W{+}\Omega A = W{+}\Omega A + \eta\delta A\ in\Omega$$
$$W{+}\lambda B = W\lambda B + \eta\delta B\ in\lambda \qquad W{+}\Omega B = W{+}\Omega B + \eta\delta B\ in\Omega$$
$$W{+}\lambda C = W\lambda C + \eta\delta C\ in\lambda \qquad W{+}\Omega C = W{+}\Omega C + \eta\delta C\ in\Omega$$

The constant η (called the learning rate, and nominally equal to one) is put in to speed up or slow down the learning if required.

Running the algorithm Now that we've seen the algorithm in detail, let's look at how it's run with a large data set. Suppose we wanted to teach a network to recognise the first four letters of the alphabet on a 5x7 grid, as shown in figure

Figure 3.5, the first four letters of the alphabet.

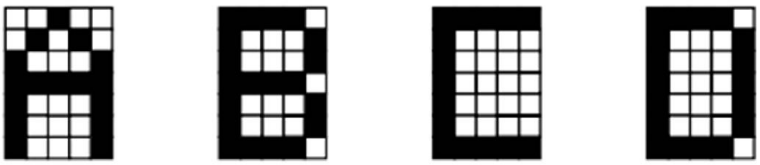


Figure 2.55 the first four letters of the alphabet

The correct way to train the network is to apply the first letter and change ALL the weights in the network ONCE . Next apply the second letter and do the same, then the third and so on. Once you have done all four letters, return to the first one again and repeat the process until the error becomes small (which means that it is recognizing all the letters).

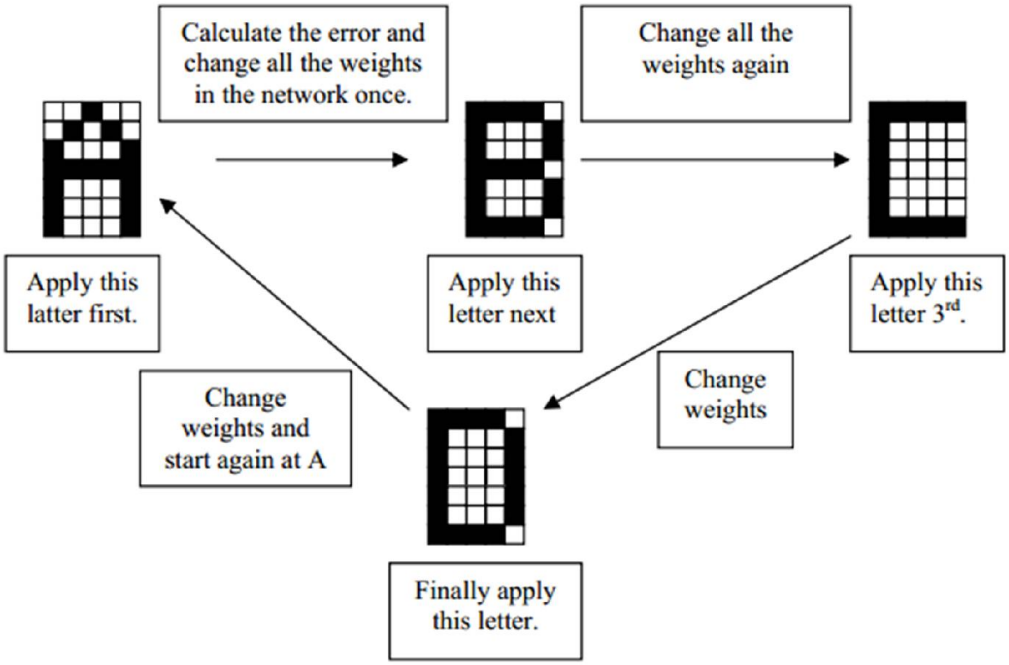


Figure 2.56 the correct running of algorithm

One of the most common mistakes for beginners is to apply the first letter to the network, run the algorithm and then repeat it until the error reduces, then apply the second letter and do the same. If you did this, the network would learn to recognize the first letter, then forget it and learn the second letter, etc. and you'd only end up with the last letter the network learned.

3.3 Stopping training when do we stop the training? We could stop it once the network can recognized all the letters successfully, but in practice it is usual to let the error fall to a lower value first. This ensures that the letters are all being well recognized. You can evaluate the total error of the network by adding up all the errors for each individual neuron and then for each pattern in turn to give you a total error as shown in figure 3.53.[2]

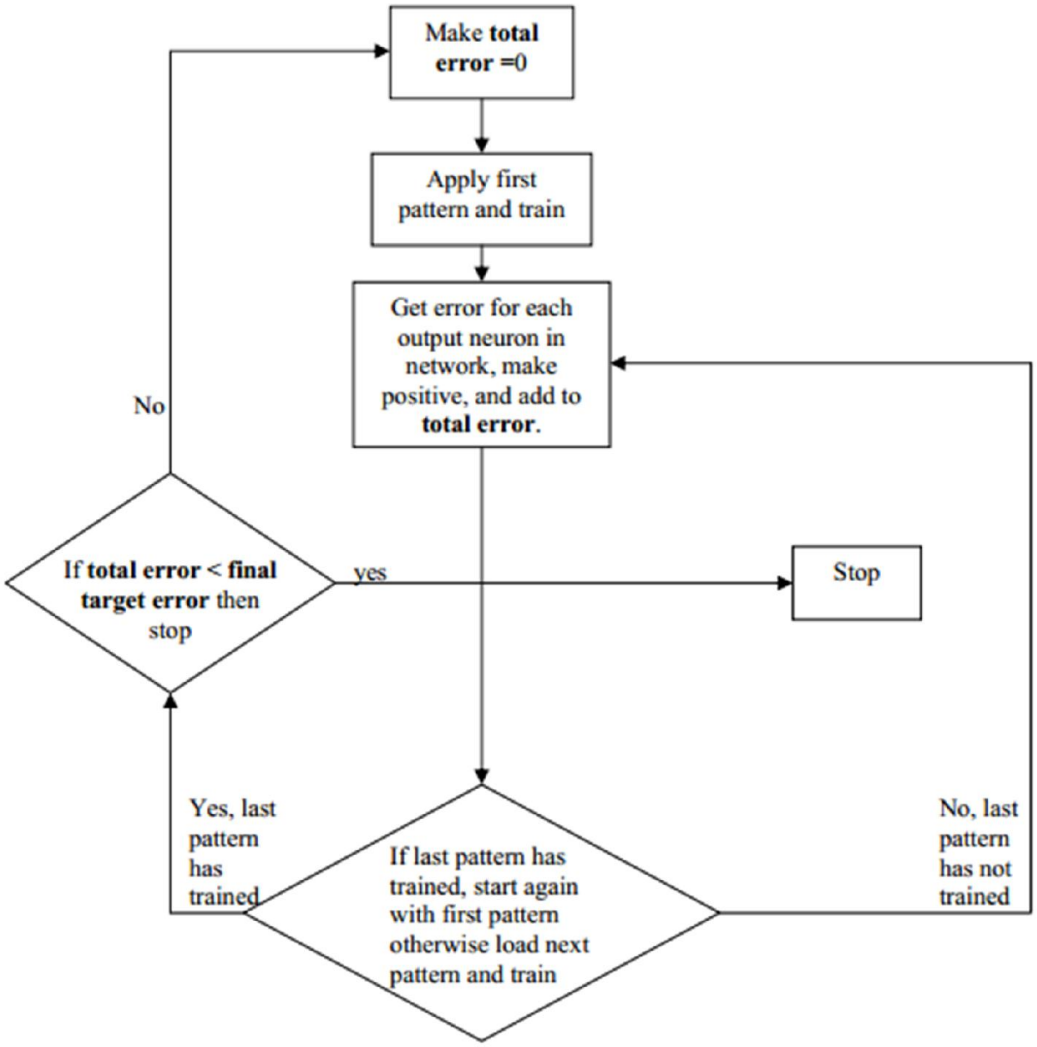


Figure 2.57 total error for work

2.3.8 Face detection

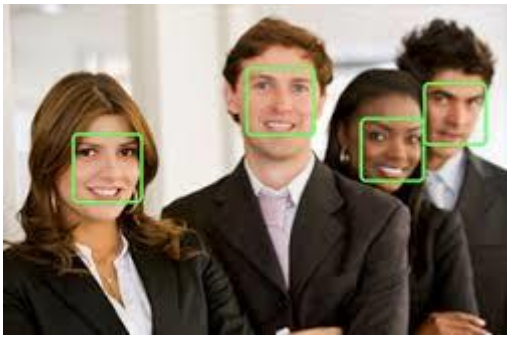


Figure 2.58 face detected image

2.3.8.1 Introduction

Face detection is a computer technology that identifies human faces in digital images. It detects human faces which might then be used for recognizing a particular face. This technology is being used in a variety of applications nowadays.

Face detection is an important first step to many advanced computer vision, biometrics recognition and multimedia applications, such as face tracking, face recognition, and video surveillance.

Face detection involves many research challenges such as scale, rotation and pose and illumination variation.

Face detection is a procedure by which we can able to extract face region from a human body. Now, the concept can be implemented in various ways but mainly we use four steps for this implementation.
In the first step, we localize the face region that means we are anticipating those parts of an image where a face may present.

In the second step we normalize the detected region, so that the alignments of various facial features are in the proper location.

In the third step we extract various facial features like eyes, nose, mouth, etc.

And in the fourth step, we actually verify whether the anticipated parts are actually carrying out a face or not.

We are doing this using some rules, template or image databases. The concept of extraction can be implemented by various techniques.

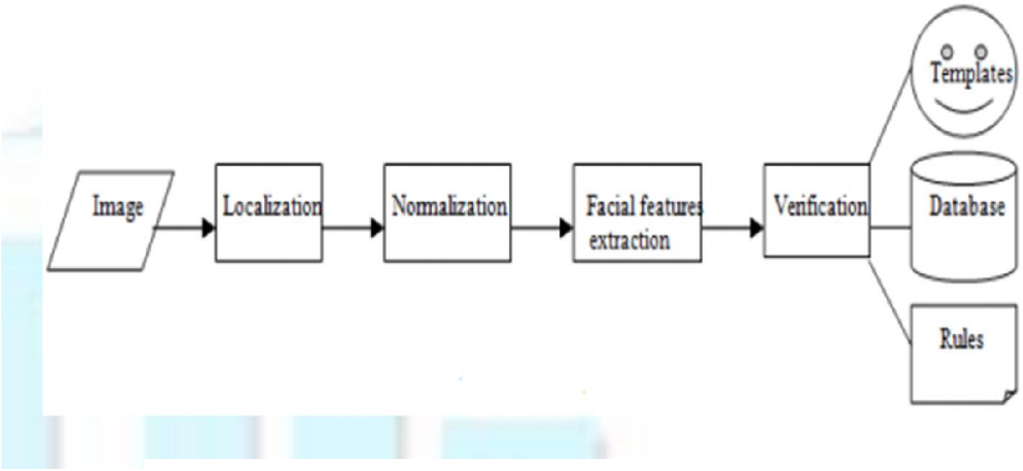


Figure 2.58 Steps of face detection

2.3.8.1.1 Localization

In this step we locate the region of an image where the face is located. A face region only contains some facial features. In the localization process we may go through some problems like false detection due to the presence of some obstacles on the face, poor quality of picture, orientation of head position, expression, etc. Hence we have to consider all these limitations to make the face detection process more powerful.

2.3.8.1.2 Normalization

After properly locating the region that contains a face, we must normalize the face region . Using normalization process, we align the face region in such a way that all the facial features are in their

proper location. Not only this, we may have to scale, rotate the image or some other transformation to correlate it with the entry in that database.

2.3.8.1 .3 Facial feature extraction

In this step of face detection, we extract various facial features like eyes, nose, mouth, etc, from the detected face region.

There are three types of feature extraction methods:

1. Generic methods based on edges, lines, and curves;
2. Feature template based methods that are used to detect various facial features like eyes, nose, and mouth;
3. Color segmentation based methods that use face color instead of the intensity values;

Appearance based methods that are able to manage changes in illumination conditions, shape, pose and reflectance and even to handle translation and partial occlusions.

2.3.8.1.4 Verification

In the verification process, we verify the relationships between various features with some database entry containing a huge number of faces. Now verification cannot be done using only database entry, but also we can use some rule based techniques that uses the correlations of various facial features as its parameter, or by using so template based methods where we use a specific template model and try to find out a face region that fits into this model.

2.3.8.2 Face detection techniques

Face detection techniques have been researched for years and much progress has been proposed in literature. Most of the face detection methods focus on detecting frontal faces with good lighting conditions. these methods can be categorized into four types: knowledge-based, feature invariant, template matching and appearance-based.

2.3.8.2.1 Knowledge-based methods

These rule-based methods encode human knowledge of what constitutes a typical face. Usually, the rules capture the relationships between facial features. These methods are designed mainly for face localization, which aims to determine the image position of a single face.

2.3.8.2.2 Feature invariant approaches

These algorithms aim to find structural features that exist even when the pose, viewpoint, or lighting conditions vary, and then use these to locate faces. To distinguish from the knowledge-based methods, the feature invariant approaches start at feature extraction process and face candidates finding, and later verify each candidate by spatial relations among these features, while the knowledge-based methods usually exploit information of the whole image and are sensitive to complicated backgrounds and other factors. Face detection based on color information, random labeled graph matching fall in this category.

2.3.8.2.3 Template matching methods

In this category, several standard patterns of a face are stored to describe the face as a whole or the facial feature separately. The correlations between

an input image and the stored pattern are computed for detection. These methods have been used for both face localization and detection. Deformable template matching falls in this category, where the template of faces is deformable according to some defined rules and constraints.

2.3.8.2.4 Appearance-based methods

In contrast to template matching, the models (or templates) are learned from a set of training images, which should capture the representative variability of facial appearance. These learned models are then used for detection.

Examples of such type of methods are view-based face detection, Haar features and the Adaboost algorithm.

Any of the methods can involve color segmentation, pattern matching, statistical analysis and complex transforms, where the common goal is classification with least amount of error. Bounds on the classification accuracy change from method to method yet the best techniques are found in areas where the models or rules for classification are dynamic and produced from machine learning processes.

In order to be successful a face detection algorithm must possess two key features, accuracy and speed. [10]

2.4 Computer vision

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory. Computer vision has also been

described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, object pose estimation, learning, indexing, motion estimation, and image restoration.

2.4.1 Related fields

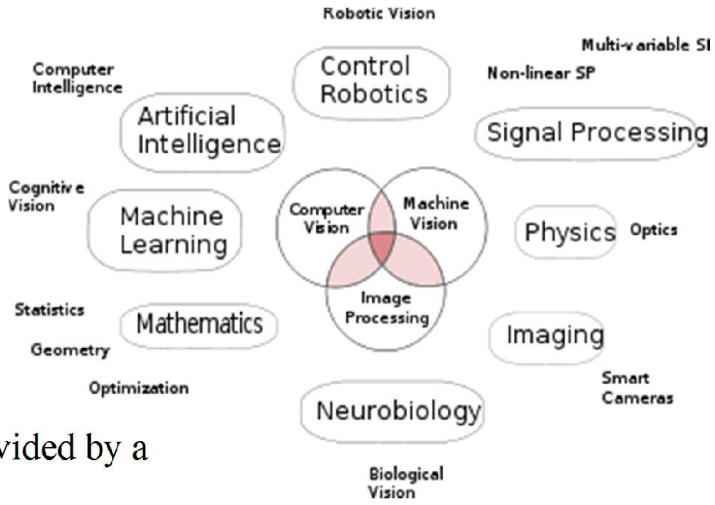


Figure 2.60 Relation between computer vision and various other field

Areas of artificial intelligence deal with autonomous planning or deliberation for robotical systems to navigate through an environment. A detailed understanding of these environments is required to navigate through them. Information about the environment could be provided by a computer vision system, acting as a vision sensor and providing high-level information about the environment and the robot.

Artificial intelligence and computer vision share other topics such as pattern recognition and learning techniques. Consequently, computer vision is sometimes seen as a part of the artificial intelligence field or the computer science field in general.

Solid-state physics is another field that is closely related to computer vision. Most computer vision systems rely on image sensors, which detect electromagnetic radiation which is typically in the form of either visible or infra-red light. The sensors are designed using quantum physics. The process by which light interacts with surfaces is explained using physics. Physics explains the behavior of optics which are a core part of most

imaging systems. Sophisticated image sensors even require quantum mechanics to provide a complete understanding of the image formation process. Also, various measurement problems in physics can be addressed using computer vision, for example motion in fluids.

A third field which plays an important role is neurobiology, specifically the study of the biological vision system. Over the last century, there has been an extensive study of eyes, neurons, and the brain structures devoted to processing of visual stimuli in both humans and various animals. This has led to a coarse, yet complicated, description of how "real" vision systems operate in order to solve certain vision related tasks. These results have led to a subfield within computer vision where artificial systems are designed to mimic the processing and behavior of biological systems, at different levels of complexity. Also, some of the learning-based methods developed within computer vision (*e.g.* neural net and deep learning based image and feature analysis and classification) have their background in biology.

Some strands of computer vision research are closely related to the study of biological vision – indeed, just as many strands of AI research are closely tied with research into human consciousness, and the use of stored knowledge to interpret, integrate and utilize visual information. The field of biological vision studies and models the physiological processes behind visual perception in humans and other animals. Computer vision, on the other hand, studies and describes the processes implemented in software and hardware behind artificial vision systems. Interdisciplinary exchange between biological and computer vision has proven fruitful for both fields.

Yet another field related to computer vision is signal processing. Many methods for processing of one-variable signals, typically temporal signals, can be extended in a natural way to processing of two-variable signals or multi-variable signals in computer vision. However, because of the specific nature of images there are many methods developed within computer vision which have no counterpart in processing of one-variable signals. Together with the multi-dimensionality of the signal, this defines a subfield in signal processing as a part of computer vision.

Beside the above-mentioned views on computer vision, many of the related research topics can also be studied from a purely mathematical point of view. For example, many methods in computer vision are based on statistics, optimization or geometry. Finally, a significant part of the field is devoted to the implementation aspect of computer vision; how existing methods can be realized in various combinations of software and hardware,

or how these methods can be modified in order to gain processing speed without losing too much performance.

The fields most closely related to computer vision are image processing, image analysis and machine vision. There is a significant overlap in the range of techniques and applications that these cover. This implies that the basic techniques that are used and developed in these fields are more or less identical, something which can be interpreted as there is only one field with different names. On the other hand, it appears to be necessary for research groups, scientific journals, conferences and companies to present or market themselves as belonging specifically to one of these fields and, hence, various characterizations which distinguish each of the fields from the others have been presented.

Computer vision is, in some ways, the inverse of computer graphics. While computer graphics produces image data from 3D models, computer vision often produces 3D models from image data. There is also a trend towards a combination of the two disciplines, *e.g.*, as explored in augmented reality.

The following characterizations appear relevant but should not be taken as universally accepted:

- Image processing and image analysis tend to focus on 2D images, how to transform one image to another, *e.g.*, by pixel-wise operations such as contrast enhancement, local operations such as edge extraction or noise removal, or geometrical transformations such as rotating the image. This characterization implies that image processing/analysis neither require assumptions nor produce interpretations about the image content.
- Computer vision includes 3D analysis from 2D images. This analyzes the 3D scene projected onto one or several images, *e.g.*, how to reconstruct structure or other information about the 3D scene from one or several images. Computer vision often relies on more or less complex assumptions about the scene depicted in an image.
- Machine vision is the process of applying a range of technologies & methods to provide imaging-based automatic inspection, process control and robot guidance in industrial applications. Machine vision tends to focus on applications, mainly in manufacturing, *e.g.*, vision based autonomous robots and systems for vision based inspection or measurement. This implies that image sensor technologies and control theory often are integrated with the processing of image data to control a robot and that real-time processing is emphasised by means of efficient implementations in hardware and software. It also

implies that the external conditions such as lighting can be and are often more controlled in machine vision than they are in general computer vision, which can enable the use of different algorithms.
- There is also a field called imaging which primarily focus on the process of producing images, but sometimes also deals with processing and analysis of images. For example, medical imaging includes substantial work on the analysis of image data in medical applications.
- Finally, pattern recognition is a field which uses various methods to extract information from signals in general, mainly based on statistical approaches and artificial neural networks. A significant part of this field is devoted to applying these methods to image data.

