Birzeit University

Faculty of Information Technology

Computer Systems Engineering Department

Hyper Optical Pointer extension

Prepared By

Motaz Sabri

A graduation project submitted to the Computer Systems Engineering Department in partial fulfillment of the requirements for the degree of B.Sc. in Computer Engineering

Birzeit

January- 2008

Contents:

List of Figures:

مستخلص

للعبن وظائف كثيرة، فبالاضافة الى الرؤية يمكننا الحصول على الكثير من المعلومات، من بينها معرفة شخصية الفرد و التعرف على تصرفاته من خلال حركة عينه، لذلك يستخدم الناس العين كوسيلة اتصال فيما بينهم . بالعين ايضا يمكننا تحديد تركيز الفرد واهتماماته .

مع تطور الحاسب ، أصبحت هناك حاجة الى تطوير الاتصال بين الحاسب والانسان، وذلك باستخدام حواس الانسان كمدخلات للحاسب ،ونتيجة لذلك أصبح بالامكان شق قناة بين اضخم قوتين عقليتين على هذه الارض .

ان فكرة هذا المشروع مبنية على تطبيق وظائف فارة الحاسب مثل : تحريك ا لمؤشر، وايقافه، والنقر الاحادي والمزدوج الايمن والايسر على عين الانسان،بحيث يمكن للانسان التحكم بمؤشر الحاسب وذلك بتحريك عينه في اتجاهات مختلفة امام الشاشة، كما يمكنه القيام بالنقر الايمن أوالايسر عن طريق اغلاق عينه اليمنى أو اليسرى .

يتم التطبيق باستخدام كاميرا يضعها الفرد أمامه عندما يستخدم الحاسب ،حيث تقوم هذه الكاميرا بالتقاط صور متتابعة لذلك الفرد ونقلها الى برنامج وظيفته تحليل تلك الصور وايجاد مركز العين فيها ،حيث يمثل موقع مركز العين موقع المؤشرعلى الشاشة ،واختلاف موقع ذلك المركز في صور مختلفة يمثل تحرك المؤشر على الشاشة ،ويمكن للفرد اغلاق عينه اليمنى أواليسرى كتعبير عن نقره على كائن معين على الشاشة .

Abstract:

Eyes do more than see. While the main function of the human eye is to perceive, we use it for more than sight. More data can be given by the eyes. From eyes, we can get information regarding a person's personality, attitude, current state of mind, etc. This data is used by humans for communication. One of the important pieces of information we can glean from the eyes is Focus or Attention. By mentally tracing the gaze direction of a human, we find out what draws his attention, what he is focusing on. Eyes are different from the other body parts that make up the human's sensor array. Eyes are extremely rapid

The project is simple in concept - We want to control the cursor position by eye tracking, also we want to add the functions that a computer-mouse provides like clicking and scrolling and moving cursor any where in the screen. All these functions are controlled by human eyes.

A simple Logitech Webcam is used to capture images (live video) of human face and software takes these images and makes processing on it, to find the center of the eyes and according that it will translate the eye tracking into specified functions.

ACKNOWLEDGEMENTS:

"Give me a firm place to stand, and I will move the earth." – Archimedes

We would like to thank Dr. Wasel Ghanem for his valuable support throughout the development of this project. This project really would not have been reach to an end without his guidance and patience.

Chapter 1:

Introduction

Controlling computers and other equipment through eye movement has great potential. But first it must become affordable. Eye gaze technology is one of the most exciting areas of assistive technology because it allows people with severe movement problems to control computers and environmental devices with hardly any effort. People that are severely paralyzed or afflicted with diseases such as ALS (Lou Gehrig's disease) or multiple sclerosis are unable to move or control any parts of their bodies except for their eyes. The eye is different from the other body parts that make up the human's sensor array. It is different, as through the eyes, a lot can be read with regards to a human's expressions. For example, it can be assumed that a person's attention is generally focused on where they are looking and because of this, tracking the eyes movement can be useful. Eyes are extremely rapid, Eye movements are natural, little conscious effort, Direction of gaze implicitly indicates the focus of attention. Basic eye-tracking systems consist of a camera, a computer and software to provide an interface between the human and the computer. In more systems, the camera is helped by a light source to create reflections on the eye and produce more accurate tracking. The camera tracks these reflections and uses them to move a cursor on the screen. State-of-the-art eye-tracking equipment has an extremely high resolution – it can detect tiny movements of a user's eye, scanning it at intervals of a few milliseconds. Due to the possibilities, tracking the eyes motion in real time is not a new idea. The most common method involves finding the centre of the pupil. From these two co-ordinates a distance can be easily measured.

There are other various ways to achieve tracking of the gaze and these are highlighted later in our project.

1.1 Scope of the Project

The aim of this project is to present a system that allows controlling the cursor position in a computer display through the human eye movements. The developed system uses a hardware which is a high resolution webcam which capture stream of images (live video) of a human face and transferring these images into a software which provides the processing by filtering the images and locate the center of the eyes by using a specified algorithm, after that the software translates the eye reactions such as movements and blinking into functions the computer-mouse usually do, like moving cursor, pointing it, clicking on some icon, scrolling some page, etc…

1.2 Benefits of Eye Tracking

There are many potential uses for Gaze Tracking and there are already many products on the market. However, these are generally for specialist applications.

- There is potential in the automobile industry to have gaze detection for control loops within cars. As cars are becoming increasingly computer controlled, the feasibility of using gaze detection in cars is growing. Gaze detection is useful in letting the computer know where the driver's attention is focused. Uses include predicting crashes and to safely deploy air bags. Using gaze detection is also beneficial in being able to tell if the driver is checking the mirrors or even falling asleep.
- Gaze tracking can be used to aid people who are disabled and lack the capacity to communicate in other ways. As long as a person's sight is active, communication can be achieved. Generally this involves a keyboard

displayed on a screen and the gaze detection system, once calibrated, tracks eye movement to discern the character to be chosen. Hospitals can provide an eye-aware communication program to people who have lost their ability to move and speak, either temporarily or permanently through a traumatic accident.

- Gaze detection is often used for statistical purposes. Applications include attention statistics for console operators in order to improve efficiency of programs. By tracking console operators gaze movements during exercises, software GUIs and programs can be configured to best accommodate users. This leads to an increase in efficiency and reaction times in emergencies.

A multitude of other uses in control are possible for this type of system, should the final product be flexible, non-intrusive and compact. The past has proven that flexibility is the key to most successful Eye Tracking systems. New uses are being engineered at a fast rate, allowing more flexible systems that are easily implemented to become very successful. The main pitfall of most of these systems so far, however, is the high costs associated with purchase. [1]

1.3 History of Eye Tracking

At least 50 years ago, someone consciously recognized that the direction of a person's gaze was directly related to the relative positions of the pupil and the reflection of an object off the cornea. In the late 1960's Kenneth Mason formalized the "pupil-center/corneal-reflection" method, an automated procedure for observing the eye with a camera, measuring the locations of the pupil center and corneal reflection, and calculating the direction of gaze.

In the early 1970's John Merchant and Richard Morrisette, in work sponsored by the U.S. Air Force, built a system that reduced the concept to practice. Their famous "Oculometer" used a video camera to observe the subject's eye and a computer to process the camera's image of the eye. Their image processing algorithms consisted of innovative methods to recognize the pupil of the eye and calculate its geometric center, and locate the relative position of the corneal reflection. They introduced the use of higher order polynomial equations to correct for nonlinearities in the Oculometer, and they developed root-mean-square regression methods for calibrating the equations to individual people's eyes.

In 1988 LC Technologies introduced the first PC-based eye-tracking system that interfaces easily with other equipment and computers. Dixon Cleveland developed advanced image processing algorithms for locating the pupil and corneal reflection more accurately and consistently. He also developed automatic focusing methods which maintain gaze point prediction accuracy as a person moves his head toward and away from the camera, making the system more tolerant to head motion. [2]

Chapter 2:

System Requirements

2.1 Hardware

The system hardware has been designed to minimize cost while still being able to effectively process and capture video data for detection.
The webcam is the core of hardware for the project. Choosing a cheap, yet quality webcam is imperative. A set of webcam criteria has been developed. This criterion has been used to determine which webcam is best suited to the project.

2.1.1 Webcam Selection Criteria

The webcam selection criteria focus mainly on price. This assumes that the camera quality is of sufficient standard. Lens and captured image quality is deemed an advantage.

The requirements that would be useful in the detection of gaze are:

1. High resolution:
The size of eye is small, so we need a huge number of pixels of the image for precise and accurate results, and to show the difference between two images when the eye changed its position.
2. Considerable frame rate:
The function of the camera is to take a continuous video of eye images, so we need a fast camera (good frame rate) to achieve a real time system
3. Webcam with light source (led):
This light is important to increase the brightness of the pupil which is needed in the algorithm described in chapter 4.

4. Cheap Cost:

This will allow any user to use the system without high cost, so the system will be flexible.

2.1.2 Webcam Analysis

A webcam is a basically a digital camera that is connected to a computer, usually through a USB port. The camera works together with the computer and Webcam software to capture the images. The software then uploads the pictures onto a specific web page. Depending on the frame rate of the webcam, the web page can upload periodic images, resulting in updated photos every few minutes or seconds, or continuous frames, resulting in live video streams (Wilson).
We choose two types of webcams, infra red, and CCD. We find that the following webcams is adequate for our project.

Quick Cam® Orbit AF is a CCD webcam and has the following features:

- Price: USD 129.99.
- Carl Zeiss® optics.
- Autofocus system.
- Ultra-high resolution 2-megapixel sensor with RightLight™ Technology
- Color depth: 24-bit true color
- Video capture: Up to 1600 by 1200 pixels (HD quality)
- Frame rate: Up to 30 frames per second
- Still-image capture: 8 megapixels (with software enhancement)
- Built-in microphone with RightSound™ Technology
- The camera works with Macintosh OS X, Windows 2000, ME or XP machines.[3]

Fig.2.1 Quick Cam version 1

Fig2.2 Quick Cam version 2

A4-Tech PK-333MB Night Vision is an IR Web Cam and has the following features:

• Price: USD 49.99.
• 6 Infrared LEDs turn your night into day
• Night Vision adjusts the color of the image
• to the day color even at night time
• Additional practical features such as: automatic white balance, digital zoom function, snapshot button, instant still image capture, motion tracking.
• Take 1.3 mega pixels dynamic images (interpolated)
• Built in microphone to experience clear sound and echo-free audio
• Integrated performance with the included VP-EYE programs
• No distortion and Capture any view at any angle
• Smart Clip-on design to be fixed on the LCD monitor
• Snap Shot Button for Install Still Image.[4].