2.4.2 Applications for computer vision

Applications range from tasks such as industrial machine vision systems which, say, inspect bottles speeding by on a production line, to research into artificial intelligence and computers or robots that can comprehend the world around them. The computer vision and machine vision fields have significant overlap. Computer vision covers the core technology of automated image analysis which is used in many fields. Machine vision usually refers to a process of combining automated image analysis with other methods and technologies to provide automated inspection and robot guidance in industrial applications. In many computer vision applications, the computers are pre-programmed to solve a particular task, but methods based on learning are now becoming increasingly common. Examples of applications of computer vision include systems for:

- Controlling processes, *e.g.*, an industrial robot;
- Navigation, *e.g.*, by an autonomous vehicle or mobile robot;
- Detecting events, *e.g.*, for visual surveillance or people counting;
- Organizing information, *e.g.*, for indexing databases of images and image sequences;
- Modeling objects or environments, *e.g.*, medical image analysis or topographical modeling;
- Interaction, *e.g.*, as the input to a device for computer-human interaction, and
- Automatic inspection, *e.g.*, in manufacturing applications.

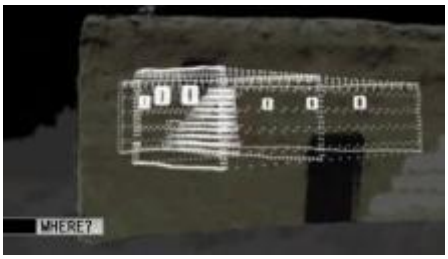


Figure 2.61 DARPA's Visual Media Reasoning concept video

One of the most prominent application fields is medical computer vision or medical image processing. This area is characterized by the extraction of information from image data for the purpose of making a medical diagnosis of a patient. Generally, image data is in the form of microscopy images, X-ray images, angiography images, ultrasonic images, and tomography images. An example of information which can be extracted from such image data is detection of tumours, arteriosclerosis or other malign changes. It can also be measurements of organ dimensions, blood flow, etc. This application area also supports medical research by providing new information, *e.g.*, about the structure of the brain, or about the quality of medical treatments. Applications of computer vision in the medical area also includes enhancement of images that are interpreted by humans, for example ultrasonic images or X-ray images, to reduce the influence of noise.

A second application area in computer vision is in industry, sometimes called machine vision, where information is extracted for the purpose of supporting a manufacturing process. One example is quality control where details or final products are being automatically inspected in order to find defects. Another example is measurement of position and orientation of details to be picked up by a robot arm. Machine vision is also heavily used in agricultural process to remove undesirable food stuff from bulk material, a process called optical sorting.

Military applications are probably one of the largest areas for computer vision. The obvious examples are detection of enemy soldiers or vehicles and missile guidance. More advanced systems for missile guidance send the missile to an area rather than a specific target, and target selection is made when the missile reaches the area based on locally acquired image data. Modern military concepts, such as "battlefield awareness", imply that various sensors, including image sensors, provide a rich set of information about a combat scene which can be used to support strategic decisions. In this case, automatic processing of the data is used to reduce complexity and to fuse information from multiple sensors to increase reliability.

Figure 2.62 Artist's Concept of Rover on Mars

One of the newer application areas is autonomous vehicles, which include submersibles, land-based vehicles (small robots with wheels, cars or trucks), aerial vehicles, and unmanned aerial vehicles (UAV). The level of autonomy ranges from fully autonomous (unmanned) vehicles to vehicles where computer vision based systems support a driver or a pilot in various situations. Fully autonomous vehicles typically use computer vision for navigation, i.e. for knowing where it is, or for producing a map of its environment (SLAM) and for detecting obstacles. It can also be used for detecting certain task specific events, *e.g.*, a UAV looking for forest fires. Examples of supporting systems are obstacle warning systems in cars, and systems for autonomous landing of aircraft. Several car manufacturers have demonstrated systems for autonomous driving of cars, but this technology has still not reached a level where it can be put on the market. There are ample examples of military autonomous vehicles ranging from advanced missiles, to UAVs for recon missions or missile guidance. Space exploration is already being made with autonomous vehicles using computer vision, *e.g.*, NASA's Mars Exploration Rover and ESA's ExoMars Rover.

Other application areas include:

- Support of visual effects creation for cinema and broadcast, *e.g.*, camera tracking (matchmoving).
- Surveillance.

2.4.3 Typical tasks of computer vision

Each of the application areas described above employ a range of computer vision tasks; more or less well-defined measurement problems or processing problems, which can be solved using a variety of methods. Some examples of typical computer vision tasks are presented below.

2.4.3.1 Recognition

The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity. Different varieties of the recognition problem are described in the literature:

- **Object recognition** (also called **object classification**) – one or several pre-specified or learned objects or object classes can be recognized, usually together with their 2D positions in the image or 3D poses in the scene. Google Goggles or LikeThat provide a stand-alone programs that illustrate this function.
- **Identification** – an individual instance of an object is recognized. Examples include identification of a specific person's face or fingerprint, identification of handwritten digits, or identification of a specific vehicle.
- **Detection** – the image data are scanned for a specific condition. Examples include detection of possible abnormal cells or tissues in medical images or detection of a vehicle in an automatic road toll system. Detection based on relatively simple and fast computations is sometimes used for finding smaller regions of interesting image data which can be further analyzed by more computationally demanding techniques to produce a correct interpretation.

Currently, the best algorithms for such tasks are based on convolutional neural networks. An illustration of their capabilities is given by the ImageNet Large Scale Visual Recognition Challenge; this is a benchmark in object classification and detection, with millions of images and hundreds of object classes. Performance of convolutional neural networks, on the ImageNet tests, is now close to that of humans. The best algorithms still struggle with objects that are small or thin, such as a small ant on a stem of a flower or a person holding a quill in their hand. They also have trouble with images that have been distorted with filters (an increasingly common phenomenon with modern digital cameras). By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues. For example, they are not good at classifying objects into fine-grained classes, such as the particular breed of dog or species of bird, whereas convolutional neural networks handle this with ease.

Several specialized tasks based on recognition exist, such as:

- **Content-based image retrieval** – finding all images in a larger set of images which have a specific content. The content can be

specified in different ways, for example in terms of similarity relative a target image (give me all images similar to image X), or in terms of high-level search criteria given as text input (give me all images which contains many houses, are taken during winter, and have no cars in them).

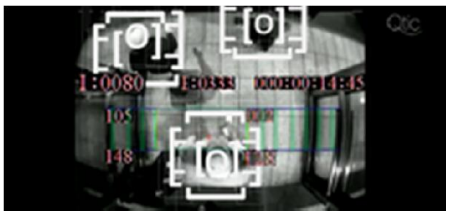


Figure 2.63 Computer vision for people counter purposes

- **Pose estimation** – estimating the position or orientation of a specific object relative to the camera. An example application for this technique would be assisting a robot arm in retrieving objects from a conveyor belt in an assembly line situation or picking parts from a bin.
- **Optical character recognition** (OCR) – identifying characters in images of printed or handwritten text, usually with a view to encoding the text in a format more amenable to editing or indexing (*e.g.* ASCII).
- **2D Code reading** Reading of 2D codes such as data matrix and QR codes.
- **Facial recognition**
- **Shape Recognition Technology** (SRT) in people counter systems differentiating human beings (head and shoulder patterns) from objects

2.4.3.2 Motion analysis

Several tasks relate to motion estimation where an image sequence is processed to produce an estimate of the velocity either at each points in the image or in the 3D scene, or even of the camera that produces the images . Examples of such tasks are:

- **Egomotion** – determining the 3D rigid motion (rotation and translation) of the camera from an image sequence produced by the camera.
- **Tracking** – following the movements of a (usually) smaller set of interest points or objects (*e.g.*, vehicles or humans) in the image sequence.
- **Optical flow** – to determine, for each point in the image, how that point is moving relative to the image plane, i.e., its apparent motion. This motion is a result both of how the corresponding 3D point is

moving in the scene and how the camera is moving relative to the scene.

2.4.3.3 Scene reconstruction

Given one or (typically) more images of a scene, or a video, scene reconstruction aims at computing a 3D model of the scene. In the simplest case the model can be a set of 3D points. More sophisticated methods produce a complete 3D surface model. The advent of 3D imaging not requiring motion or scanning, and related processing algorithms is enabling rapid advances in this field. Grid-based 3D sensing can be used to acquire 3D images from multiple angles. Algorithms are now available to stitch multiple 3D images together into point clouds and 3D models.

2.4.3.4 Image restoration

The aim of image restoration is the removal of noise (sensor noise, motion blur, etc.) from images. The simplest possible approach for noise removal is various types of filters such as low-pass filters or median filters. More sophisticated methods assume a model of how the local image structures look like, a model which distinguishes them from the noise. By first analysing the image data in terms of the local image structures, such as lines or edges, and then controlling the filtering based on local information from the analysis step, a better level of noise removal is usually obtained compared to the simpler approaches.

An example in this field is in painting.

2.4.4 Computer vision system methods

The organization of a computer vision system is highly application dependent. Some systems are stand-alone applications which solve a specific measurement or detection problem, while others constitute a sub-system of a larger design which, for example, also contains sub-systems for control of mechanical actuators, planning, information databases, man-machine interfaces, etc. The specific implementation of a computer vision system also depends on if its functionality is pre-specified or if some part of it can be learned or modified during operation. Many functions are unique to the application. There are, however, typical functions which are found in many computer vision systems.

- **Image acquisition** – A digital image is produced by one or several image sensors, which, besides various types of light-sensitive cameras, include range sensors, tomography devices, radar, ultra-sonic cameras, etc. Depending on the type of sensor, the resulting image data is an ordinary 2D image, a 3D volume, or an image sequence. The pixel values typically correspond to light intensity in one or several spectral bands (gray images or colour images), but can also be related to various physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves, or nuclear magnetic resonance.
- **Pre-processing** – Before a computer vision method can be applied to image data in order to extract some specific piece of information, it is usually necessary to process the data in order to assure that it satisfies certain assumptions implied by the method. Examples are
    - Re-sampling in order to assure that the image coordinate system is correct.
    - Noise reduction in order to assure that sensor noise does not introduce false information.
    - Contrast enhancement to assure that relevant information can be detected.
    - Scale space representation to enhance image structures at locally appropriate scales.
- **Feature extraction** – Image features at various levels of complexity are extracted from the image data. Typical examples of such features are
    - Lines, edges and ridges.
    - Localized interest points such as corners, blobs or points.

  More complex features may be related to texture, shape or motion.

- **Detection/segmentation** – At some point in the processing a decision is made about which image points or regions of the image are relevant for further processing. Examples are
    - Selection of a specific set of interest points
    - Segmentation of one or multiple image regions which contain a specific object of interest.
- **High-level processing** – At this step the input is typically a small set of data, for example a set of points or an image region which is assumed to contain a specific object. The remaining processing deals with, for example:
    - Verification that the data satisfy model-based and application specific assumptions.

  - Estimation of application specific parameters, such as object pose or object size.
  - Image recognition – classifying a detected object into different categories.
  - Image registration – comparing and combining two different views of the same object.
- **Decision making** Making the final decision required for the application, for example:
  - Pass/fail on automatic inspection applications
  - Match / no-match in recognition applications
  - Flag for further human review in medical, military, security and recognition applications

2.4.5 Computer vision hardware

There are many kinds of computer vision systems, nevertheless all of them contain these basic elements: a power source, at least one image acquisition device (i.e. camera, ccd, etc.), a processor as well as control and communication cables or some kind of wireless interconnection mechanism. In addition, a practical vision system contains software, as well as a display in order to monitor the system. Vision systems for inner spaces, as most industrial ones, contain an illumination system and may be placed in a controlled environment. Furthermore, a completed system includes many accessories like camera supports, cables and connectors.

2.5 Speech Recognition

Speech recognition is the process of converting spoken words into data. The process is used in many sectors, including business or households. It is helpful for people with physical disabilities, such as loss of vision or loss of use of hands. Speech recognition is also used by the healthcare systems, military, and telephony.

Speech recognition allows a computer to recognize words and follow basic vocal instructions by distinguishing phonemes (distinct sounds) and morphemes, the smallest units of linguistic meaning in a language. Speech recognition systems are useful when individuals are unable to use a keyboard because their hands are occupied or disabled. Instead of typing commands or using a mouse to select menu items, an individual can speak into a computer's microphone.

Speech recognition systems usually require a training session during which the computer learns a particular voice and accent. Some systems also require the speaker to talk slowly and distinctly, and to separate words with short pauses. When compared to the human brain, today's computer brains are not skilled at speech recognition. As Alan Phelps wrote in the July 2000 issue of Smart Computing magazine, "Although babies all over the world pick up language abilities at an astonishing rate, computers muddle along with their super-duper electronic brains, incapable of understanding even rudimentary speech. Even your lowly mutt who sleeps on a pile of old blankets in the garage can make more sense of your voice."

The first step in speech recognition is converting a speaker's voice into something a computer can recognize. This is accomplished with sampling, as described in the "Digitizing Speech" section. In addition to sampling speech, the computer also listens for pauses so it can sample background noises and remove them from the data. Speech recognition programs compare the sampled sound to known characteristics of human speech and remove obvious noise, such as automobiles and ringing phones.

After sampling, the next step for voice recognition software is to find phonemes within the string of incoming values. A computer can usually recognize phonemes, but not always, depending on the speaker's tone, accent, and rate of speaking. If a phoneme is not obvious, the software makes educated guesses based on linguistic research into which phonemes typically follow others. These conjectures are aided by the fact that the software already learned how the current user speaks.

Once the software understands phonemes, it begins combining the phonemes into morphemes and words. Once again, statistical analysis helps complete the job. For example, perhaps you want to compliment the computer and say, "This software is great." Maybe you are unsure of this statement, so you don't articulate "great." The software thinks it heard, "This software is" and then something with the /e/ phoneme. By combining the recognized sounds and word-probability scores, the software can make a good guess that you said, "This software is great," and not "This software is ate" or "This software is grape."

Computers can be programmed with a knowledge base that models a human expert and is capable of distinguishing useful utterances from

similar, but not useful, utterances. For example, expert systems for the medical industry let doctors ask questions that the computer answers. If the doctors ask open-ended questions, however, the computer also needs natural language processing capabilities, which is a harder problem.

As computers become faster, software can analyze numerous probabilities so quickly that the user does not notice a slow response time as the speech recognition software moves from phonemes to morphemes to words. Being able to find the probable next phoneme or word is not the same as language comprehension, however. Human language is open-ended. It does not consist of a finite database of fixed words and sentences. Instead, human speakers learn a grammar that lets them construct novel sentences and assign meaning to original utterances by others. Computer engineers face a major hurdle when attempting to emulate these sophisticated comprehension abilities.

There are many applications for speech recognition, for example, telephone systems that let you say numbers rather than enter them with your fingers. User-interface software from Apple Computer, Microsoft and other companies can turn voice commands into the mouse clicks that programs expect. Many new wireless phones let drivers dial numbers by voice so they can keep their hands on the steering wheel. Technologies such as Microsoft's Auto PC also let you ask for directions, control a CD player, and read your e-mail aloud, all while driving your car. One of the more fun uses for voice recognition is surfing the Web with your voice. Dragon System's NaturallySpeaking and IBM's ViaVoice products offer the ability to surf the Web without touching the keyboard.

Security systems that recognize words and identify individual voiceprints are used to guard access to ATMs, buildings, computers, voice mail, and wireless phones. Voice Track Corporation's voice recognition products allow corrections officers to track the whereabouts of parolees. Using voice recognition for identification and authorization has some limitations, however. Most systems cannot account for a user with laryngitis, for example. Also, a good recording can fool some systems. Using speech recognition for security should be combined with other biometrics, such as finger or retina scanning.

2.5.1 Digitizing Speech

Computers work with discrete digital values. For example, a digital watch is called digital because it can display a finite number of times. In contrast, watches with hands are analog because the hands move continuously around the clock face. As the minute hand travels around the circle, it touches the numbers 1 through 12 and also the infinite number of points in between.

The human voice produces an analog signal.

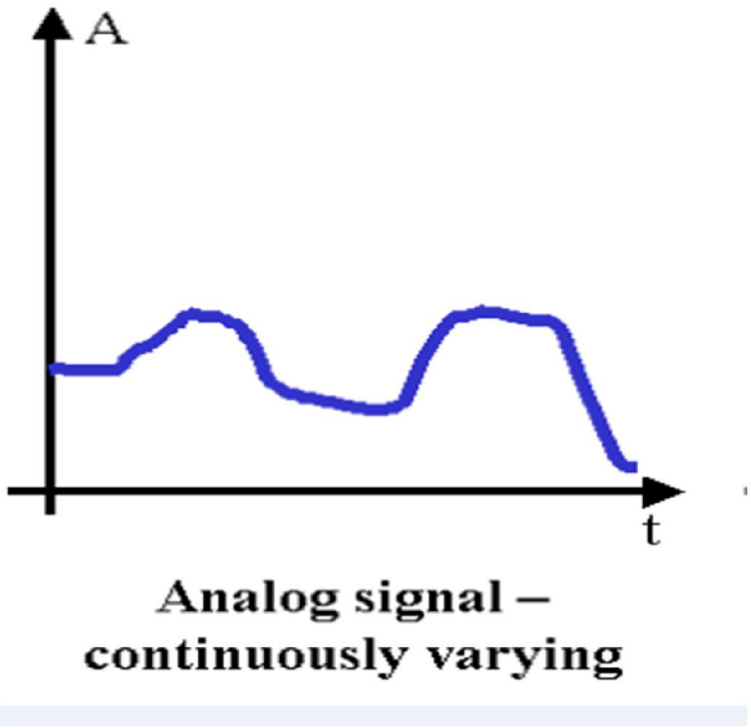


Figure 2.64 analog signal of human voice

When a speaker pushes air out the lungs through the glottis, air pulses escape out the mouth and sometimes the nose. These pulses produce small variations in air pressure that result in an analog signal.

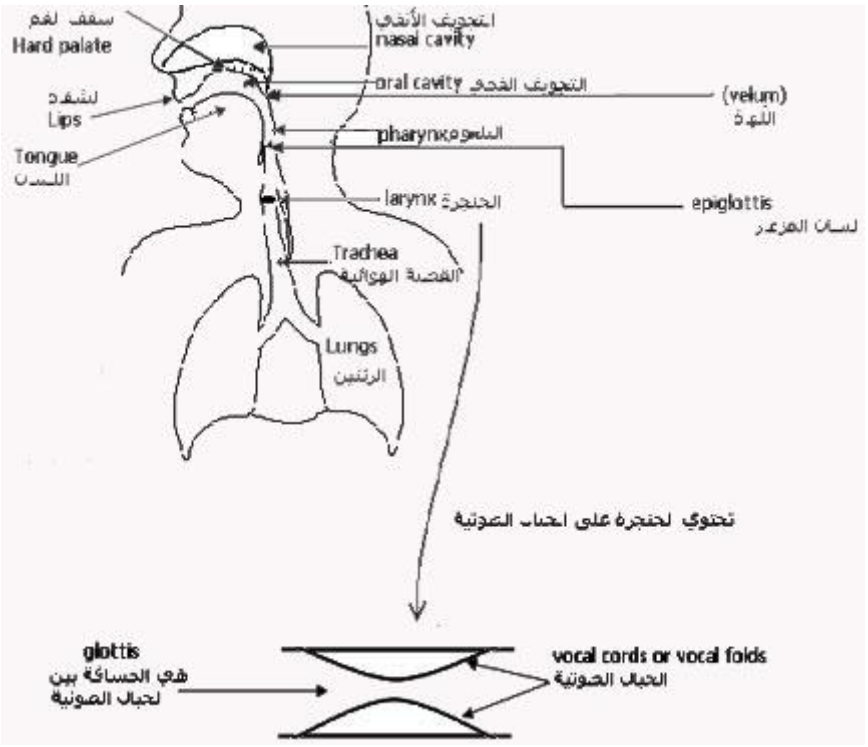


Figure 2.65 The human pronounce system

According to Fromkin and Rodman, "The sounds we produce can be described in terms of how fast the variations of the air pressure occur, which determines the fundamental frequency of the sounds and is perceived by the hearer as pitch. We can also describe the magnitude or intensity of the variations, which determines the loudness of the sounds."

Human speech can be represented as an analog wave that varies over time and has a smooth, continuous curve. The height of the wave represents intensity (loudness), and the shape of the wave represents frequency (pitch).

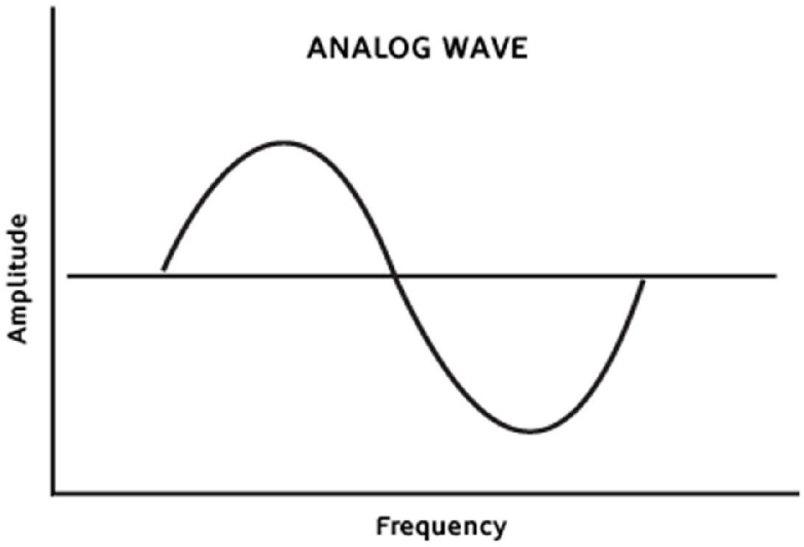


Figure 2.66 Human speech as an analog wave varies over time

The continuous curve of the wave accommodates an infinity of possible values. A computer must convert these values into a set of discrete values, using a process called digitization.

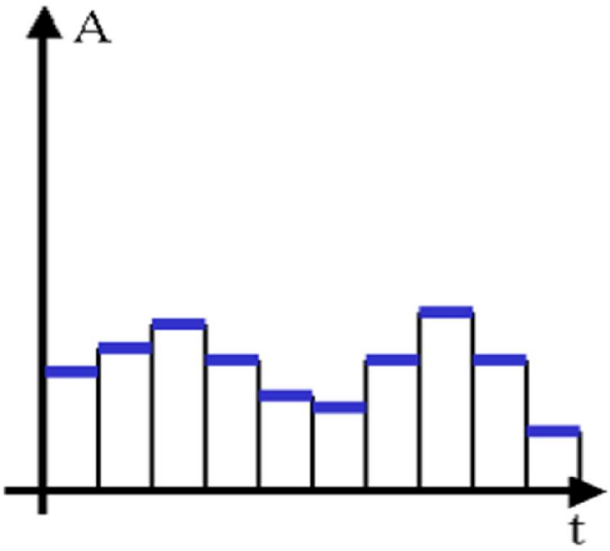


Figure 2.67 digital signal-large time divisions

Once speech is digitized, a computer can store speech on a hard drive and transmit speech across digital networks, including corporate networks, the Internet, and telephone-company networks, which are increasingly using digital components.

To digitize speech, an analog-digital converter samples the value of the analog signal repeatedly and encodes each result in a set of bits. Before sampling, the converter filters the signal so that most of it lies between 300 and 3400 Hz. While humans can hear frequencies as high as 20 kHz, most of the information conveyed in speech does not exceed 4 kHz. (A Hertz, or Hz, is a unit of frequency equal to one cycle per second.)

Sampling uses a theorem developed by the American physicist Harry Nyquist in the 1920s. Nyquist's Theorem states that the sampling frequency must be at least twice as high as the highest input frequency for the result to closely resemble the original signal. Thus, the voice signal is sampled at 8000 Hz so that frequencies up to 4000 Hz can be recorded. Every 125 microseconds ($1/8000^{th}$ of a second), the value of the analog voice signal is recorded in an 8-bit byte, the basic unit of storage on modern-day computers. By sampling this often, according to Nyquist, the result will be a faithful representation of the original signal, and the human ear will not hear distortion.

2.5.2 History of Speech Recognition

The last two decades have seen an amazing growth in development of speech recognition. However, the process of speech recognition began as early as the 1870's when Alexander Graham Bell discovered how to convert sound or air pressure waves into electrical impulses. He wanted to create a device to change words into pictures that can be understood by a deaf person so that he could communicate with his wife. It eventually led to the invention of the telephone. This triggered a process of mathematical and scientific method of understanding speech. The earliest device to recognize speech was developed in 1952 by Bell laboratories, which identified single spoken digits. In 1964, IBM's shoebox was displayed at the New York World Fair.

In the 1970's, the technology was developed further by ARPA Speech Understanding Research. The most notable discovery was that speech

should be understood, and not merely identified. In 1980, the discovery of two products made speech recognition accessible to the common man. The first one was helpful for processing transactions over the telephone as it identified small vocabulary. The other software which was offered by IBM, Dragon Systems and Kurzweil Applied Intelligence was for large vocabulary where text documents can be created by voice dictation.

2.5.3 Speech Recognition Issues

There are many factors that interfere with accurate speech recognition. Noise from the background is one such factor. If there is too much noise in the background, words cannot be heard or understood properly. This can be defined as low signal to noise ratio. Users should work in a silent place and speak as close to the microphone as possible. In some cases, the sound cards used may not be able to block signals from other computer parts, causing hissing noises. When there is overlapping speech as in a meeting or conference with many people, then voice recognition gets difficult. The processor needs to do a lot of work in order to comprehend complicated words, phrases or long winded sentences. Speech recognition vocabulary takes up a lot of hard disk space. Another difficulty is words that sound similar but spelt differently or words that sound similar but with different meanings. Extensive training has helped reduce that issue. Some other factors affecting speech recognition are the sex of the speaker, style of speaking, speed, dialects, and word boundary ambiguity (many ways of grouping words).[1]

2.5.4 Possible Uses for Speech Recognition

Speech Recognition technology has huge potential for human computer interaction. Over the last ten years, recognition technology has begun to appear in a multitude of devices from home computers to mobile phones and home security systems. It is the simplest way for a user to interact since speech is the most common and natural form of communication. It does not require complicated user interfaces.

2.5.5 Applications

Therapeutic use

Prolonged use of speech recognition software in conjunction with word processors has shown benefits to short-term-memory restrengthening in brain AVM patients who have been treated with resection. Further research needs to be conducted to determine cognitive benefits for individuals whose AVMs have been treated using radiologic techniques.

Usage in education and daily life

For language learning, speech recognition can be useful for learning a second language. It can teach proper pronunciation, in addition to helping a person develop fluency with their speaking skills.

Students who are blind (see Blindness and education) or have very low vision can benefit from using the technology to convey words and then hear the computer recite them, as well as use a computer by commanding with their voice, instead of having to look at the screen and keyboard.[61]

Students who are physically disabled or suffer from Repetitive strain injury/other injuries to the upper extremities can be relieved from having to worry about handwriting, typing, or working with scribe on school assignments by using speech-to-text programs. They can also utilize speech recognition technology to freely enjoy searching the Internet or using a computer at home without having to physically operate a mouse and keyboard.

Speech recognition can allow students with learning disabilities to become better writers. By saying the words aloud, they can increase the fluidity of their writing, and be alleviated of concerns regarding spelling, punctuation, and other mechanics of writing. Also, see Learning disability.

Voice Recognition Software's use, in conjunction with a digital audio recorder, a personal computer and Microsoft Word has proven to be positive for restoring damaged short-term-memory capacity, in stroke and craniotomy individuals.

People with disabilities

People with disabilities can benefit from speech recognition programs. For individuals that are Deaf or Hard of Hearing, speech recognition software is used to automatically generate a closed-captioning of conversations such

as discussions in conference rooms, classroom lectures, and/or religious services.

Speech recognition is also very useful for people who have difficulty using their hands, ranging from mild repetitive stress injuries to involved disabilities that preclude using conventional computer input devices. In fact, people who used the keyboard a lot and developed RSI became an urgent early market for speech recognition. Speech recognition is used in deaf telephony, such as voicemail to text, relay services, and captioned telephone. Individuals with learning disabilities who have problems with thought-to-paper communication (essentially they think of an idea but it is processed incorrectly causing it to end up differently on paper) can possibly benefit from the software but the technology is not bug proof. Also the whole idea of speak to text can be hard for intellectually disabled person's due to the fact that it is rare that anyone tries to learn the technology to teach the person with the disability.

This type of technology can help those with dyslexia but other disabilities are still in question. The effectiveness of the product is the problem that is hindering it being effective. Although a kid may be able to say a word depending on how clear they say it the technology may think they are saying another word and input the wrong one. Giving them more work to fix, causing them to have to take more time with fixing the wrong word .[13]

2.5.6 Advantages / Disadvantages of speech recognition system

Advantages

First benefits of this strategy is that degradation of the possibility of copying security passwords because there is no need of composing security passwords and the whole can be done without any worry.
The significant advantage of this program is in contact centers where a huge number of clients are on line toenquire and a representative needs to be online to be present at the call. With the help of this technological innovation calling can be joined successfully and with more efficiency.
Person who is unable to write or see with the help of this application can perform their task such as inquiring or transaction process etc.

Country like India has so many dialects variation with the help of this technology the dependency of human staff trained in different languages has been reduced significantly.
This application has proved a revolution to improve customer happiness in addition to improving companies' earnings by simply achieving new customers in addition to holding onto existing customers.
This is very beneficial for who are able to read and write up to some extent and know how to make use of cell but can't write or speak in English to type English letters as passwords.

Disadvantages

- Even the best speech recognition systems sometimes make errors. If there is noise or some other sound in the room (e.g. the television or a kettle boiling), the number of errors will increase.
- Speech Recognition works best if the microphone is close to the user (e.g. in a phone, or if the user is wearing a microphone). More distant microphones (e.g. on a table or wall) will tend to increase the number of errors.

2.6 Related work

2.6.1 Camera Mouse

Camera Mouse is a program that allows you to control the mouse pointer on a Windows
computer just by moving your head.
Camera Mouse uses a standard built-in camera or USB webcam to track your head. If you move
your head to the left, the mouse pointer moves to the left, and so on.
Clicking can be done by
"dwell time". If you hold the mouse pointer within a certain area of the screen for, say, a second

a mouse click will be issued by the program.

2.6.1.1 Hardware and software requirements

To run Camera Mouse 2015 you will need a computer or tablet with Windows 8 or Windows 7
or Windows Vista or Windows XP. Camera Mouse does not run on Windows RT.
Your computer also needs Microsoft .NET Framework 4.0 or higher.
You also will need a built-in camera or a standard USB webcam. We use a Microsoft LifeCam
Studio HD or a Logitech HD Pro Webcam C920 for development. Just about any good quality
Webcam should do, including most built-in webcams. Inexpensive webcams, including some built in to notebook computers or tablets, can produce noisy or muddy images, which Camera mouse can have a more difficult time tracking.

2.6.1.2 Disadvantages

1) He suppose that he will help disabled people **BUT** When ever you open the computer you need to open the camera mouse **by your hand** and adjust every usage settings and choose sub image you want to track it so need external people to help disabled people

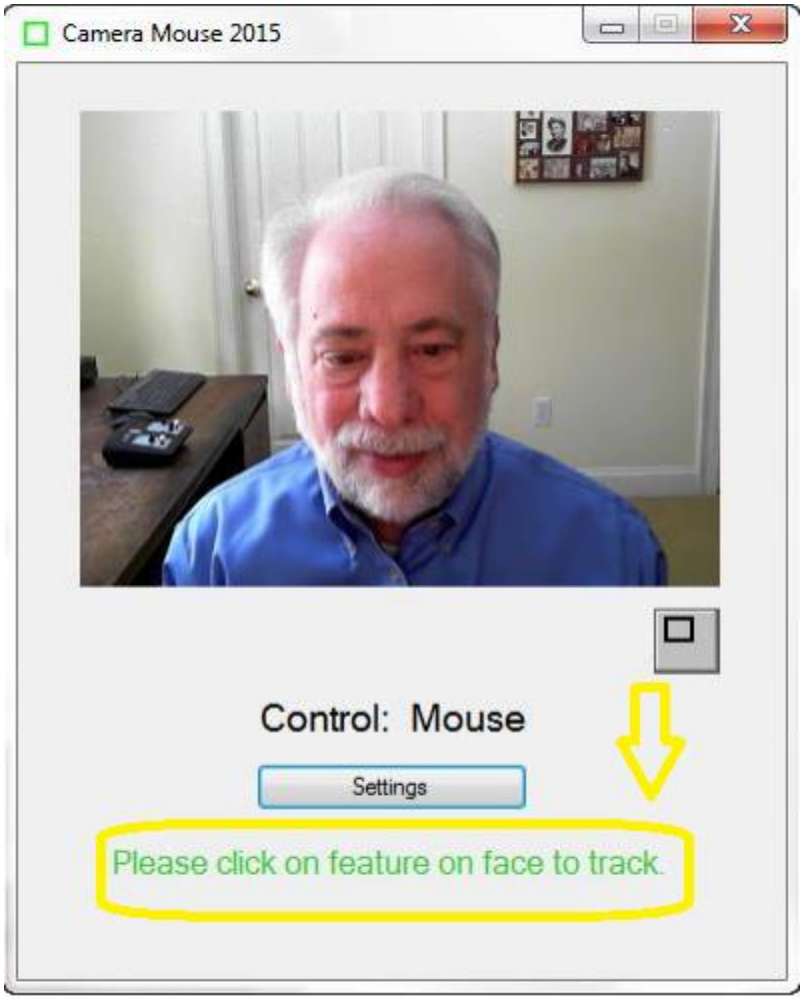


Figure 2.68 camera mouse control dis addvantage -1-

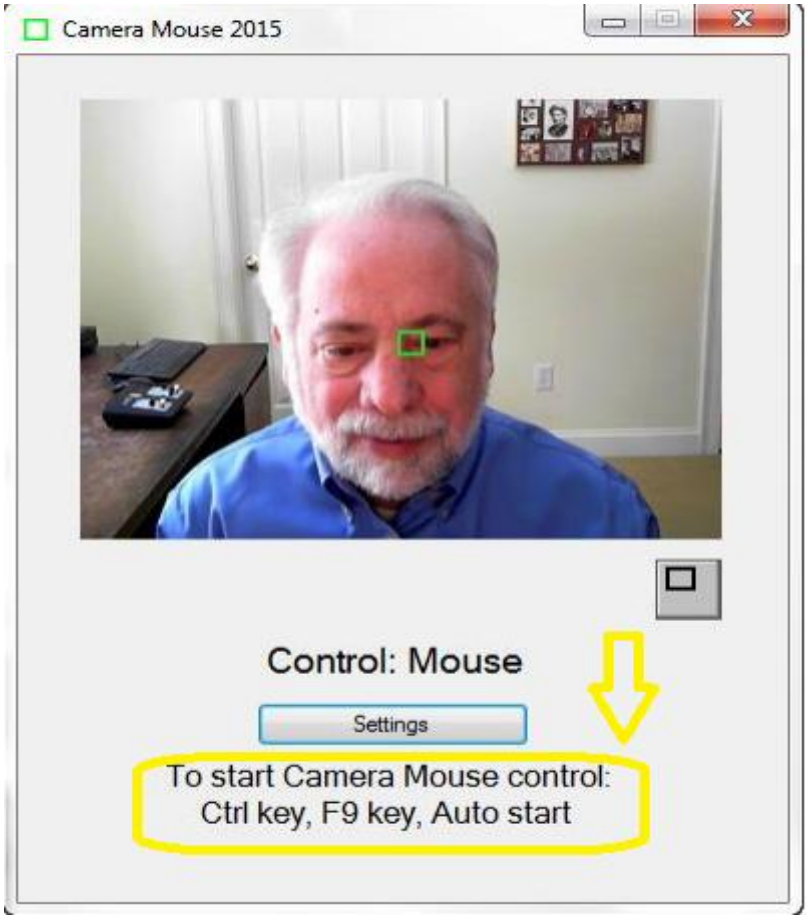


Figure 2.69 camera mouse control dis advantage -2-

And so on ,,,,,,

2) And only can make left click but you can't make right click by camera mouse

3) moving accuracy not high

But our system will run as a service and it will start with a default option from three options that come below.

It don't need any external help from other people so it will be very helpful for disabled people .

2.6.2 Tobii EyeX Dev Kit

The Tobii PCEye Go is a peripheral eye tracker that enhances computer accessibility with the speed, power and accuracy of gaze interaction. The device replaces the standard mouse, allowing you to navigate and control a desktop or laptop computer using only your eyes.

Figure 2.70 tobii

2.6.2.1 Disadvantages

1) use H/W (this have additional cost)

2) Only can make left clicks

3) have rays that may have bad effect on human

4) User should open his eye as large as he can (this may confuse user)

2.7 Virtual mouse control (VMC)

Virtual Mouse Controller (VMC) is a software application that help arm Disabilities (such as Parkinson's disease, Arm amputations or they born With Congenital disorder in their arms) to control mouse operation using Face not physical mouse in suitable manner,

VMC is alternative source for physical mouse (traditional mouse)

User can make click by one of three option

4) first option by speech recognition (user say left for left click and right for right click)
5) second option by time limit (stay in same position for one second means left click and for three seconds means double click)
6) third option by eye blink (user blink left eye means left click and blink right eye means right click)
   VMC use only webcam of laptop or any USB webcam(this to reduce cost as possible as we can)

   Objectives of our Project VMC:
   4) help disabled people to control computer by themselves
   5) spread virtual mouse technology
   6) reduce cost (almost no additional cost)

Chapter 3

Analysis and Design

3.1 Overview

3.2 System Requirements

3.3 use case diagram

3.4 System Sequence Diagram (SSD)

3.5 data flow diagram (DFD)

3.6 flow chart diagram

3.1 Overview

This chapter present the analysis phase of the system.In this chapter we will discuss the system functional requirments and non-functional requirmnets.

3.2 System Requirements

3.2.1 Functional Requirements

The functional requirements of the system, are categorized into two categories to different user's perspectives which are the following :

- **System User :-**

1) help disabled people to control computer by themselves

1.1) by moving the mouse pointer by his head

1.2) user can make click by one of three option

1.2.1) by speech recognition (user say left for left click and right for right click)

1.2.2) by time limit (stay in same position for one second means left click and for three seconds means double click)

1.2.3) By eye blink (user blink left eye means left click and blink right eye means right click.

2) Spread virtual mouse technology

3) Reduce cost (almost no additional cost)

3.2.2 Non-Functional Requirements

1. **Accessibility :** the system should be easy to access.
   - The application will be desktop-based and can be accessed from any device which the system will be installed on it.
2. **Availability :** the system should be in a specified operable state at the start of the mission .
3. **Accuracy :** The system should provide accurate results
4. **Usability :** The system should be easy to use .
   - User friendly interface .

3.3 use case diagrem

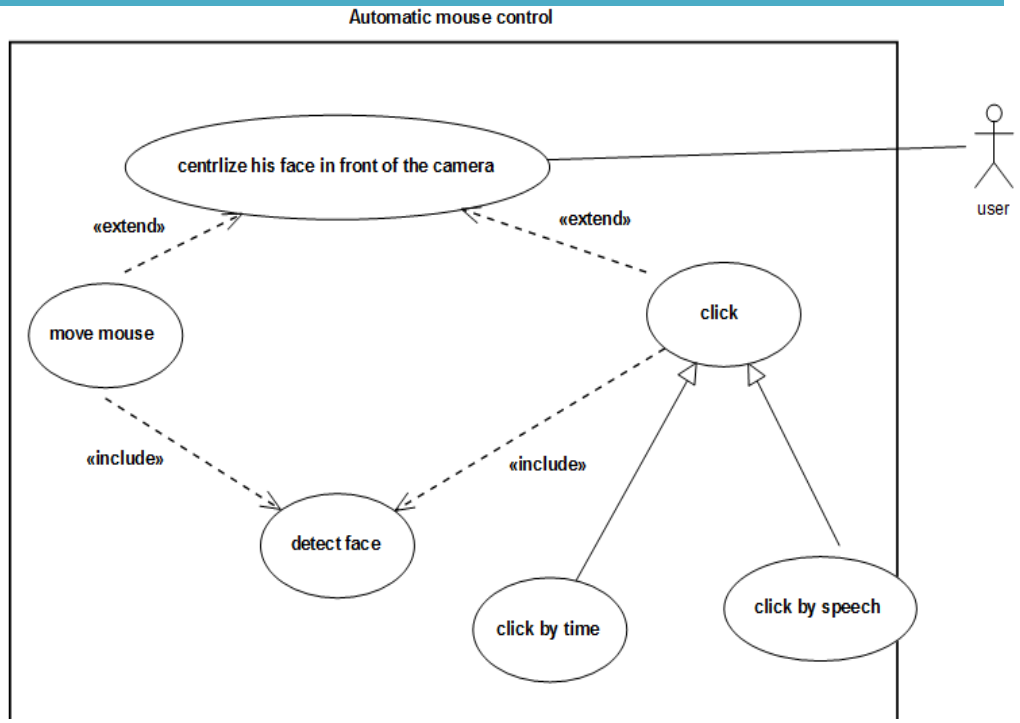


Figure 3.1 Use case diagrams

3.3.1 "move mouse pointer " use case analysis

Use case name :	Move mouse pointer	
Actor(s):	System user	
Description :	This use case describes the process of moving mouse pointer depends on moving of user face	
Typical Course of Events:	Actor(s)	System
	Step1: This use case is initiated when user decide to moving mouse pointer by his head **Step3:** the user centralize his face **Step7:** This use case concludes when the user see the mouse pointer move according his face movement	**Step2:** the system ask user to centralize his face **Step4:** the system take captured images from the camera and make image processing operations on it **Step5:**the system detect the face from the image **Step 6:**the system move the mouse pointer according to user's face movement
Alternate Courses:	Step5:if the system doesn't detect the user face it will remember initial coordinate	
Precondition:	the system check camera availability and open it	

Post condition:	User move the mouse pointer successfully and travel to using mouse functions with the help of the system
Assumption:	None at this time.