Fig.2.3 A4-Tech PK IR webcam

we decided to buy the Logitech webcam which provides a color image instead of the IR webcam which provides an IR image because it is found that there is a library operates in C++ environment called OPEN-CV library which gives us 'laplace' function that transform the color image into image that look like a night vision image which is like what IR webcam do, so we change the image in the software rather than the hardware, this will help us in consuming the excellent features provided by the Logitech such as The face focus system; it has a face tracking software, the webcam stays focused on the user's face, freeing up the person from having to manually adjust the camera, this helps us greatly, human should not stay focus on the camera to take his eye image it will search for his face to capture an image, this will free the movement of the user. It also provides an auto-zoom features let the camera zoom in up to three times the normal viewing size. It has two shapes one with a 9-inch stand, which lets users elevate the camera to eye-level, and other without. The first version helps users with comfortable stand in front of the compute

2.1.3 Led and its Effect:

Light source has a great effect on the image the software capture, when there is a good light source the eye detection become easier, because we depend in primary way on the brightness of the gaze to detect it. Our algorithm does not work in darkness, so to increase the performance we will add a small light source (led).

2.1.4 CPU and RAM:

To achieve a real time system our project will function well in a PC with at least 2 GHz processor and a RAM with 256 MB

2.2 Software

.

2.2.1 Programming language and OS:

We will use C++ language for our project for the following features:

- Availability. C++ can be used on a variety of computers ranging from personal computers to supercomputers.
- Portability. C++ is in the last stages of standardization, and this process has been closely followed by compiler vendors, ensuring quick availability of a common standard language.
- Speed. Even with the speed of modern computers there are problems for which efficiency of language is of paramount importance. C++ inherits the speed of C.
- Generality. Science and engineering now extends beyond computation and into realms of graphics and robotics. The C++ language is not only suited

for formulas and computations (FORTRAN's strong suit), it can be used effectively in data base and business applications (COBOL's strong suit).

- Reusability. Clearly it would be an advantage to use previously written code if it can be inserted into one's current project. The capability of easy re-use and modification of existing code to fit new problems is an important feature of the C++ language.
- Object Orientation. As computers have become larger and faster, they have been programmed to solve larger and more complex problems. Object Oriented Programming (OOP) is a strategy for simplifying the programming task, and is supported by the language features of C++.

The project will operate on Microsoft Windows Operating System. [5]

2.2.2 Intel Open-CV:

We search for a robust, stable and reliable solution to help us in image processing and eye detection issues, we found out that Intel had developed an open library for image processing which called open source computer vision (Open-CV).

The Open-CV Library is mainly aimed at real time computer vision. Some example areas would be Human-Computer Interaction (HCI); Object Identification, Segmentation, and Recognition; Face Recognition; Gesture Recognition; Motion Tracking, Ego Motion, and Motion Understanding; Structure from Motion (SFM); and Mobile Robotics.
The Open-CV Library software package supports many functions whose performance can be significantly enhanced on the Intel® architecture (IA), particularly...

The Open-CV Library is a collection of low-overhead, high-performance operations performed on images.

The Open-CV implements a wide variety of tools for image interpretation. It is compatible with Intel® Image Processing Library (IPL) that implements low-level operations on digital images. In spite of primitives such as binarization, filtering, image statistics, pyramids, Open-CV is mostly a high-level library implementing algorithms for calibration techniques (Camera Calibration), feature detection (Feature) and tracking (Optical Flow), shape analysis (Geometry, Contour Processing), motion analysis (Motion Templates, Estimators), 3D reconstruction (View Morphing), object segmentation and recognition (Histogram, Embedded Hidden Markov Models, Eigen Objects).

The essential feature of the library along with functionality and quality is performance.

The algorithms are based on highly flexible data structures (Dynamic Data Structures) coupled with IPL data structures; more than a half of the functions have been assembler-optimized taking advantage of Intel® Architecture (Pentium® MMX□, Pentium® Pro, Pentium® III, Pentium® 4).

The Open-CV Library is a way of establishing an open source vision community that will make better use of up-to-date opportunities to apply computer vision in the growing PC environment. The software provides a set of image processing functions, as well as image and pattern analysis functions. The functions are optimized for Intel® architecture processors, and are particularly effective at taking advantage of MMX technology.

The Open-CV Library has platform-independent interface and supplied with whole C sources. Open-CV is open. [6]

Chapter 3

Theory

In this chapter we are talking in details about the theoretical part in our project, and that include talking about eye specification, image processing, gaze detection, gaze tracking and finally blink detection.

3.1 Eye Specifications:

In this part we are going to talk about the eye, since the eye is the major element in our project, the main challenges in our project is to determine the location of the eye in a frame then go deeper inside it to locate the gaze, so its worth to talk about the eye structure and the effect of the light on it.

3.1.1 Eye Structure:

A conceptual technique for the establishment of our project would be to perform a physiological analysis of the eye, the nerve paths to the brain, and those parts of the brain involved in visual perception. The very back of the eye is lined with a layer called the retina which acts very much like the film of the camera. The retina is a membrane containing photoreceptor nerve cells that lines the inside back wall of the eye. The photoreceptor nerve cells of the retina change the light rays entering through the pupil into electrical impulses and transmit them through the optic nerve to the visual part of the brain where an image is perceived [7].

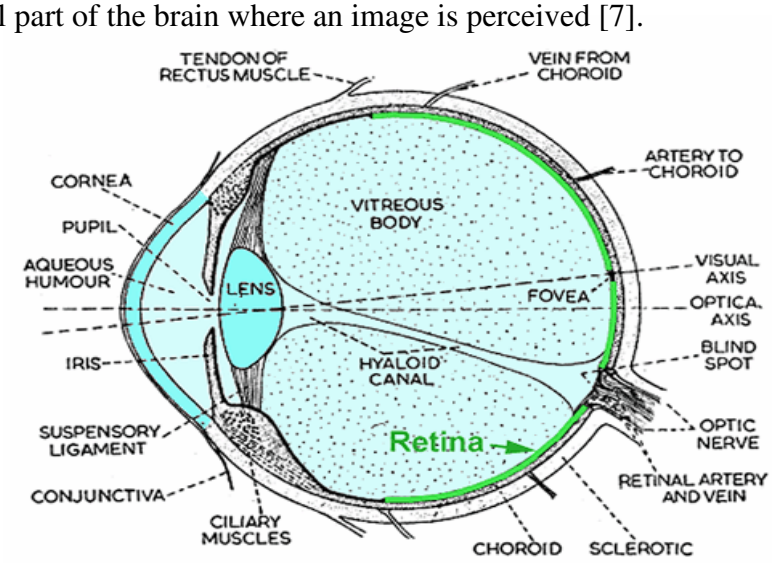

Fig. 3.1.Eye cross section.

Figure 3.1.1 shows the horizontal cross section of a human eyeball. The front of the eye is covered by a transparent surface called the cornea. The remaining outer cover,

called the sclera, is composed of a fibrous coat that surrounds the choroid, a layer containing the blood capillaries. Inside the choroid is the retina, which is composed of two types of receptors: rods and cones. Nerves connecting to the retina leave the eyeball through the optic nerve bundle. Light entering the cornea is focused on the retina surface by a lens that changes shape under muscular control to perform proper focusing of near and distant objects. An iris acts as a diaphram to control the amount of light entering the eye. The position of eye over the head decides the field of vision. There are two types of fields of vision; the field of view of an individual eye, and overlapped portion of field of view (binocular field). When the eyes are on either side of the head, view is almost panoramic, but there is a loss of stereoscopic depth perception. Humans have a total field of view between 160 to 208 degrees [8].

The most important property of our eye movement system is that it can move the eye from one point of gaze to a new location of gaze very rapidly. These saccadic eye movements are among the fastest movements the body can make; the eyes can rotate at over 500 deg/sec, and over one hundred thousand of these saccades are done during the day. These rapid eye movements are accomplished by a set of six muscles attached to the outside of each eye. They are arranged in three pairs of agonist antagonist pairs; one pair rotates the eye horizontally (left - right), the second rotates the eye vertically (up - down), the third allows 'cyclotorsion,' or rotation about the line of sight.

We design our algorithm to deal with all this variables in the eye movements; our algorithm will have the capable of determining the gaze in a very high accuracy and in these cases too:

- Saccadic eye movements while face is stationary
- Face is moving and eyes are making saccadic movements
- Video images which do not only contain human face images with scale and Rotation invariance property.

3.1.2 Light Effect On Eyes:

Light is known to be a form of electromagnetic radiation lying in a relatively narrow region of the electromagnetic spectrum over a wavelength band of about 350 to 780 nanometers (nm). A physical light source may be characterized by the rate of radiant energy (radiant intensity) that it emits at a particular spectral wavelength [9]. Light entering the human visual system originates either from a self-luminous source or

from light reflected from some object or from light transmitted through some translucent object. Let E (λ) represent the spectral energy distribution of light emitted from some primary light source, and also let t (λ) and r (λ) denote the wavelength-dependent transmissivity and reflectivity, respectively, of an object. Then, for a transmissive object, the observed light spectral energy distribution is

$$C(\lambda) = t(\lambda)\, E(\lambda) \qquad 3.1$$

Figure 3.2 show a small diagram that explains this equation. According to this equation 'C' is a function of ' λ ' and it represents the observed light spectral energy distribution for a transmissive object, in our case the eye is a transmissive object.

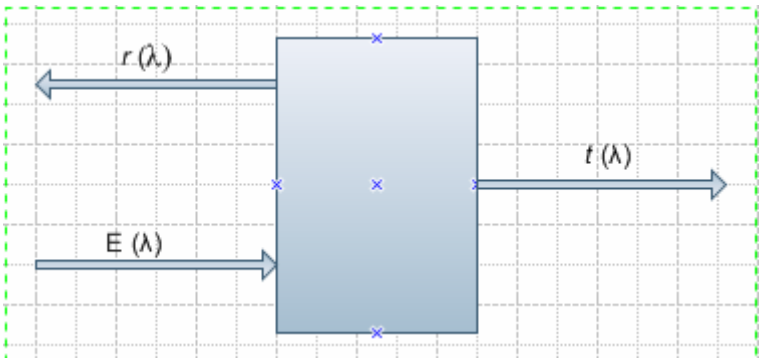

Fig 3.2. Object transmissivity t (λ) and reflectivity r (λ)

Equation 3.2 represents the observed light energy distribution ' C (λ) ' for a reflective object

$$C(\lambda) = r(\lambda)\, E(\lambda) \qquad 3.2$$

3.2 Image Processing:

Images are built up of pixels that contain color information and are aligned with the Cartesian coordinate system. The zero point is found at the top-left corner of the image the image's width is represented by the variable X, the image's height with the variable Y. When working with pixel-based images, one can work with black-and-white (bitmap), grayscale and colored images. You have to understand that the pixel information between these image modes is different. This also plays a part in the image file's size and memory size needed. . Image processing is the operation of taking one array of pixels as input and producing another array of pixels as output which represent an improvement to the original array. In this section we will talk about image processing in frequency domain, Laplace filtering, grayscale filtering, and inversion filtering.

3.2.1 Frequency Domain Image Processing:

The frequency domain is a space in which each image value at image position F represents the amount that the intensity values in image I vary over a specific distance related to F. In the frequency domain, changes in image position correspond to changes in the spatial frequency, are changing in the spatial domain image I.