Table 3.1 “move mouse pointer “use case scenario

3.3.2 "click by time " use case analysis

Use case name :	Click by time	
Actor(s):	System user	
Description :	This use case describes the process of clicking by time	
Typical Course of Events:	Actor(s)	System
	Step1: This use case is initiated when user decide to click single or double left click instead of mouse left button	**Step2:** the system waits the mouse pointer to still in an area for a second to make one left click **Step3:** the system waits the mouse pointer to still in an area for a three seconds to make double left click
	Step7: This use case concludes when the user see the mouse pointer clicks according to the time passed	

Alternate Courses:	Step2:if the mouse pointer move before the time passed equal one second the system won't do any clicks
Precondition:	The user move the mouse pointer to the icon that he want
Post condition:	Click by time Successfully completed
Assumption:	None at this time.

Table 3.2 "click by time "use case scenario

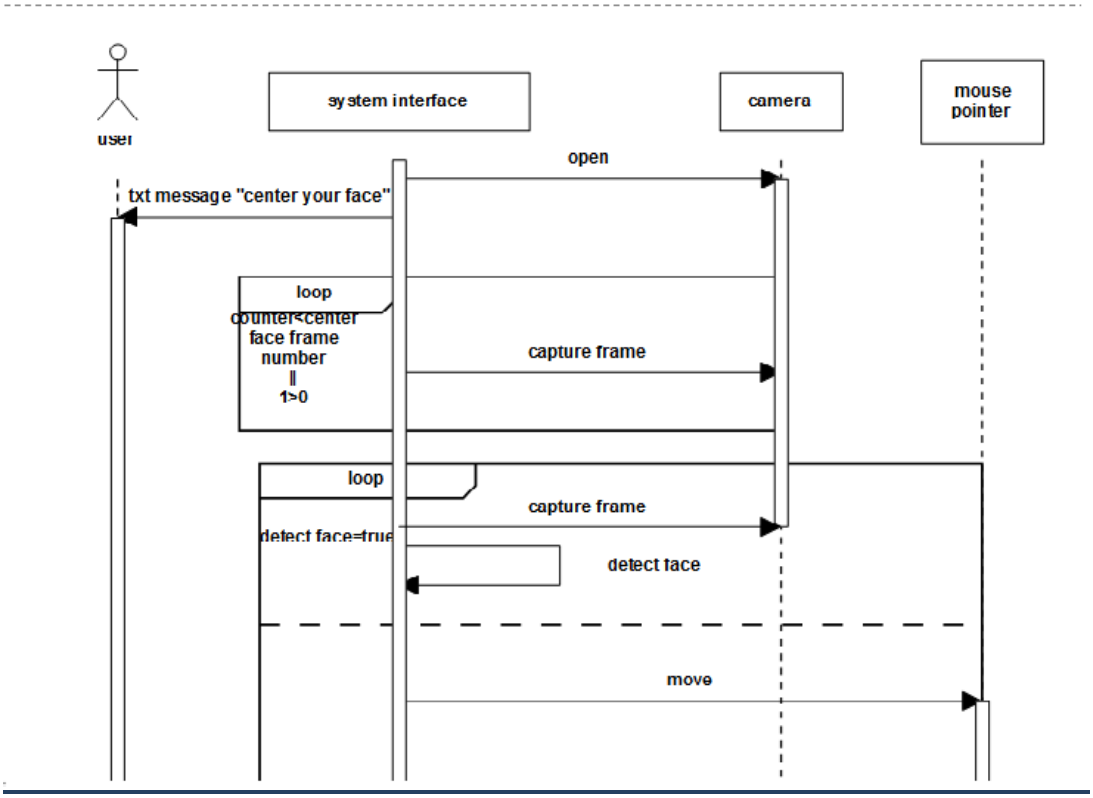
3.3.3 "click by speech commands" use case analysis

Use case name :	Click by speech	
Actor(s):	System user	
Description :	This use case describes the process of clicking by speech recognition	
Typical Course of Events:	Actor(s)	System
	Step1: This use case is initiated when user decide to use mouse functions by his voice **Step5:** This use case concludes when the user see the mouse pointer clicks	**Step2:** the system waits the user to say "left" to do left click **Step3:** the system waits the user to say "right" to do right click **Step4:** the system waits the user to say "control" to do down control button

	according to his Voice commands	
Alternate Courses:	Step2, Step3, Step4,if the user don't say anything the system won't do any clicks	
Precondition:	The user move the mouse pointer to the icon that he want	
Post condition:	Click by speech commands Successfully completed	
Assumption:	None at this time.	

Table 3.2 "click by speech commands " use case scenario

3.4 System Sequence Diagram (SSD)


user
system interface
camera
mouse pointer
open
txt message "center your face"
loop
counter<center face frame number
||
1>0
capture frame
loop
capture frame
detect face=true
detect face
move

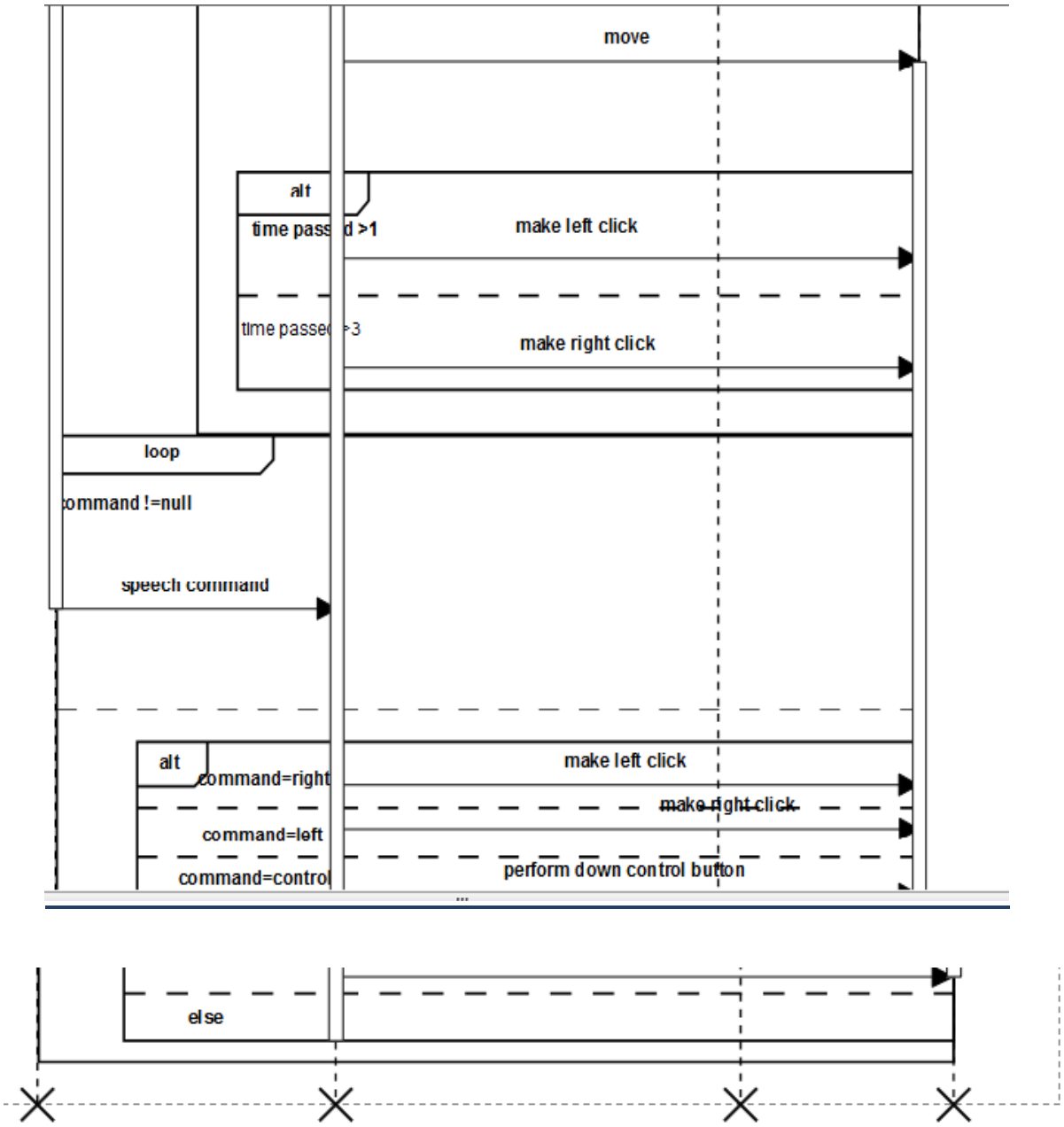


Figure 3.2 system sequence

3.5 data flow diagram (DFD)

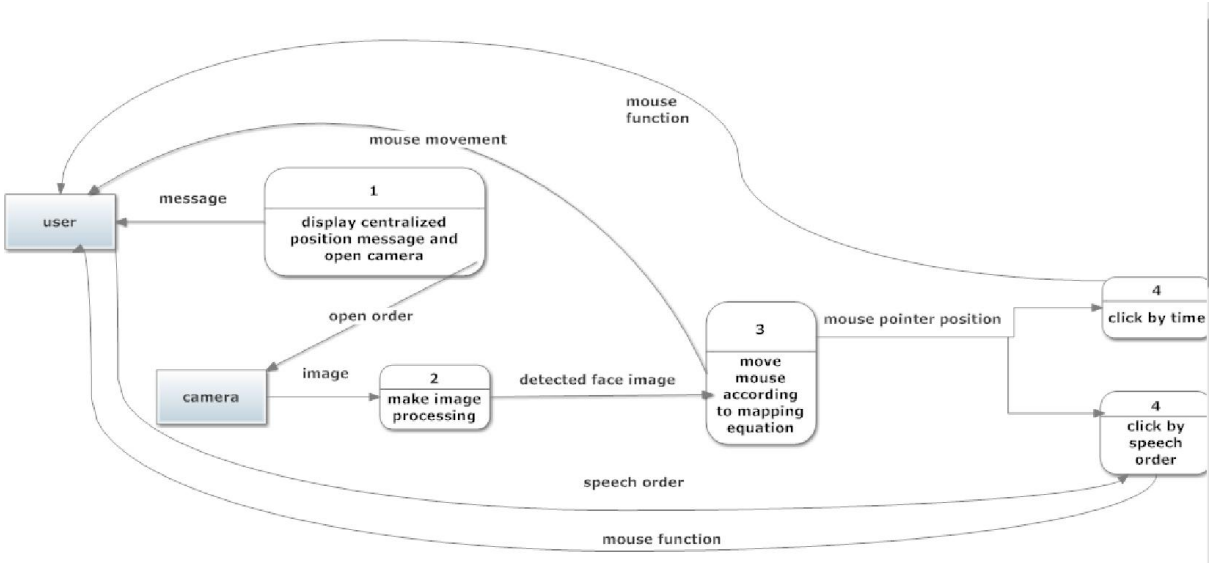
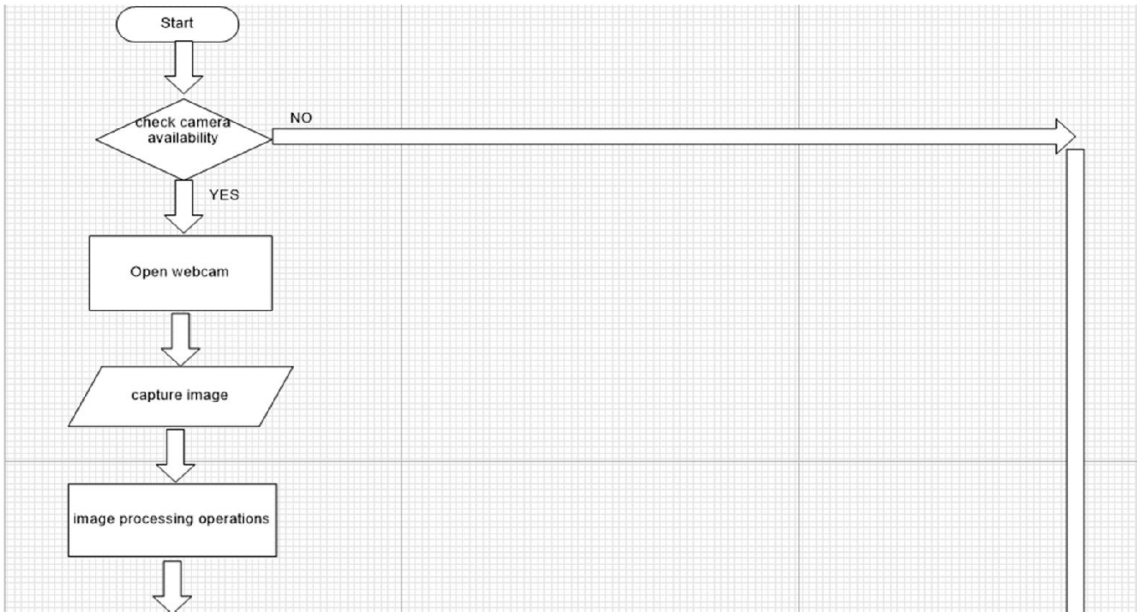
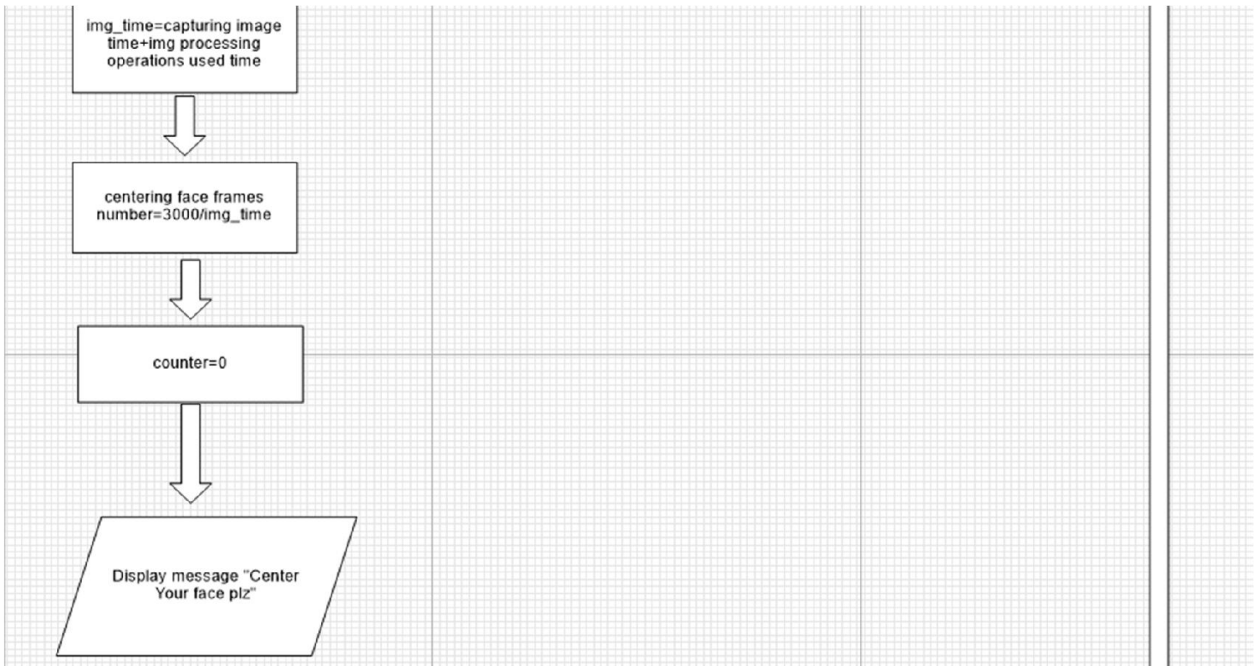
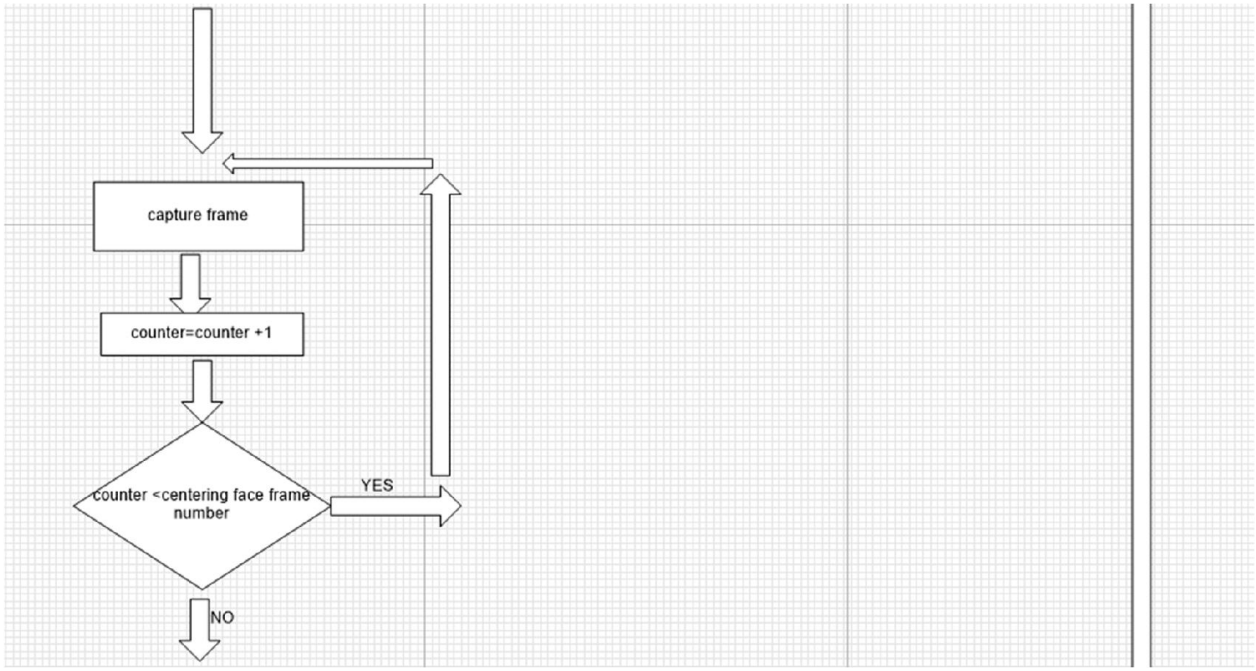