Spatial frequency refers to the periodicity with which the image intensity values change. Image features with high spatial frequency (such as edges) are those that change greatly in intensity over short image distances. Most of the processing in our work happens in frequency domain, so in order to convert images from spatial domain to frequency domain we will use Fourier transformation. Fourier analysis is used in image processing in much the same way as with one-dimensional signals. However, images do not have their information encoded in the frequency domain, making the techniques much less useful. Taking the Fourier transform of an image converts the straightforward information in the spatial domain into a scrambled form in the frequency domain [10].

3.2.1.1 Fourier Transform:

The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image. As we are only concerned with digital images, we will restrict this discussion to the Discrete Fourier Transform (DFT). The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The number of frequencies corresponds to the number of pixels in the spatial domain image or a square image of size N×N, the spatial domain image is first transformed into an intermediate image

using N one-dimensional Fourier Transforms [10]. This intermediate image is then transformed into the final image, again using N one-dimensional Fourier Transforms. Expressing the two-dimensional Fourier Transform in terms of a series of 2N one-dimensional transforms decreases the number of required computations. The Fourier Transform produces a complex number valued output image which can be displayed with two images, either with the real and imaginary part or with magnitude and phase. In image processing, often only the magnitude of the Fourier Transform is displayed, as it contains most of the information of the geometric structure of the spatial domain image. However, if we want to re-transform the Fourier image into the correct spatial domain after some processing in the frequency domain, we must make sure to preserve both magnitude and phase of the Fourier image [11].

The Fourier domain image has a much greater range than the image in the spatial domain. Hence, to be sufficiently accurate, its values are usually calculated and stored in float values.

3.2.2 Laplace Filtering:

The first step in our work in image processing is to perform Laplacian filter to every image we capture, The Laplacian filter are a set of three by three kernels which approximate the Laplacian operator, where a Laplacian operator is defined as the sum of the partial second derivatives in x and y direction simultaneously. The Laplace operator is presented as an isotropic edge detector for images with low noise. The Laplacian of an image highlights regions of rapid intensity change [11]. The input for Laplace filter is list of RGB images to be convolved and the output is a list of images that look like a night vision images, the number of output images must equal the number of input images. If the input image name equals the output image name the convolved image will replace the input image. The Laplacian $L(x, y)$ of an image with pixel intensity values $I(x, y)$ is given by:

$$L(x, y) = d^2 I/ dx^2 + d^2 I/ d y^2. \quad 3.3$$

As we know the input image is represented as a set of pixels "discrete pixels", we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small

kernels are shown in Figure 3.3. Using one of these kernels, the Laplacian can be calculated using standard convolution methods. The Laplacian filters are high-pass filters which act as a local edge detector.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Fig. 3.3. Kernel matrices

At first and for each image we perform horizontal Laplace filter using the appropriate matrix as you can see in the figure

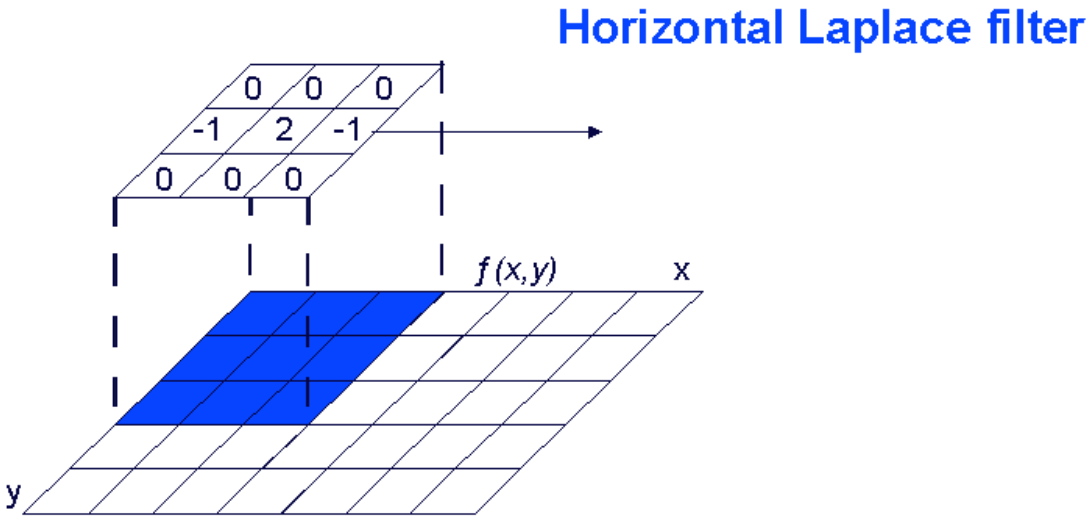

Fig. 3.4. Horizontal Laplace filter

Then we perform vertical Laplace filter as u can see in the figure

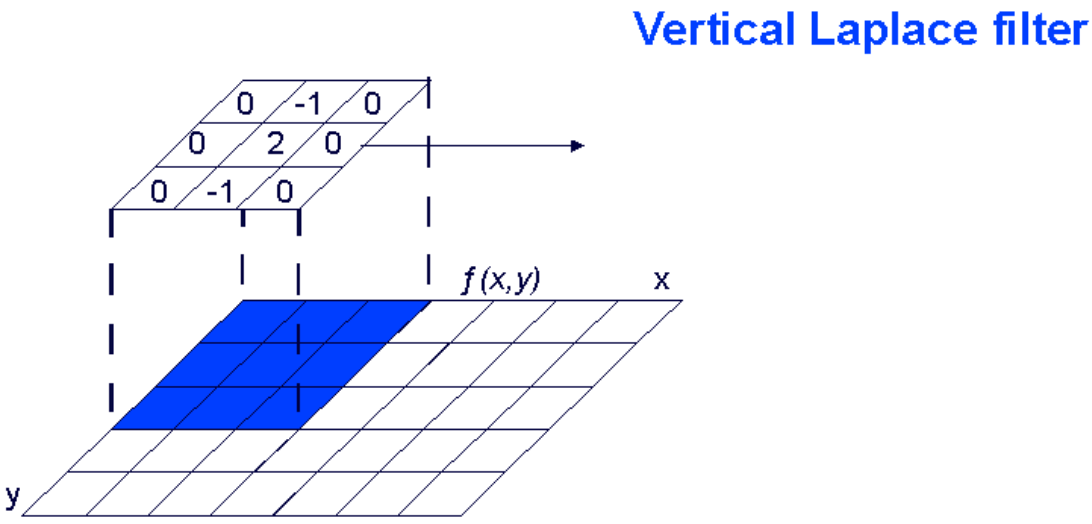

Fig 3.5.Virtical Laplace filter

The sum of the two matrices that appear in Figure 3.4 and Figure 3.5 will give you the first matrix that appears in figure 3.3.

3.2.2.1 Kernel Matrix:

A kernel is a smallish matrix of numbers that is used in image convolutions. Differently sized kernels containing different patterns of numbers give rise to different results under convolution. For instance, Figure 3.3. shows 3×3 kernels that implements a Laplacian filter.

3.2.2.2 Convolution:

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of multiplying together two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values. In an image processing context, in our case one of the input arrays is normally just a RGB image. The second array is usually much smaller, and is also three-dimensional and is known as the kernel. There are tow methods to perform convolution:

First method is based in multiplying the image with a specific kernel matrix. Figure 3.6 show an image of size 9*9 and a kernel matrix of size 3*3. The

convolution is performed by sliding the kernel over the image, generally starting at the top left corner, so as to move the kernel through all the positions where the kernel fits entirely within the boundaries of the image. Each kernel position corresponds to a single output pixel, the value of which is calculated by multiplying together the kernel value and the underlying image pixel value for each of the cells in the kernel, and then adding all these numbers together.

I1	I2	I3	I4	I5	I6	I7	I8	I9
I10	I11	I12	I13	I14	I15	I16	I17	I18
I19	I20	I21	I22	I23	I24	I25	I26	I27
I28	I29	I30	I31	I32	I33	I34	I35	I36
I37	I38	I39	I40	I41	I42	I43	I44	I45
I46	I47	I48	I49	I50	I51	I52	I53	I54
I55	I56	I57	I58	I59	I60	I61	I62	I63
I64	I65	I66	I67	I68	I69	I70	I71	I72
I73	I74	I75	I76	I77	I78	I79	I80	I81

K1	K2	K3
K4	K5	K6
K7	K8	K9

Figure 3.6 Image of size 9*9 and kernel matrix

So, in our example, the value of the bottom right pixel in the output image will be given by:

O61 = I61* K1 + I62* K2 + I63 *K3 + I70* K4 + I71* K5 + I72* K6 + I79* K7 + I80*K8 + I81* K9

If the image has M rows and N columns, and the kernel has m rows and n columns, then the size of the output image will have "$M - m + 1$" rows, and "$N - n + 1$" columns.

The second method to perform convolution happened in different way, the kernel can only moved to position where it fits entirely within the image. These implementations typically slide the kernel to all positions where just the top left corner of the kernel is within the image. Therefore the kernel overlaps the image on the bottom and right edges. One advantage of this approach is that the output image is the same size as the input image. Unfortunately, in order to calculate the output pixel values for the bottom and right edges of the

image, it is necessary to invent input pixel values for places where the kernel extends off the end of the image. Typically pixel values of zero are chosen for regions outside the true image, but this can often distort the output image at these places. In our work we are going to use the second way of convolution since the information that we want is located in the middle of the image "the eyes" and also we can fix this problem by Removing n - 1 pixels from the right hand side and *m* - 1 pixels from the bottom will fix things.

3.2.3 Grayscale:

A grayscale (or gray level) image is simply one in which the only colors are shades of gray. The reason for differentiating such images from any other sort of color image is that less information needs to be provided for each pixel. In fact a gray color is one in which the red, green and blue components all have equal intensity in RGB space, and so it is only necessary to specify a single intensity value for each pixel, as opposed to the three intensities needed to specify each pixel in a full color image. Often, the grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. If the levels are evenly spaced then the difference between successive gray levels is significantly better than the gray level resolving power of the human eye. In our design the grayscale stage come directly after performing a Laplace filter on the captured image, we will convert the resulted image from the Laplace stage (night vision image) to grayscale image were all the colors of pixels in the resulted image will be between the ranges of 0-255. This will help us to remove the un wanted data from the image and also help us to remove the noise in the images that result from the Laplace filter.

3.2.4 **Inversion**:

The main idea behind this stage is to invert the color in an image. In our case the image will be in grayscale, so The Invert stage is used to invert color values in pixels in grayscale. Invert simply switches 0 to 255 and 255 to 0 and likewise switches mirror-fashion all values in between. In a simple black and white image, using 0 for black and 255 for white, a near-black pixel value of 5 will be converted to 250, or near-white. The result is a photographic negative image. For

RGB color images or any other image with more than one channel this process is applied to each channel. If a pixel has a high blue value and low red and green values, the blue value will end up low and the red and green values will be high to result in a yellow tone.