Figure 3.3 data flow diagram

3.6 flow chart diagram




img_time=capturing image time+img processing operations used time
centering face frames number=3000/img_time
counter=0
Display message "Center Your face plz"


capture frame
counter=counter +1
counter <centering face frame number
YES
NO

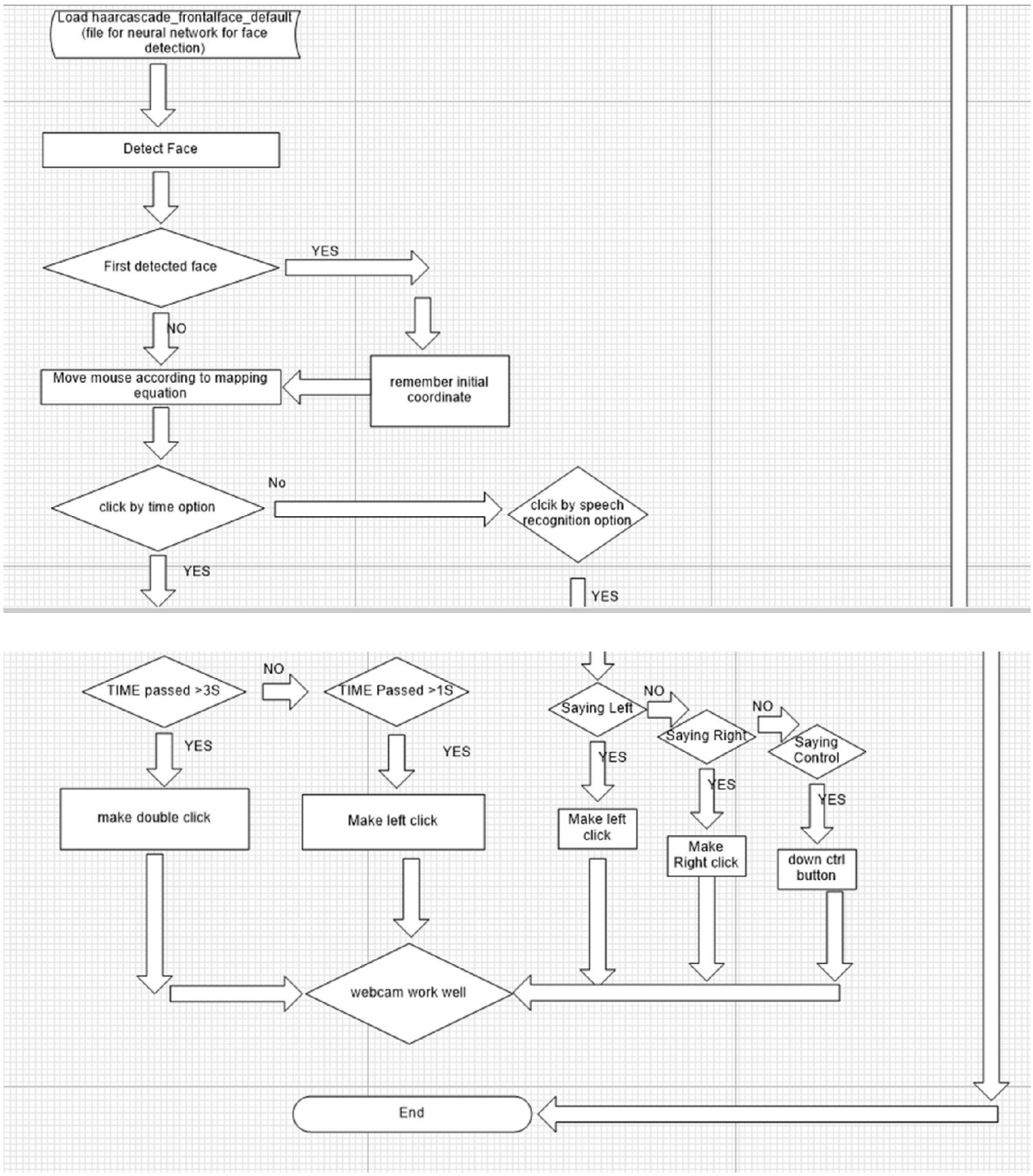


Figure 3.4 flow chart diagram

Chapter 4

Implementation

4.1 Overview

4.2 OpenCV

4.3 check webcam availability implementation

4.4 Digital image processing

4.5 Face detection

4.6 mouse location calculation in VMC

4.7 speech recognition

4.1Overview

The implementation phase comes after analysis and design phase in waterfall design model. So, this chapter focuses on the implementation phase. The implementation is a realization of technical specification or algorithm as a program, software component, or other computer system through programming and deployment. This chapter provides more technical details about developing previously mentioned technical issues in chapter 2 .System implementation is the software process in which actual coding takes place. A software program is written based upon the algorithm designed in the system design phase. A piece of code is written for every module and checked for the output. The purpose of the implementation phase is to release a fully tested and operational product to an end user or customer and to deploy and enable operations of the new information system in the production environment. The product should meet all the requirements that were documented and pass the testing phase before it can be released to a production environment. It focuses primarily on implementing the previously defined modules in to units of code. These units are developed independently are intergraded as the system is put together as part of a whole system. This is the phase where everything comes together and the intended audience receives the project or instruction. Planning for implementation is important. After all the analysis, design and development the team has done so far it would be terrible if the instruction or project fails due to poor implementation.

4.2 Opencv

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel Russia research center in Nizhny Novgorod, and now supported by Willow Garage and Itseez.[1] It is free for use under theopen-source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel'sIntegrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself. [12]

4.2.1 openCV history

OpenCV was started at Intel in 1999 by Gary Bradski for the purposes of accelerating research in and commercial applications of computer vision in the world and, for Intel, creating a demand for ever more powerful computers by such applications. Vadim Pisarevsky joined Gary to manage Intel's Russian software OpenCV team. Over time the OpenCV team moved on to other companies and other Research. Several of the original team eventually ended up working in robotics and found their way to Willow Garage. In 2008, Willow Garage saw the need to rapidly advance robotic perception capabilities in an open way that leverages the entire research and commercial

community and began actively supporting OpenCV, with Gary and Vadim once again leading the effort.opencv History.[16]

4.2.2 Load openCV dll files in java

```
private static void loadLibrary() {
                    String path=get_path();
    Tr {
     InputStream in = null;
     File fileOut = null;
     String osName = System.getProperty("os.name");
     //Utils.out.println(Trail.class, osName);
     if(osName.startsWith("Windows")){
       int bitness =
Integer.parseInt(System.getProperty("sun.arch.data.model"));
        if(bitness == 32)
              {
                 //   Utils.out.println(Trail.class, "32 bit detected");
               in=new FileInputStream(path+"x86\\opencv_java2410.dll");
              //in=new FileInputStream(inn.toString());
               fileOut = File.createTempFile("lib", ".dll");}

      else if (bitness == 64){
          //  Utils.out.println(Main.class, "64 bit detected");
```

```
            in=new FileInputStream(path+"x64\\opencv_java2410.dll");
           // Trail.class.getResource(osName)
            fileOut = File.createTempFile("lib", ".dll");
          }
          else{
         //  Utils.out.println(Main.class, "Unknown bit detected - trying with
32 bit");
           in=new FileInputStream(path+"x86\\opencv_java2410.dll");
             fileOut = File.createTempFile("lib", ".dll");
          }
       }
       else if(osName.equals("Mac OS X")){
         in=new
FileInputStream("F:\\openCV\\mac\\libopencv_java245.dylib");
         // in =
Trail.class.getResourceAsStream("/opencv/mac/libopencv_java245.dylib");
          fileOut = File.createTempFile("lib", ".dylib");
       }
       OutputStream out = FileUtils.openOutputStream(fileOut);
     System.out.println(in.toString());
       IOUtils.copy(in, out);
       in.close();
       out.close();
       System.load(fileOut.toString());
```

```
        }
          catch (FileNotFoundException ex)
           {
             System.out.println(ex.getMessage());
           }
           catch(IOException ex)
           {
             System.out.println(ex.getMessage());
           }
           catch(Exception ex)
           {
             System.out.println(ex.getMessage());
           }
        }
```

To get path of folder contain jar file

```
private static String get_path()
  {
     String path="";
     try {
   URL url =
 Trail.class.getProtectionDomain().getCodeSource().getLocation();
 //Gets the path
         String jarPath = null;
```

```
                try{
                        jarPath = URLDecoder.decode(url.getFile(), "UTF-8");
//Should fix it to be read correctly by the system
                    }
                catch (UnsupportedEncodingException e) {
                    System.out.println(e.getMessage());
                }
                catch (Exception e) {
                        System.out.println(e.getMessage());
                }
                    String parentPath = new
File(jarPath).getParentFile().getPath();    //Path of the jar
                parentPath = parentPath + File.separator;
        System.out.println("Path: " + parentPath);
        System.out.println(path);
        path=parentPath;
        } catch (Exception ex) {
     Logger.getLogger(Trail.class.getName()).log(Level.SEVERE, null, ex);
        }
      return path;
   }
        return dst;
  }
```

4.3 check webcam availability implementation

```
Try
{

VideoCapture capture=new VideoCapture(1);

    if(capture.open(1))
    {

//JOptionPane.showMessageDialog(null,"this
webcam not work well");
    }
  else
    {

JOptionPane.showMessageDialog(null,"this
webcam not work well");
    }
}
  catch (Exception ex)
{
    System.out.println(ex.getMessage());
}
```

4.4Digital image processing

Digital image processing deals with manipulation of digital images through a digital computer. It is a subfield of signals and systems but focus particularly on images. DIP focuses on developing a computer system that is able to perform processing on an image. The input of that system is a digital image and the system process that image using efficient algorithms, and gives an image as an output. The most common example is Adobe Photoshop. It is one of the widely used application for processing digital images.

4.4.1 How it works.

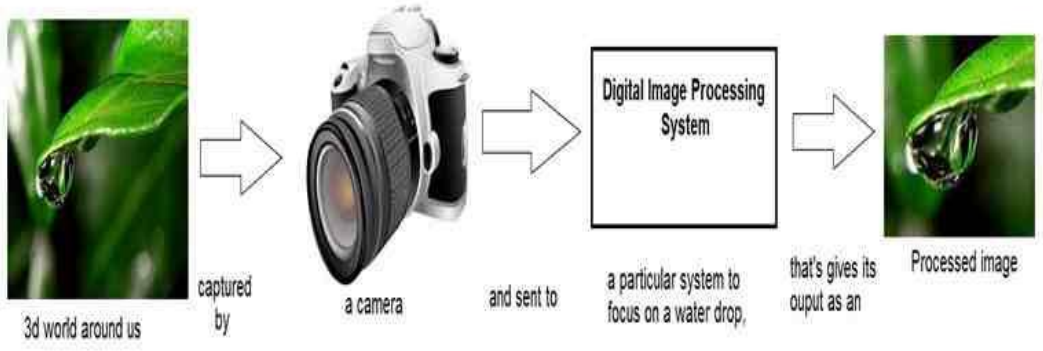


Figure 4.1 how DIP work

In the above figure, an image has been captured by a camera and has been sent to a digital system to remove all the other details, and just focus on the water drop by zooming it in such a way that the quality of the image remains the same.

4.4.2 Grayscale

In photography and computing, a **grayscale** or **greyscale** digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-

and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest.

Grayscale images are distinct from one-bit bi-tonal black-and-white images, which in the context of computer imaging are images with only the two colors, black, and white (also called bilevel or binary images). Grayscale images have many shades of gray in between.

Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the electromagnetic spectrum (e.g.infrared, visible light, ultraviolet, etc.), and in such cases they are monochromatic proper when only a given frequency is captured. But also they can be synthesized from a full color image.[5]

4.4.2.1 Converting color to grayscale

Conversion of a color image to grayscale is not unique; different weighting of the color channels effectively represent the effect of shooting black-and-white film with different-colored photographic filters on the cameras.

Colorimetric (luminance-preserving) conversion to grayscale

A common strategy is to use the principles of photometry or, more broadly, colorimetry to match the luminance of the grayscale image to the luminance of the original color image. This also ensures that both images will have the same absolute luminance, as can be measured in its SI units of candelas per square meter, in any given area of the image, given equal whitepoints. In addition, matching luminance provides matching perceptual lightness measures, such as L^* (as in the 1976 CIE *L*ab color space) which is determined by the luminance Y (as in the CIE 1931 *XYZ* color space) .

To convert a color from a colorspace based on an RGB color model to a grayscale representation of its luminance, weighted sums must be calculated in a linear RGB space, that is, after the gamma compression function has been removed first via gamma expansion.

For the sRGB color space, gamma expansion is defined as

$$C_{\text{linear}} = \begin{cases} \frac{C_{\text{srgb}}}{12.92}, & C_{\text{srgb}} \leq 0.04045 \\ \left(\frac{C_{\text{srgb}}+0.055}{1.055}\right)^{2.4}, & C_{\text{srgb}} > 0.04045 \end{cases}$$

where C_{srgb} represents any of the three gamma-compressed sRGB primaries (R_{srgb}, G_{srgb}, and B_{srgb}, each in range [0,1]) and C_{linear} is the corresponding linear-intensity value (R, G, and B, also in range [0,1]).

Then, luminance is calculated as a weighted sum of the three linear-intensity values. The sRGB color space is defined in terms of the CIE 1931linear luminance Y, which is given by

$$Y = 0.2126R + 0.7152G + 0.0722B.$$ [5]

The coefficients represent the measured intensity perception of typical trichromat humans, depending on the primaries being used; in particular, human vision is most sensitive to green and least sensitive to blue. To encode grayscale intensity in linear RGB, each of the three primaries can be set to equal the calculated linear luminance Y (replacing R,G,B by Y,Y,Y to get this linear grayscale). Linear luminance typically needs to be gamma compressed to get back to a conventional non-linear representation. For sRGB, each of its three primaries is then set to the same gamma-compressed Y_{srgb} given by the inverse of the gamma expansion above as

$$Y_{\text{srgb}} = \begin{cases} 12.92\ Y, & Y \leq 0.0031308 \\ 1.055\ Y^{1/2.4} - 0.055, & Y > 0.0031308. \end{cases}$$

In practice, because the three sRGB components are then equal, it is only necessary to store these values once in sRGB-compatible image formats that support a single-channel representation. Web browsers and other software that recognizes sRGB images will typically produce the same rendering for a such a grayscale image as it would for an sRGB image having the same values in all three color channels.

Luma coding in video systems

For images in color spaces such as Y'UV and its relatives, which are used in standard color TV and video systems such as PAL, SECAM, and NTSC, a nonlinear luma component(Y') is calculated directly from gamma-compressed primary intensities as a weighted sum, which can be calculated quickly without the gamma expansion and compression used in colorimetric grayscale calculations. In the Y'UV and Y'IQ models used by PAL and NTSC, the rec601 luma (Y') component is computed as

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

where we use the prime to distinguish these gamma-compressed values from the linear R, G, B, and Y discussed above. The ITU-R BT.709 standard used for HDTV developed by the ATSC uses different color coefficients, computing the luma component as

$$Y' = 0.2126R' + 0.7152G' + 0.0722B'.$$

Although these are numerically the same coefficients used in sRGB above, the effect is different because here they are being applied directly to gamma-compressed values.

Normally these colorspaces are transformed back to R'G'B' before rendering for viewing. To the extent that enough precision remains, they can then be rendered accurately.

But if the luma component by itself is instead used directly as a grayscale representation of the color image, luminance is not preserved: two colors can have the same luma Y' but different CIE linear luminance Y (and thus different nonlinear Y_{srgb} as defined above) and therefore appear darker or lighter to a typical human than the original color. Similarly, two colors having the same luminance Y (and thus the same Y_{srgb}) will in general have different luma by either of the Y' luma definitions above.[5]

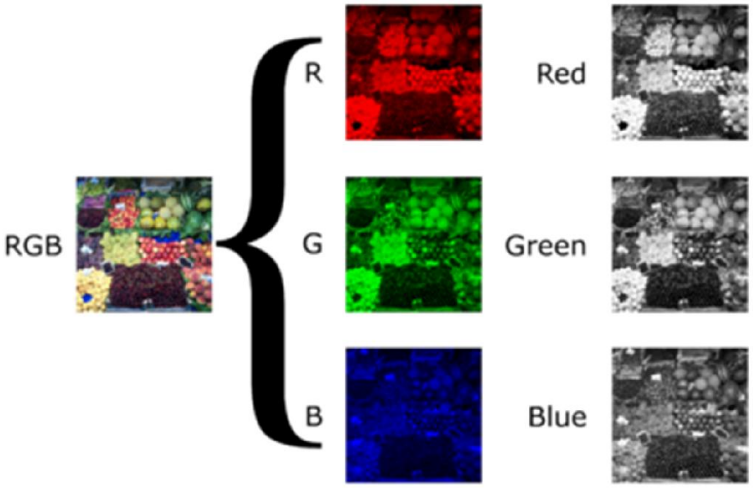


Figure 4.2 Composition of RGB from 3 Grayscale images

4.4.2.1.1 Converting color to grayscale implementation

```
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;

import java.io.File;
import javax.imageio.ImageIO;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.imgproc.Imgproc;

public class Main

 {

  public static void main( String[] args )
 {

    try
 {

loadLibrary( Core.NATIVE_LIBRARY_NAME
);

        File input = new
File("digital_image_processing.jpg");
        BufferedImage image =
ImageIO.read(input);
```

```
        byte[] data = ((DataBufferByte)
image.getRaster().getDataBuffer()).getData();
        Mat mat = new Mat(image.getHeight(),
image.getWidth(), CvType.CV_8UC3);
        mat.put(0, 0, data);

        Mat mat1 = new
Mat(image.getHeight(),image.getWidth(),CvType
.CV_8UC1);

        Imgproc.cvtColor(mat, mat1,
Imgproc.COLOR_RGB2GRAY);

        byte[] data1 = new byte[mat1.rows() *
mat1.cols() * (int)(mat1.elemSize())];
  mat1.get(0, 0, data1);
        BufferedImage image1 = new
BufferedImage(mat1.cols(),mat1.rows(),
BufferedImage.TYPE_BYTE_GRAY);

     image1.getRaster().setDataElements(0, 0,
mat1.cols(), mat1.rows(), data1);

        File ouptut = new File("grayscale.jpg");

        ImageIO.write(image1, "jpg", ouptut);

     }
catch (Exception e)

  {
        System.out.println("Error: " +
e.getMessage());

     }
     }
     }
```

Original Image

Figure 4.3 original image before grayscale

Grayscale Image

Figure 4.4 grayscale image

4.4.3 Image enhancement

the process of improving the quality of a digitally stored image by manipulating the image with software. It is quite easy, for example, to make an image lighter or darker, or to increase or decrease contrast. Advanced image enhancement software also supports many filters for altering images in various ways.

Figure 4.5 image enhancement

4.4.3.1 light normalization

- Face Detection remains an unsolved problem in general
  - Pose variation
  - Illumination variation
- Canonical form approaches for illumination variation
  - Normalize the variation in appearance

HE, GIC, QIR

4.4.3.2 Histogram equalization (HE)

Mentioned in chapter 2 (2.2.5.2.2)

Result of HE

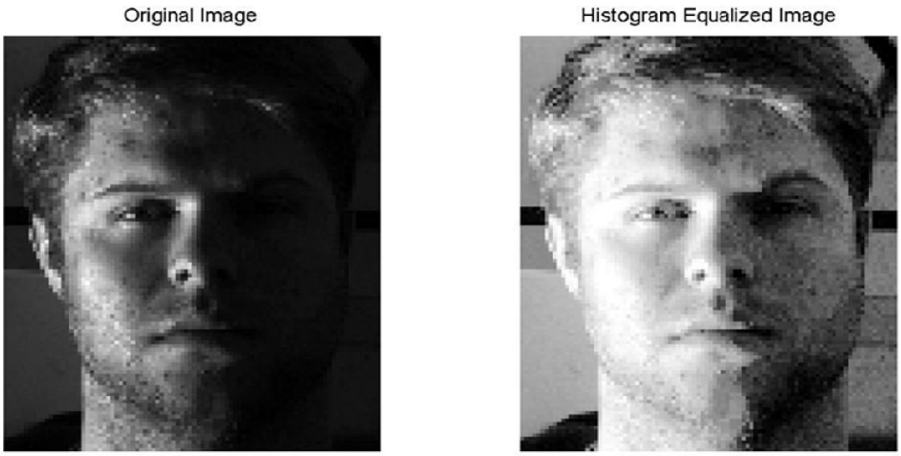


Figure 4.6 Result of HE

4.4.3.2.1 HE implementation

```
import org.opencv.core.Core;

import org.opencv.core.Mat;

import org.opencv.highgui.Highgui;

import org.opencv.imgproc.Imgproc;

public class Main

{

  static int width;

  static int height;
```

```
  static double alpha = 2;
  static double beta = 50;
  public static void main( String[] args )
{
    try
 {
       System.loadLibrary( Core.NATIVE_LIBRARY_NAME );
       Mat source = Highgui.imread("grayscale.jpg",
       Highgui.CV_LOAD_IMAGE_GRAYSCALE);
       Mat destination = new
Mat(source.rows(),source.cols(),source.type());
         Imgproc.equalizeHist(source, destination);
        Highgui.imwrite("contrast.jpg", destination);
    }
// handle any exception may occur
catch (Exception e)
 {
       System.out.println("error: " + e.getMessage());
  }
   }
}
```

4.4.3.3 Gamma Intensity Correction

Correct the overall brightness of the face image to a predefined ‘canonical’ image

Canonical image

Face image under some normal lighting condition

Correction is performed using Gamma Trans.

Result of GIC

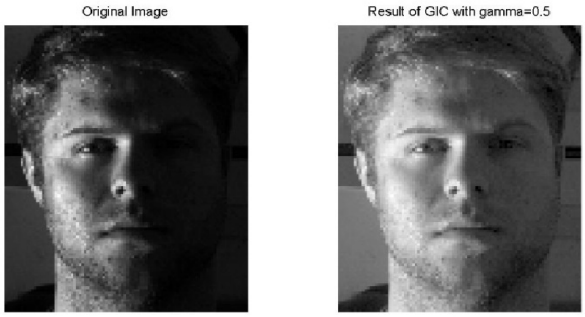


Figure 4.7 Result of GIC

4.4.3.4 Normalization

normalization is a process that changes the range of pixel intensity values. Applications include photographs with poor contrast due to glare, for example. Normalization is sometimes called contrast stretching or histogram stretching. In more general fields of data processing, such as digital signal processing, it is referred to asdynamic range expansion.

The purpose of dynamic range expansion in the various applications is usually to bring the image, or other type of signal, into a range that is more familiar or normal to the senses, hence the term normalization. Often, the motivation is to achieve consistency in dynamic range for a set of data, signals, or images to avoid mental distraction or fatigue.

The linear normalization of a grayscale digital image is performed according to the formula

$$I_N = (I - \text{Min})\frac{\text{newMax} - \text{newMin}}{\text{Max} - \text{Min}} + \text{newMin}$$

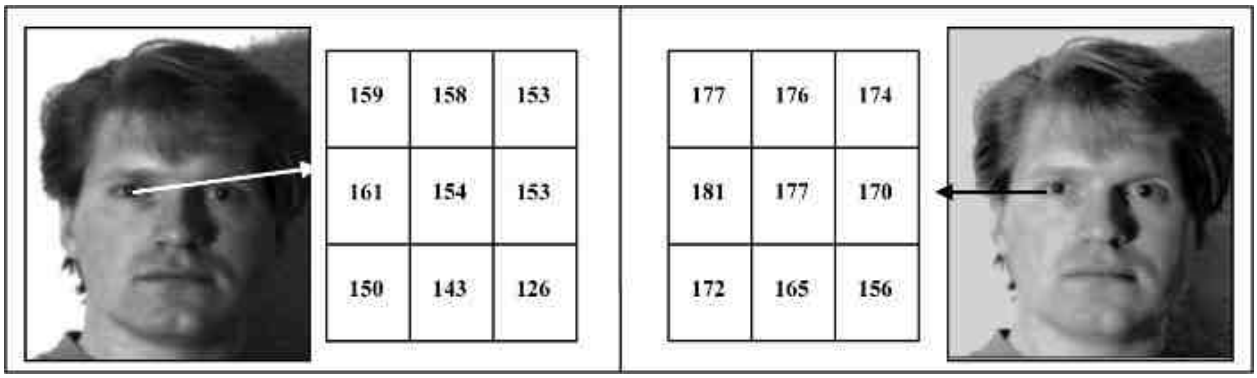


Figure 4.8 Result of normalization

4.4.3.1 illumination normalization implementation

```
Mat img_normalization(Mat img)

 {

   Mat   float_gray, blur, num, den ;

   float_gray=new Mat();

  // convert to floating-point image

  img.convertTo(float_gray, CvType.CV_32F, 1/255.0);

  blur=new Mat();

  num=new Mat();

  den=new Mat();

 // numerator = img - gauss_blur(img)

  Imgproc.GaussianBlur(float_gray, blur, new Size(0,0), 2,2);
```

```
Core.subtract(float_gray, blur, num);

// denominator = sqrt(gauss_blur(img^2))

Imgproc.GaussianBlur(num.mul(num), blur,new Size(0,0), 20, 20);

Core.pow(blur, 0.5, den);

// normalize output into [0,1]

Core.normalize(img, img, 77.5, 177.5, Core.NORM_MINMAX, -1);

return img;

}
```

4.5 Face detection

Face detection is a computer technology that identifies human faces in digital images. It detects human faces which might then be used for recognizing a particular face. This technology is being used in a variety of applications nowadays.

Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene.

4.5.1 face detection implementation

Step 1:load opencv

loadLibrary();

Step 2: Load the Haar cascade

```
// load cascade file from application resources

InputStream is =
```

```
getResources().openRawResource(R.raw.lbpcasca
de_frontalface);
   File cascadeDir = getDir("cascade",
Context.MODE_PRIVATE);
   mCascadeFile = new File(cascadeDir,
"lbpcascade_frontalface.xml");
   FileOutputStream os = new
FileOutputStream(mCascadeFile);
```

Step 3: Instantiate the OpenCV Cascade Classifier

```
   mJavaDetector = new
CascadeClassifier(mCascadeFile.getAbsolutePath
());
   if (mJavaDetector.empty())
   {
       Log.e(TAG, "Failed to load cascade
classifier");
       mJavaDetector = null;
   }
   else
       Log.i(TAG, "Loaded cascade classifier
from " + mCascadeFile.getAbsolutePath());

   cascadeDir.delete();
```

Step 4: For each video frame received, call the cascade classifier

```
   public Mat onCameraFrame(Mat inputFrame)
// Called by framework with latest frame
   {
       inputFrame.copyTo(mRgba);
       Imgproc.cvtColor(inputFrame, mGray,
Imgproc.COLOR_RGBA2GRAY); // Convert to
grayscale
       MatOfRect faces = new MatOfRect();
```

```
        if (mJavaDetector != null)
            mJavaDetector.detectMultiScale(mGray,
faces, 1.1, 2, 2, // TODO:
objdetect.CV_HAAR_SCALE_IMAGE
                    new Size(mAbsoluteFaceSize,
mAbsoluteFaceSize), new Size());      }

   // Each rectangle in the faces array is a face
   // Draw a rectangle around each face
   Rect[] facesArray = faces.toArray();
   for (int i = 0; i < facesArray.length; i++)
          Core.rectangle(mRgba, facesArray[i].tl(),
facesArray[i].br(), FACE_RECT_COLOR, 3);

   return mRgba;
```

4.6 mouse location calculation in VMC

Position of mouse cursor calculated from mapping equation between Window size that contain image and screen size and previous image if it has previous image.

As for example if we have So if the video image is 640 x 480 and the screen is 1280 x 960, then if the trackpoint in the video frame is (160, 120) then the mouse pointer in the monitor screen is (320, 240).

To smooth out the jitteriness of the cursor in Camera Mouse and EagleEyes we use a well-known method called Exponential Smoothing (see references at the end). The idea is that the position of

the cursor should depend both on the current calculation for the position and on the past cursor positions. That way if there is an anomaly or jitter in the current reading it will be smoothed out by past readings. This also allows for relatively low resolution readings to spread across a higher resolution screen.

We assume we are getting a sequence of calculations of x, y screen coordinates over time (xt, yt) from the EagleEyes box or Camera Mouse software. These are (x1, y1), (x2, y2), (x3, y3) and so on as time progresses. (I should use subscripts for the t but I'm too lazy.) We want to display the cursor at the smoothed values (sxt, syt). The cursor actually will be displayed at (sx1, sy1), (sx2, sy2), (sy3, sy3) and so on over time.

Initially we need to decide on a smoothing factor a, which has a value between 1 and 0. If a is 1 then there is no smoothing (The current readings count for 100% and the past for 0%). If a is 0.9 there is little smoothing (The current readings count for 90% and the past for 10%). If a is 0.1

There is lots of smoothing (The current readings count for 10% and the past for 90%) and it will be like driving a car on a sheet of ice.

To begin we set sx1 = x1 and sy1 = y1. No smoothing for the very first moment of time. Then sx2 = a * x2 + (1 – a) * sx1

and

sy2 = a * y2 + (1 – a) * sy1

Assume a is set to 0.8. Then sx2, the smoothed x value of the cursor position at time 2, is set Equal to 80% times x2, the reading at time 2, plus 20% times the previous cursor position sx1 at time 1.

Then

sx3 = a * x3 + (1 – a) * sx2

And

sy3 = a * y3 + (1 – a) * sy2

And so on forever. Note that with Exponential Smoothing you don't actually need to record the past readings in an array and do complicated

calculations. All the past that you need is accumulated in the values of sxt and syt.

Intuitively the new smoothed coordinate is 80% based on the new reading and 20% based on the past readings. That is with a = 0.8. If we set a = 0.3 then the new smoothed coordinate would be 30% based on the new reading and 70 % based on the past readings. Much more smoothing in the case where a = 0.3; the past has much more influence. [11]

4.6.1 mouse motion according to mouse implementation in VMC

```
package VMC;

import java.awt.AWTException;

import java.awt.Dimension;

import java.awt.List;

import java.awt.MouseInfo;

import java.awt.PointerInfo;

import java.awt.Robot;

import java.awt.Toolkit;

import java.util.ArrayList;

import javax.swing.JOptionPane;

import org.opencv.core.Mat;

import org.opencv.video.KalmanFilter;

import trail.position;

class facetracker
```

```
{
    int top=0;
    static double x_prev;
    static double y_prev;
   static boolean get_first=false;
   ArrayList<position> positions=new ArrayList<>();
   double a=.4;
   position first;
    position second;
    public void set_positions(position p1,position p2)
    {
       first=p1;
       second=p2;
    }
    public void decision() throws AWTException
    {
      // int top=0;
       double a_paper=.4; /// from wor send to me
       int alphax=13;
       int alphay=7;
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
double width = screenSize.getWidth();
double height = screenSize.getHeight();
double height_ratio=height/(2*alphay);
double width_ratio=width/(2*alphax);
```

```
PointerInfo aa = MouseInfo.getPointerInfo();
        java.awt.Point b = aa.getLocation();
        double x_pos= b.getX();
       double y_pos=b.getY();
        Robot mouse=new Robot();
       /*if(get_first==false)
         y_prev=(int)((second.y*height_ratio)+(1-a)*y_prev);
         x_prev=(((a*second.x*width_ratio))+(1-a)*x_prev);
       }*/
        if(second.x<first.x-alphax)
           second.x=first.x-alphax;
        if(second.x>first.x+alphax)
           second.x=first.x+alphax;
        if(second.y<first.y-alphay)
           second.y=first.y-alphay;
        if(second.y>first.y+alphay)
           second.y=first.y+alphay;
        int dx=(int) (1366-((second.x-(first.x-alphax ))*width_ratio));
        int dy=(int) ((second.y-(first.y-alphay ))*height_ratio);
      //   System.out.println(dx+","+dy);
        if(get_first==false)
        {
        mouse.mouseMove(dx,dy);
```

```
x_prev=dx;
y_prev=dy;
get_first=true;
}
else
{
mouse.mouseMove((int)((a_paper*dx)+((1-
a_paper)*x_prev)),(int)((dy*a_paper)+((1-a_paper)*y_prev)));
y_prev=((dy*a_paper)+((1-a_paper)*y_prev));
x_prev=((a_paper*dx)+((1-a_paper)*x_prev));
}
/*
try
{
//if(get_first==true)
{
int valx=Math.abs((int)second.getX()-(int)x_prev);
int valy=Math.abs((int)second.getY()-(int)y_prev);
System.err.println(x_prev +","+y_prev);
System.err.println(second.getX() +","+second.getY());
}
position getaverage(ArrayList<position> list)
{
```

```
    position temp=new position(0, 0);
    for(position pos: list)
    {
    temp.x+=pos.x;
    temp.y+=pos.y;
    }
    temp.x/=list.size();
    temp.y/=list.size();
    return temp;
}
public position kalman(KalmanFilter kf,position pos)
{
    Mat m=new Mat();
    m.put(0, 0, pos.x);
   m.put(0, 1, pos.y);
    //KalmanFilter kf=new KalmanFilter(4, 2);
   Mat corect= kf.correct(m);
   Mat state=kf.predict();
   position poss=new position();
   poss.x=state.get(0, 0)[0];
   poss.y=state.get(0, 0)[0];
   return poss
}
```

}

4.7 speech recognition

In computer science and electrical engineering, **speech recognition** (SR) is the translation of spoken words into text. It is also known as "automatic speech recognition" (ASR), "computer speech recognition", or just "speech to text" (STT).

Some SR systems use "training" (also called "enrolment") where an individual speaker reads text or isolated vocabulary into the system. The system analyzes the person's specific voice and uses it to fine-tune the recognition of that person's speech, resulting in more increased accuracy. Systems that do not use training are called "speaker independent" systems. Systems that use training are called "speaker dependent".

Speech recognition applications include voice user interfaces such as voice dialling (e.g. "Call home"), call routing (e.g. "I would like to make a collect call"), domotic appliance control, search (e.g. find a podcast where particular words were spoken), simple data entry (e.g., entering a credit card number), preparation of structured documents (e.g. a radiology report), speech-to-text processing (e.g., word processors or emails), and aircraft (usually termed Direct Voice Input).

The term *voice recognition* or *speaker identification* refers to identifying the speaker, rather than what they are saying. Recognizing the speaker can simplify the task of translating speech in systems that have been trained on a specific person's voice or it can be used to authenticate or verify the identity of a speaker as part of a security process.

From the technology perspective, speech recognition has a long history with several waves of major innovations. Most recently, the field has benefited from advances in deep learning and big data. The advances are evidenced not only by the surge of academic papers published in the field, but more importantly by the world-wide industry adoption of a variety of deep learning methods in designing and deploying speech recognition systems. These speech industry players include Microsoft, Google, IBM, Baidu (China), Apple, Amazon, Nuance, IflyTek (China), many of which have publicized the core technology in their speech recognition systems being based on deep learning.[13]

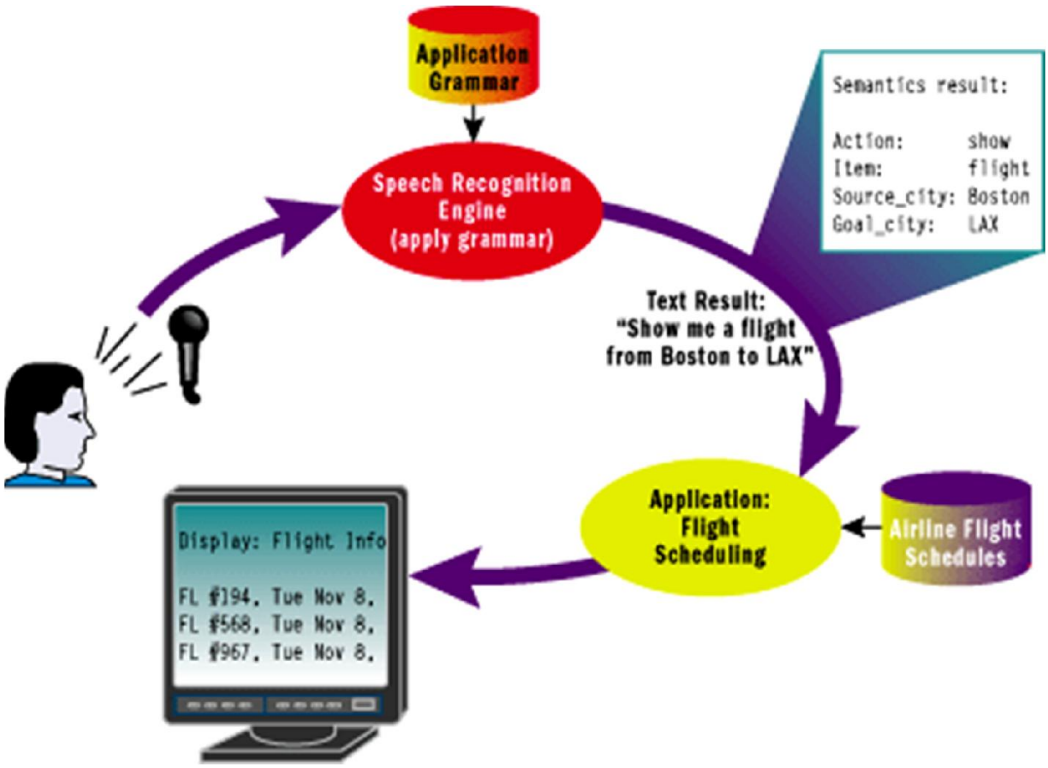


Figure 4.9 speech recognition grammar

Speech Recognition Grammar Specification (SRGS) is a W3C standard for how *speech recognition grammars* are specified. A speech recognition grammar is a set of word patterns, and tells a speech recognition system what to expect a human to say. For instance, if you call an auto attendant application, it will prompt you for the name of a person (with the expectation that your call will be transferred to that person's phone). It will then start up a speech recognizer, giving it a speech recognition grammar. This grammar contains the names of the people in the auto attendant's directory and a collection of sentence patterns that are the typical responses from callers to the prompt.

SRGS specifies two alternate but equivalent syntaxes, one based on XML, and one using Augmented BNF format. In practice, the XML syntax is used more frequently.

Both the ABNF Form and XML Form have the expressive power of a Context Free Grammar. A grammar processor that does not support recursive grammars has the expressive power of a Finite State Machine or regular expression language.

If the speech recognizer returned just a string containing the actual words spoken by the user, the voice application would have to do the tedious job of extracting the semantic meaning from those words. For this reason, SRGS grammars can be decorated with *tag* elements, which when executed, build up the semantic result. SRGS does not specify the

contents of the tag elements: this is done in a companion W3C standard, Semantic Interpretation for Speech Recognition (SISR). SISR is based on ECMAScript, and ECMAScript statements inside the SRGS tags build up an ECMAScript semantic result object that is easy for the voice application to process.

Both SRGS and SISR are W3C Recommendations, the final stage of the W3C standards track. The W3C VoiceXML standard, which defines how voice dialogs are specified, depends heavily on SRGS and SISR.

4.7.1 speech recognition grammar implementation

```
JSGF V1.0;
grammar hello;
public <greet> = (left | right | control | release );
explanation :-
left means left click in mouse
right means right click in mouse
control means down ctrl button in keyboard
release means up ctrl button in keyboard
```

4.7.2 speech recognition implementation(by sphinx)

```
package Speech;

import edu.cmu.sphinx.frontend.util.Microphone;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
import edu.cmu.sphinx.util.props.PropertyException;
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;
```

```
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Speach_recognition {

   /**
    * Main method for running the HelloWorld demo.
    */
   public static void main(String[] args) {
      try {
         URL url;
         if (args.length > 0) {
            url = new File(args[0]).toURI().toURL();
         } else {
            url =
Speach_recognition.class.getResource("helloworld.config.xml");
         }
         System.out.println("Loading...");
         ConfigurationManager cm = new
ConfigurationManager(url);
           Recognizer recognizer = (Recognizer)
cm.lookup("recognizer");
           Microphone microphone = (Microphone)
cm.lookup("microphone");

         /* allocate the resource necessary for the recognizer */
         recognizer.allocate();

         /* the microphone will keep recording until the program exits
*/
           if (microphone.startRecording()) {
```

```
                System.out.println
                   ("Say: (LEFT | Right | Control | Release) ");

                while (true) {
                   System.out.println
                        ("Start speaking. Press Ctrl-C to quit.\n");
                /*
                 * This method will return when the end of speech
                 * is reached. Note that the endpointer will determine
                 * the end of speech.
                 */
                   Result result = recognizer.recognize();
                   if (result != null) {
                        String resultText =
result.getBestFinalResultNoFiller();
                System.out.println("You said: " + resultText + "\n");
                   Robot robot=new Robot();
                  if(resultText.equalsIgnoreCase("Left"))
                  {
                     robot.mousePress(InputEvent.BUTTON1_MASK);
    robot.mouseRelease(InputEvent.BUTTON1_MASK);
     try {
       //Thread.sleep(1000);
        }
     catch (Exception ex)
     {
       //
Logger.getLogger(mouse_actions.class.getName()).log(Level.SEVE
RE, null, ex);
     }
                  }
     else if(resultText.equalsIgnoreCase("right"))
                  {

robot.mousePress(InputEvent.BUTTON3_MASK);
    robot.mouseRelease(InputEvent.BUTTON3_MASK);
```

```
                }

            else if(resultText.equalsIgnoreCase("control"))
                    {
                //   robot.keyPress(KeyEvent.VK_CONTROL);
                    }

            else if(resultText.equalsIgnoreCase("release"))
                    {
                 //     robot.keyRelease(KeyEvent.VK_CONTROL);
                    }
                   }
                   else {
                    System.out.println("I can't hear what you said.\n");
                       }
                   }
                       }
              else {
                    System.out.println("Cannot start microphone.");
                    recognizer.deallocate();
                    System.exit(1);
                }
        } catch (IOException e) {
            System.err.println("Problem when loading HelloWorld: " +
e);
            e.printStackTrace();
        } catch (PropertyException e) {
            System.err.println("Problem configuring HelloWorld: " + e);
            e.printStackTrace();
        } catch (InstantiationException e) {
            System.err.println("Problem creating HelloWorld: " + e);
            e.printStackTrace();
        } catch (AWTException ex) {
Logger.getLogger(Speach_recognition.class.getName()).log(Level.S
EVERE, null, ex);
        }
    }}
```

Chapter 5

Testing

5.1 Overview

5.2 Testing system performance

5.1 Overviw

In previous chapters, the design and development processes of the proposed system are mentioned in details. However, this chapter provides a different look to the proposed system as the implantation phase of it. This chapter will provide the proposed system as a real running system not only conceptual model of it. This is done by providing test cases to` it with input and output screenshots to it.