3.3 Gaze Detection:

Gaze detection systems can provide a good interaction interface for computers a webcam is used to captures a face image and analyzes it to estimate the gaze location in that image without going to face detection or to eye detection; we go directly to the gaze. In order to determine the gaze , the captured image go through several stages starting from laplace filtering then transform the image into gray scale and before scanning the image we take the inverted copy of this image. The final step is to scan the inverted image to find the gaze, our aim behind the search is to find the pixel that have a color similar to the color of the gaze, the gaze color will be between 0-10 or 30-55 or 86 or 88, every pixel has a color similar or between this ranges will be a strong candidate to be the gaze. Each image may contain several pixel that have this property, so in order To distinguish true gazes from false candidates, we assume such conditions as blinks occurring in both eyes at the same time or blinks occurring in one eye, the distance between the eyes being within a certain range , this range is
[175 - 300] where these numbers are given in pixels. knowing the location of a user’s gaze may help a computer to interpret the user’s request and possibly enable a computer to ascertain some cognitive states of the user, in our project determining the gaze location will help use to determine where the user are looking and according to this we can control and move the cursor in the specific location that the user want,

3.4 Gaze Tracking:

The main goal behind tracking the gaze is to know the current location of the gaze every time it changes, in order to do that we have a great procedure to do that. Our procedure doesn't require a template matching to match the current location of the gaze with the next one. When we detect the gaze the result will be a location represent by a pair of X and Y this pair is going to scale and translate to location on the computer screen. So every time we detect the gaze, the result

will be a new location of the gaze, so we perform a non direct gaze tracking, in the same step we detect and track the eye. This producer is better than saving a copy of a pervious image of the eye, or by comparing the current image with a pre define eye templates which will consume both the memory and time , storing a copy of a previous image will consume the memory specially if we capture many frames in one second , also performing compare operation between images will consume a lot of operation and this will effect the performance of our work , as I said before there is no need to do all of this we go direct to the gaze and we keep tracking it location every time we detect it.

3.5 Blink Detection :

A human must periodically blink to keep his eyes moist. Blinking is involuntary and fast. Most people do not notice when the blink. However, detecting a blinking pattern in an image sequence is an easy and reliable means to detect the presence of a face. Blinking provides a space-time signal which is easily detected and unique to faces. The fact that both eyes blink together provides a redundancy which permits blinking to be discriminated from other motions in the scene. The fact that the eyes are symmetrically positioned with a fixed separation provides a means to normalize the size and orientation of the head. We have built a simple blink detector which works as follow: every time we capture an image we detect tow gaze , we always expect tow gaze on the same line separated with a specific distance, the first configuration of our work is to determine the location of this tow gaze, after doing this we go on , if we find one gaze we will know that the other one is hidden since a blink has been occurred , we can also determine which eye blink and that by using the existence gaze , and relative to this gaze we can determine that the left of the right eye is has blink.

Chapter 4

4.1. Design:

In this chapter we are going to talk about the system design, work flow, adopted algorithm and the design output.

The design process in our project is divided into 3 main stages:

1. Receiving input from webcam: our system uses the webcam as an input device to perform this task we will use the Intel computer vision system which offers a suitable environment for live frame processing under windows and Linux operating systems.
2. Processing frame to get gaze coordinates: The goal of this stage is to have accurate gaze movement detection using the minimum computational power in order to ensure a high performance, to perform this task we had implemented an algorithm of our own, this algorithm is based on frequency image processing, it results from studying present face and eye detection algorithms and analyzing there concepts, advantages over other algorithms and there drawbacks.
3. Mapping the coordinates and the main mouse operation into the computer system, during this step we will translate the user actions (e.g. gaze movement or blink) into effects at the computer system using the minimum possible processing power.

After the system had been divided into 3 parts we will talk about each part alone knowing that each stage output is the input of the next stage.

4.1.1 Receiving The Input:

At this stage we will capture live frames from the webcam, to receive a proper frames we need extra lights from the front side projected to the eye, sure the light shouldn't be so bright because this may hurt the user eyes, so we will use a light emitting diode, our initial testing shows that the best location for the led to be at the top of the screen and the camera should be at the bottom of the screen, frame capture

is performed by Intel computer vision system which will offer a suitable environment for live frame processing.

When input received it will be processed as an object which can be converted into 2 dimensional array that can be processed by the next stage, we are able to treat a number of frames per second according to the type of the webcam, but having a high number of frames per second may cause system slowdown, so we prefer to keep the capture rate high enough to detect accurate values also it should be low enough to ensure high performance (up to 15 -20 frames per second).

4.1.2 Processing The Frame:

This stage is the core of our system, we had divided it into 5 stages, and each stage will forward its output to the next one.

4.1.2.1: Laplacican transformation:

The first stage of our system is the Laplace transformation of the input, this will convert the frame into a sort of night vision view that has many white pixels including the gaze, and an explanation of this behavior is mentioned at the theory chapter.

A sample image from the Laplace transformation is shown at figure 4-1:

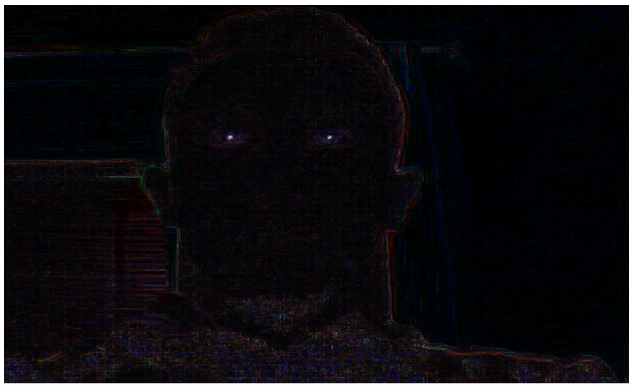

Fig.4.1: Laplacian transformation of the captured frame

As we can see from the image shown above, the gaze of the eye is clear enough to be considered as a measurement of movement, unfortunately there is a lot of noise at each frame we are going to capture due to the surrounding environment so we had to deal with this problem considering the computational cost.