5.2 Testing system performance

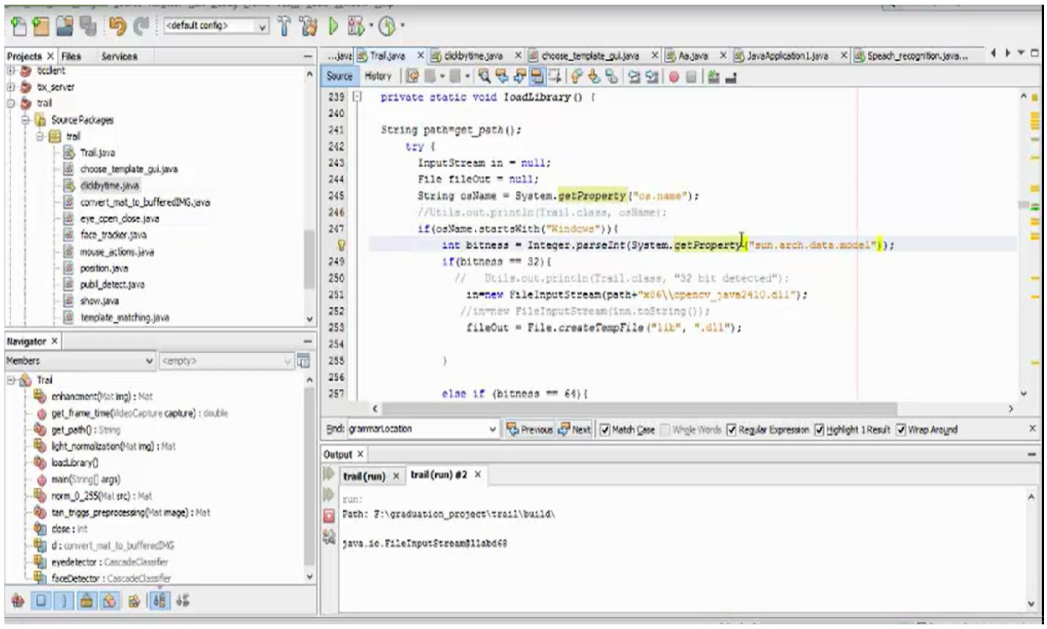


Figure 5.1: running the system

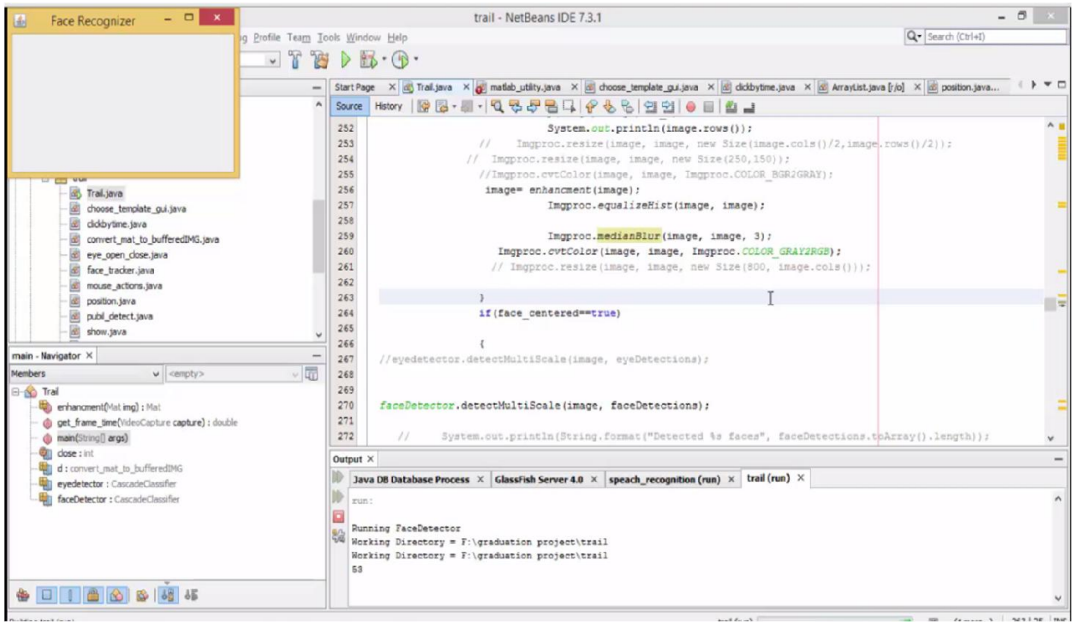


Figure 5.2: appearance of face recognizer

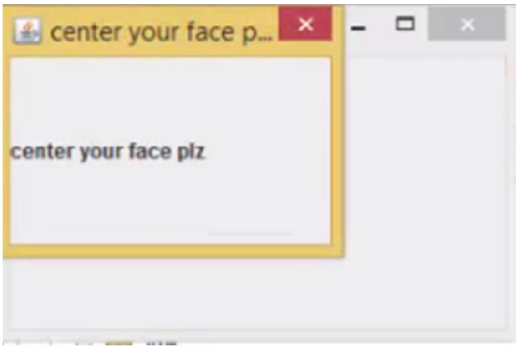


Figure 5.3: the system demand from the user to center his face

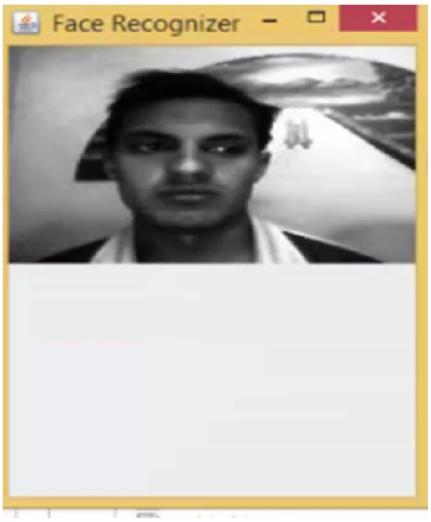


Figure 5.4: the user try to center his face

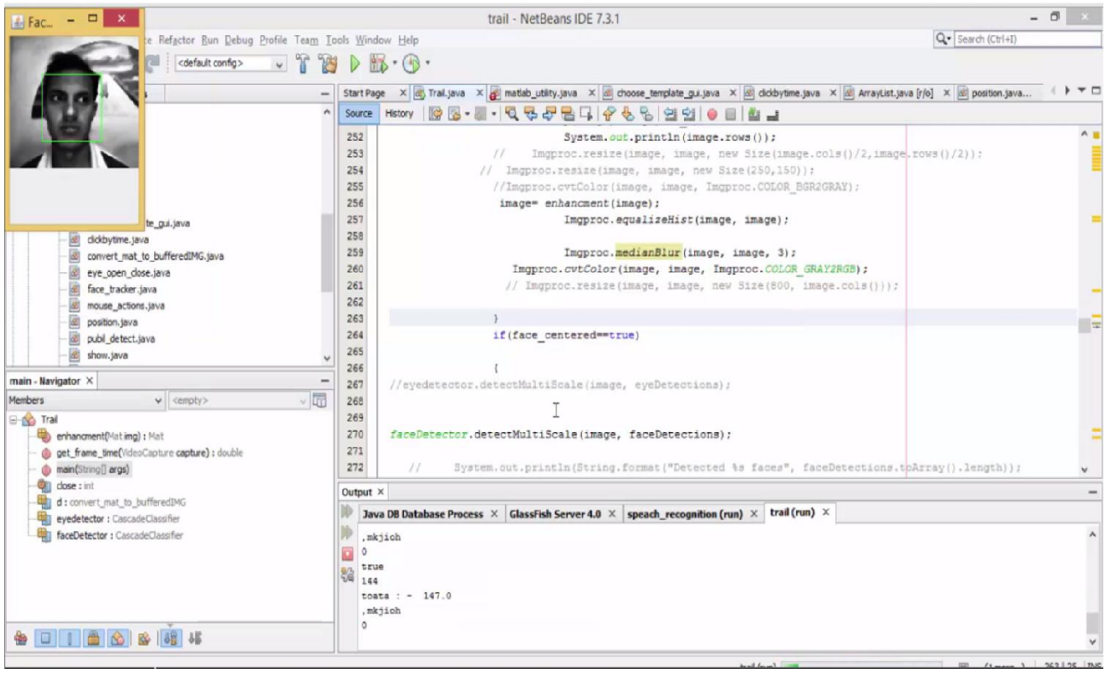


Figure 5.5: the system ready to use

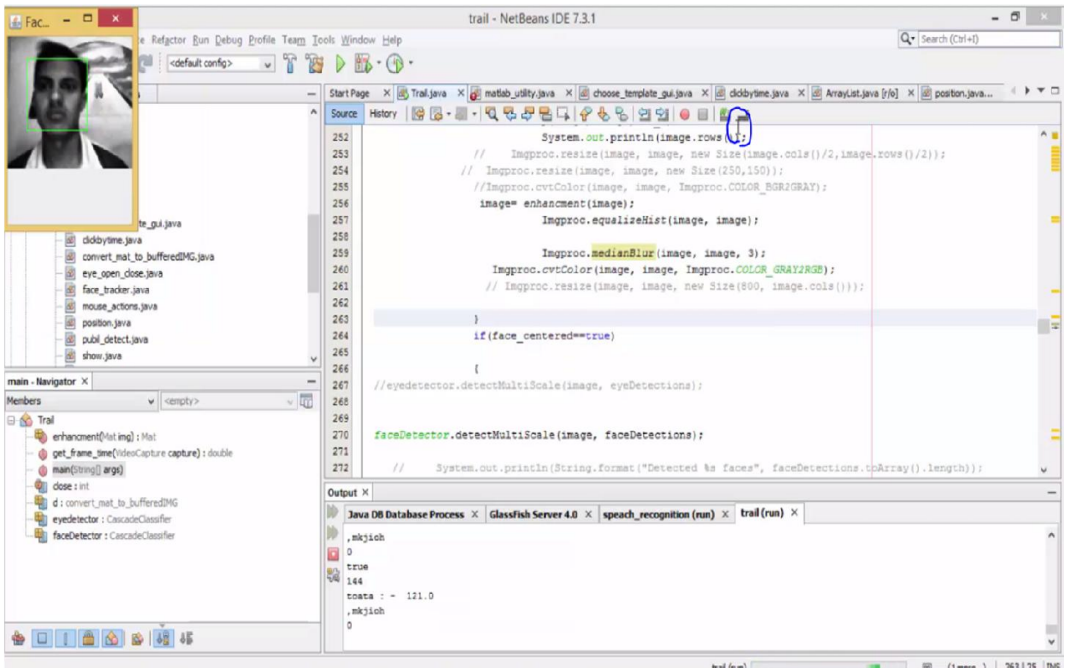


Figure 5.6: mouse pointer position

5.2.1 testing the system by speech command option

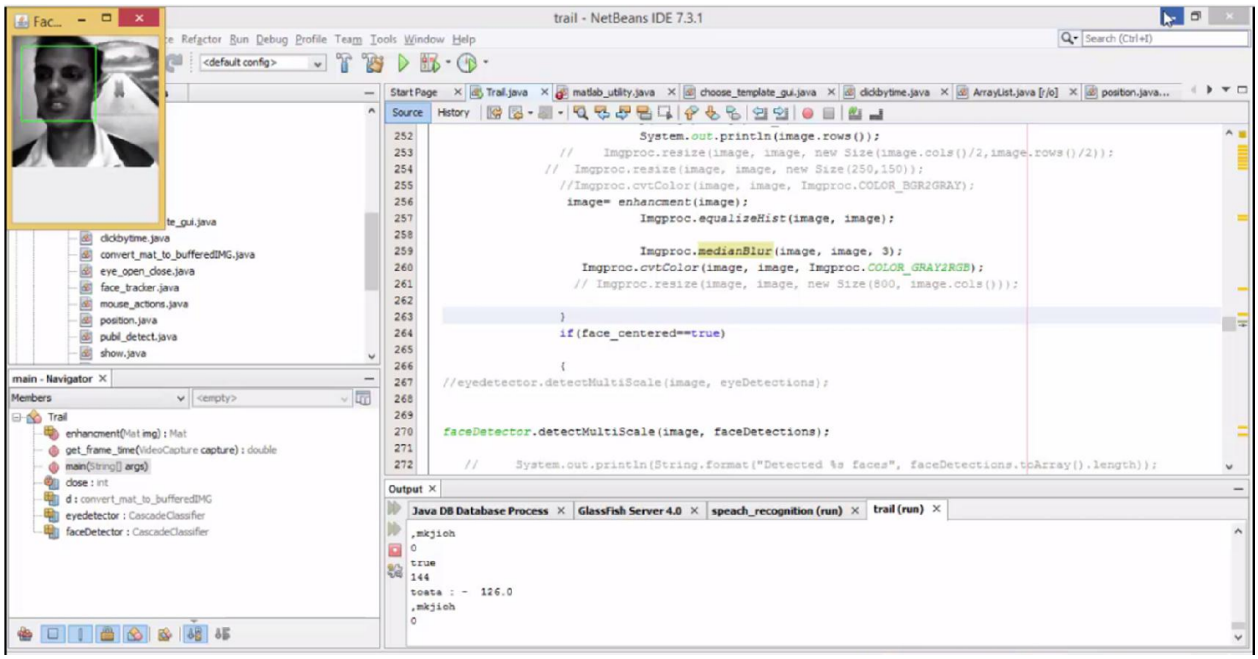


Figure 5.7: moving mouse pointer to minimize icon
to press left click by saying “left”

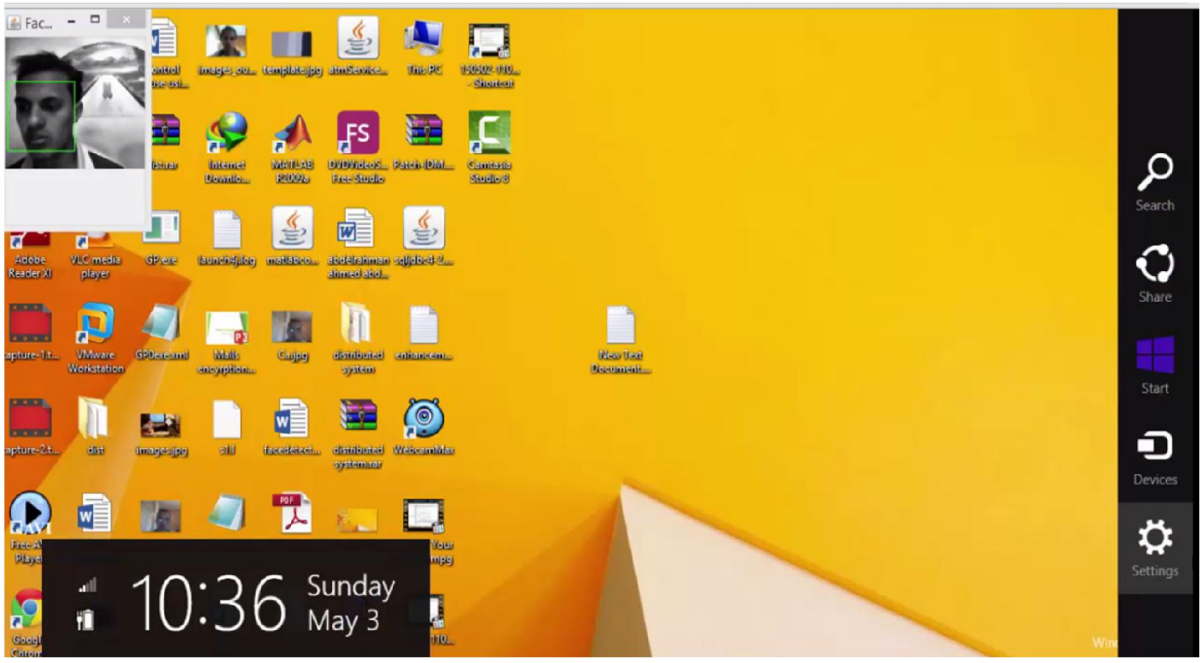


Figure 5.8: the result of left click

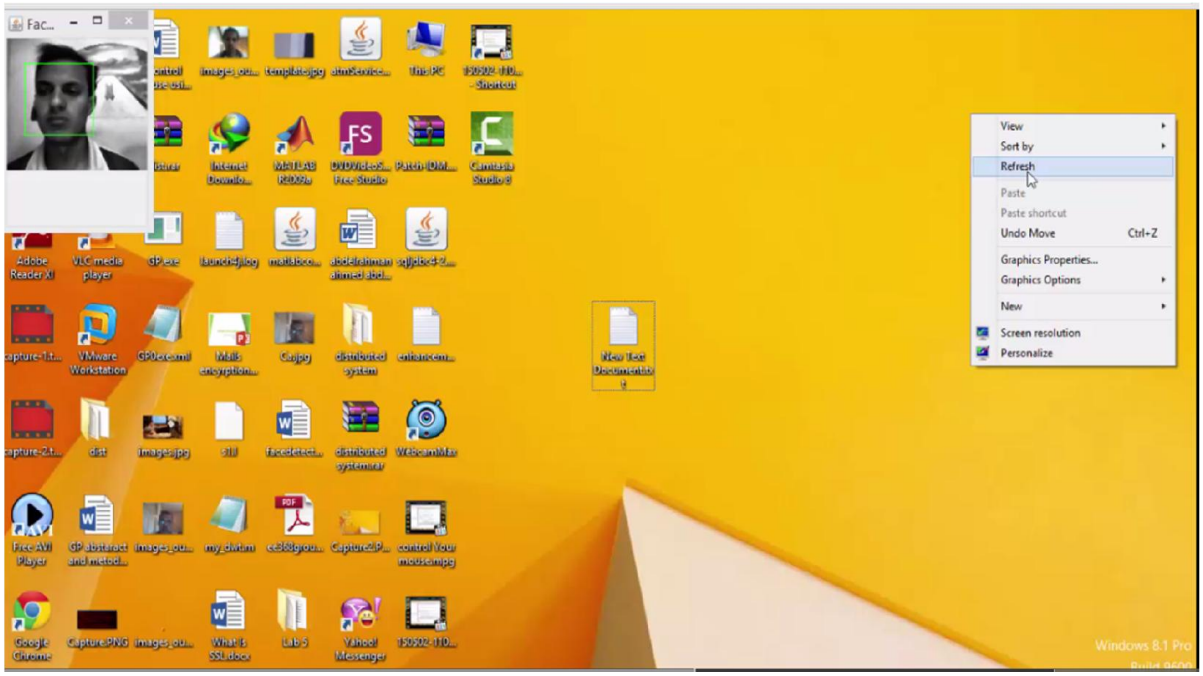


Figure 5.9: pressing right click by saying "right"

5.2.2 testing the system by time limiter option

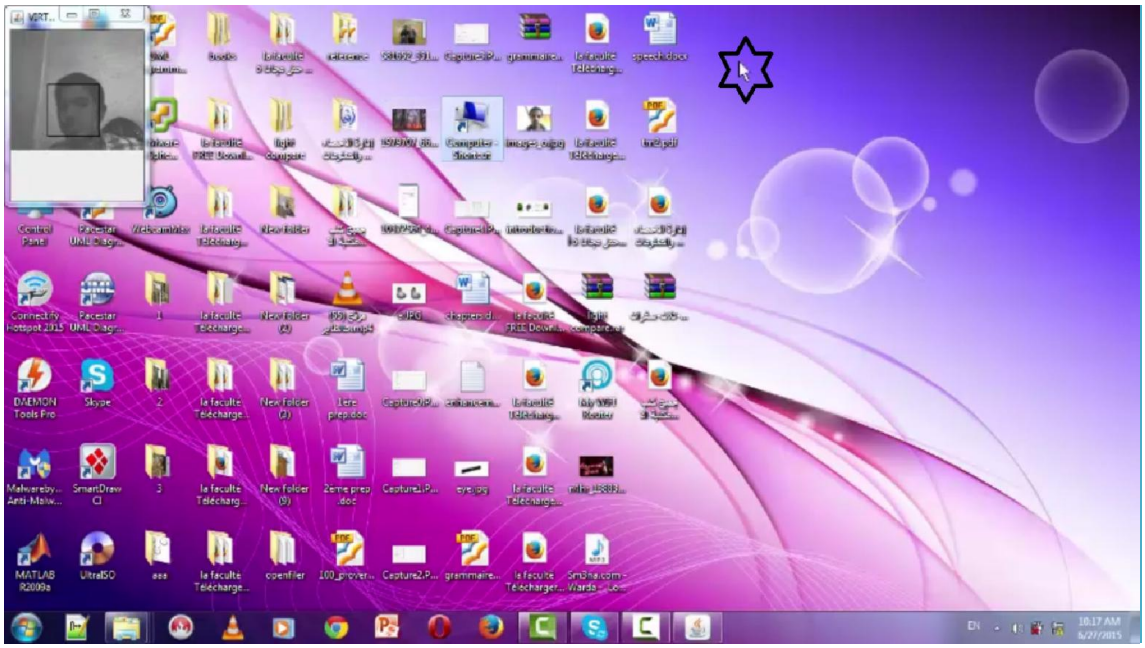


Figure 5.10: first mouse position

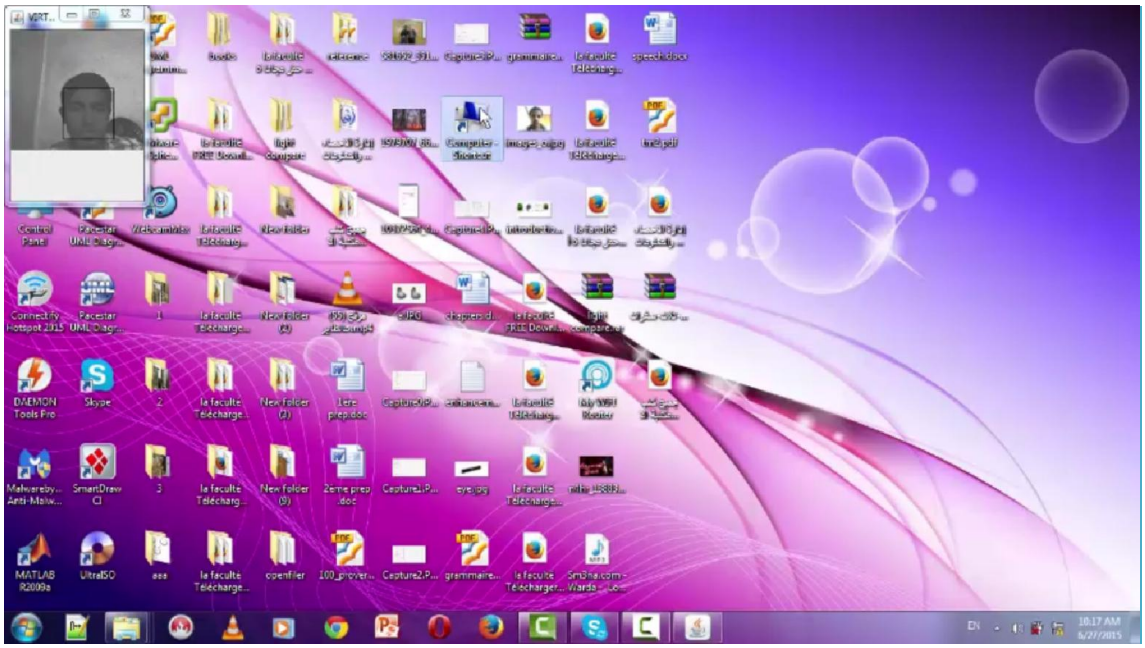


Figure 5.11: moving mouse pointer to my computer to press double click by time limiter

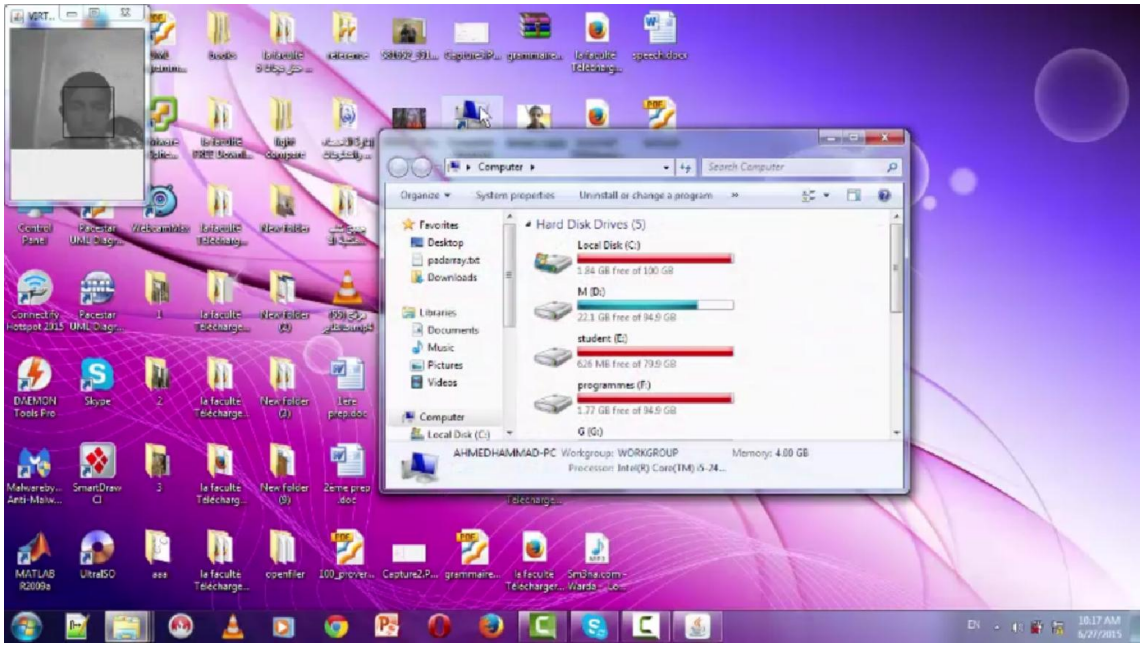


Figure 5.12: result of double click

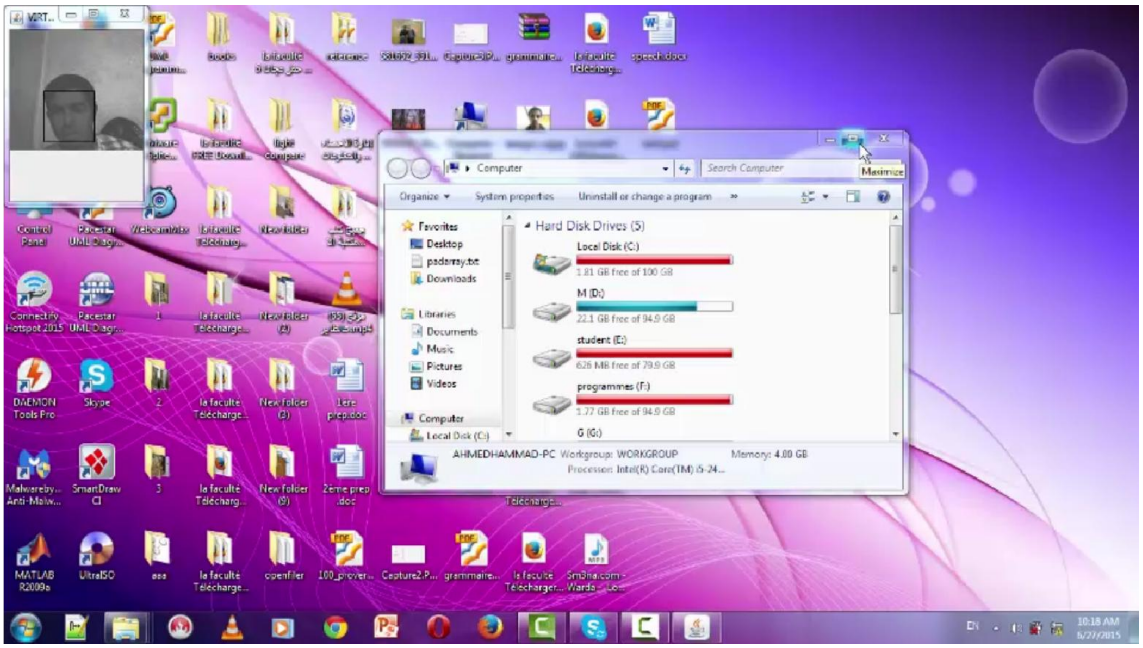


Figure 5.13 pressing left click on maximize icon by time limiter

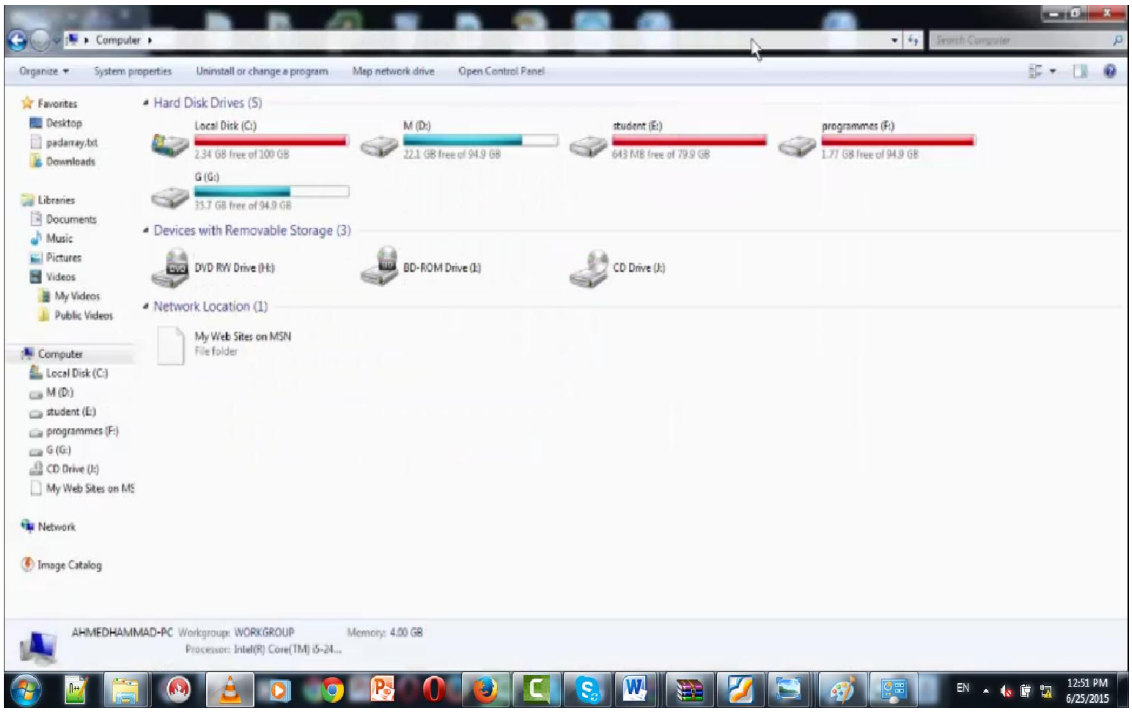


Figure 5.14 :result of left click

Chapter 6

Conclusion and Future Work

6.1 Overview

6.2 future work

6.3 difficulties

6.4 conclusion

6.1 Overview

Nowadays Computers have become more popular and important to our society and there are millions of users for it. Then I can said, how many people have arm disabilities hence they can't use computer in suitable manner because they can't use physical mouse . So, we have to find solution for this

The Virtual Mouse Controller (VMC), is represented as an intelligent assessment system for disabled people.

- **It has different characteristics which are the following:**

2. **Accessibility :** the system should be easy to access.
    - The application will be desktop-based and can be accessed from any device which the system will be installed on it.
    - The Application is developed by java programming language so it work in any platform(such as windows ,mac ,Linux …)
3. **Availability :** the system should be in a specified operable state at the start of the mission as NN Files replicated .
4. **Accuracy :** The system should provide accurate results
5. **Usability :** The system should be easy to use .
    - (as GUI need limit interaction from user as he have arm disabilities)

- **And it has different benefits such as :**

- Help arm disabilities with needing limit interaction from them(Almost no interaction)
- No additional H/W required(require only laptop webcam or any USB camera)
- Reduce Cost
- No need for traditional mouse

6.2 future work

We hope to achieve these things in the near future:-

1- To provide a feature of clicking by eye
   * closing left eye equal to press mouse's left button
   * closing right eye equal to press mouse's right button
   * closing both eyes equal to scroll

6.3 difficulties

1- camera resolution .

2- light difference.

6.4 conclusion

In Chapter 1: we refer to the following:

1- Project management plan

2- Organization of the project documentation

In Chapter 2: we refer to the following:

1- Traditional mouse
2- Image processing
3- Artificial Neural Network
4- Computer vision
5- Speech recognition
6- Related work

In chapter 3: we refer to the following:

1- Function and Non-function requirements
2- Use Case Diagram
3- Use case scenarios
4- System Sequence Diagram
5- Data Flow Diagram
6- Flow chart diagram

In Chapter 4: we refer to the implementation phase of the project life cycle

In Chapter 5 : we refer to the test phase of the project life cycle

REFERENCES

[1] arkdin collaboration service . 2015.Ultimate Guide to Speech Recognition. Accuconference. Accuconference (Accessed 2015-3-1).

[2] Christopher MacLeod. The Back Propagation Algorithm. The Back Propagation Algorithm.16.

[3] Computer vision.Wikipedia.2015. https://en.wikipedia.org/wiki/Computer_vision (Accessed 2015-2-5).

[4] Exponential smoothing. Wikipedia.2015. https://en.wikipedia.org/wiki/Exponential_smoothing (Accessed 2015-4-4).

[5] Grayscale. Wikipedia.2015. https://en.wikipedia.org/wiki/Grayscale (Accessed 2015-1-12).

[6] HYPERMEDIA IMAGE PROCESSING REFERENCE.2003. Digital Filters. HIPR2. http://homepages.inf.ed.ac.uk/rbf/HIPR2/copyrght.htm (Accessed 2015-1-10).

[7] James Gips (2015) *Camera Mouse*, Boston College: James Gips.

[8] Lee Jacobson.the Project Spot.2013. Introduction to Artificial Neural Networks - Part 1. Theprojectspot. 5th December 2013 at 7:42. http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7 (Accessed 2015-1-20).

[9] Mouse (computing). Wikipedia.2015. https://en.wikipedia.org/wiki/Mouse_%28computing%29 (Accessed 2015-1-1).

[10] Mrs. Sunita Roy and Mr. Susanta Podder (2013) *Face detection and its applications* , Ph.D. scholar in the Dept. of Computer Science & Engineering, University of Calcutta, Kolkata, India. Ph.D. scholar, CMJ University, Shillong, Meghalaya-793 003, India. : www.ijreat.org .

[11] Nist Sematech.engineering statics handbook. Single Exponential Smoothing. itl.nist.gov. http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm (Accessed 2015-4-3).

[12] OpenCV. Wikipedia.2015. https://en.wikipedia.org/wiki/OpenCV (Accessed 2015-3-20).

[13] Speech recognition. Wikipedia.2015. https://en.wikipedia.org/?title=Speech_recognition (Accessed 2015-4-27).

[14] Tobii EyeX.2015. Be first to get a Tobii EyeX Dev Kit. Tobii. http://www.tobii.com/buy-eyex (Accessed 2015-3-18).

[15] Tutorials Point.2014. Digital Image Processing. Tutorialspoint. http://www.tutorialspoint.com/about/index.htm (Accessed 2015-1-4).

[16] Willow Garage.2015. OpenCV. Willowgarage. https://www.willowgarage.com/pages/software/opencv (Accessed 2015-3-21).

GLOSSARY

A

- **Arm disabled:** A person who has a physical impairment with his arm .

- **Artificial Neural Network**: is a computational model based on the structure and functions of biological neural networks. ANNs are considered nonlinear statistical data modeling tools where the complex relationships between inputs and outputs are modeled or patterns are found. ANN is also known as a neural network.

B

- **Back Propagation Learning:** Learning algorithm for modifying a feed-forward neural network which minimizes a continuous "error function" or "objective function." Back-propagation is a "gradient descent" method of training in that it uses gradient information to modify the network weights to decrease the value of the error function on subsequent tests of the inputs.

C

- **Computer vision** : is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.

D

- **Data Flow Diagram (DFD):** graphical representation of the flow of data through a system. it consists of four basic elements:
  - Data sources and destinations. Also called external entities
  - Data flows
  - Processes
  - Data stores
- **Digital Image Processing: is** a representation of a two-dimensional image as a finite set of digital values, called picture elements or pixels.
- **Disability**: any restriction or lack (resulting from an impairment) of the ability to perform an activity in the manner or within the range considered normal for a human being.

E

- **Edge :**The line where two surfaces meet

- **Edge Detection:** refers to the process of identifying and locating sharp discontinuities in an image.

F

- **Face recognition:** is a computer technology that identifies human faces in digital images
- **Feed-forward networks:** in which, the signal flow is from input to output units, strictly in a feed-forward direction.
- **Filter :**In computer programming, a filter is a program or section of code that is designed to examine each input or output request for certain qualifying criteria and then process or forward it accordingly
- **Functional Requirements:** specifies what the system should do:

  "A requirement specifies a function that a system or component must be able to perform."

G

- **greyscale digital image :**is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest.

H

- **Histogram equalization:** is a method in image processing of contrast adjustment using the image's histogram.

I

- **Image processing:** is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it

M

- **Median filter:** is a nonlinear digital filtering technique, often used to remove noise.
- **Mouse** : is a pointing device that detects two-dimensional motion relative to a surface. This motion is typically translated into the motion of a pointer on a display, which allows for fine control of a graphical user interface.

N

- **Neural Network Architectures (NNA):** consists of three types of neuron layers: input, hidden, and output layers.

- **Non Functional Requirements:** specifies how the system should behave:"A non-functional requirement is a statement of how a system must behave, it is a constraint upon the systems behavior."

P

- **Perception learning rule:** The perception is a single layer neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The training technique used is called the perception learning rule.

- **Project plan:** is a discipline for stating how to complete a project within a certain timeframe, usually with defined stages,
- **Pixel** : is a physical point in a raster image, or the smallest addressable element in an all points addressable display device; so it is the smallest controllable element of a picture represented on the screen.

S

- **speech recognition**: is the translation of spoken words into text

ملخص

Virtual mouse controller (VMC)

هو عباره عن برنامج بديل للفأره التقليديه يساعد ذوى الاختياجات الخاصه الذين يعانون من عدم القدره على تحريك ذراعهم بشكل صحيح (وذلك يرجع الى عده اسباب منهم البتر او الشلل الرعاش او عيب خلقى فى ذراعهم او لاسباب اخرى) من استخدام الكمبيوتر بشكل صحيح حيث يستطيعون تحريك مؤشر الفأره عن طريق او الوجه ليس عن طرق الفأره الحقيقيه ويمكنهم عمل نقرات عن طريق واحد من ثلاث اختيارات

١) التعرف على الاصوات حيث يقول المستخدم
بدلا من الضغط على زر الفأرة الايسر او -right - بدلا من الضغط على زر الفأرة الايمن -left

٢) محدد الوقت حيث اذا استمر مؤشر الفأره لمده ثانيه ف نفس الماكن يقوم البرنامج بعمل ضغطه شمال واذا استمر لمده ثلاث ثوانى يقوم البرنامج بعمل ضغطتين شمال

٣) التعرف على حاله العين حيث اذا اغلق المستخدم العين الشمال ذلك يعنى نقره شمال واذ اغلق المستخدم العين اليمين ذلك يعنى نقره يمين