4.1.2.2. Gray Scaling:

The third stage of frame processing is gray scaling, through many tests we had maintained a solution for the noise problem, by applying gray scaling to the captured frames we could remove many of the interfering pixels that has a close pixel colors to the gaze, this is right because the grayscale images has a range of colors that varies between 0-255 thus many of the pixels colors will be approximated causing lose in frame data especially high contrast boundaries, the output of the grayscale process is shown at figure 4-2:

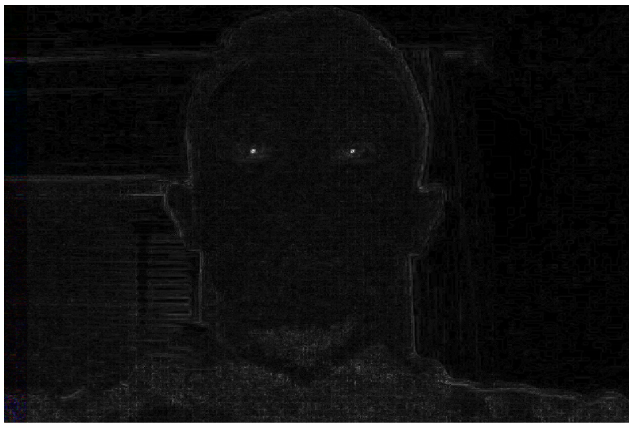

Fig.4.2: Gray scaling of the captured frame.

4.1.2.3. Inversion:

The third stage of frame processing is called inversion, the inversion function will do another approximation to the image pixels causing another lose in frame data, such a lose will convert the user gazes into black and approximately the entire background into white as shown at figure 4-3, although in some cases there will be extra noise due to the surrounding environment, so we will send the stage 3 processed frame to the next 2 filter.

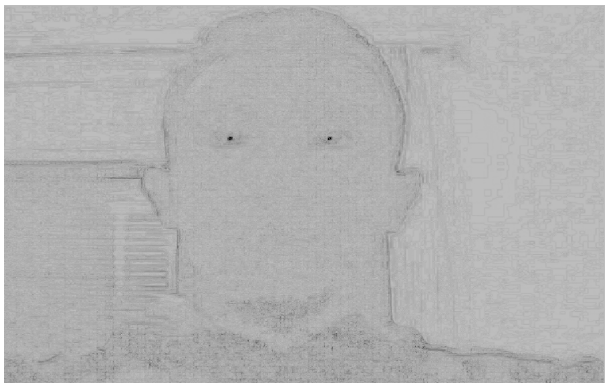

Fig.4.2: Color inversion of the captured frame

4.1.2.4. Color Filtration:

The fourth stage of frame processing is color filtration, By studying the structure of the eye we had observed that the eye will reflect the light in different levels according to light projection and reflection angles, we had analyzed the color of the gaze after Laplace transformation, gray scaling, inversion, and found out that it will be within the following ranges:

If the pixel color is between 0 and 10 then it could be a human gaze.

If the pixel color is between 30 and 55 then it could be a human gaze.

If the pixel color is 86 then it could be a human gaze.

If the pixel color is 88 then it could be a human gaze.

The values above results from many eye samples that includes variety in colors, size and projection angle, after detecting these pixels in a frame we will collect there coordinates and store them, although this stage filters many noise at the background, but the frame still have some unwanted reflections that we are going to filter them out through the next stage.

4.1.2.5. Distance Filtration:

The fourth stage of frame processing is Distance filtration, the distance filter is based on a smart idea that the distance between the 2 eyes gazes is usually constant, so we will scan the collected coordinates that we had obtained from previous filter and calculate the distance between all concentrated regions, this calculations are not based on x-coordinates or y-coordinates separately but it is based the relation that determines the distance between 2 points within a 2 dimensional region, by testing many face samples we had found that the distance between 2 gazes cant be out of bounds of the interval [175 - 300] where these numbers are given in pixels.

The output of stage five is a list of x and y coordinates per frame that represents the location of the gaze within the eye, these coordinates have an indirect relation with the place where the user is looking at, we will describe this relation later through this chapter.

4.1.3. Detection Gaze Movement:

Gaze location variation can be accurately discovered because we are using a very small parameter to monitor this movement; this parameter can be detected by light reflection over the eye, this will support the mapping software (described later) with accurate input with a form of coordinates obtained through the gaze detection stage described previously.

4.1.4. Detection of Main Operations:

Our system will be capable of handling the main mouse operation without the need of the mouse hardware; it only requires reactions from the user eyes these operations are:

- Left click: this can be handled by blinking the left eye for a 1 second, once a blink is detected a counter is initiated and when it reaches one second it will send a request to the operating system to perform a left click at the place where the "active" eye is looking at. The one second interval is long enough to distinguish the difference between normal blink and the click.
- Right click is performed with the same technique but with a blink in the right eye.
- Double click is performed using the same technique but the interval of closing the left eye is increased to 2 seconds to distinguish between single click and double click.

  Through the 3 previous operations the other "active" eye should be kept at the location where the click is wanted to be.
- The last operation that we intend to implement is the mouse scrolling, this can be done be closing the right eye for 2 seconds and move the left eye up and down relative to the user need.

Figure 4-4 shows the flowchart of the process of detecting gaze movement and main operations occurrences.

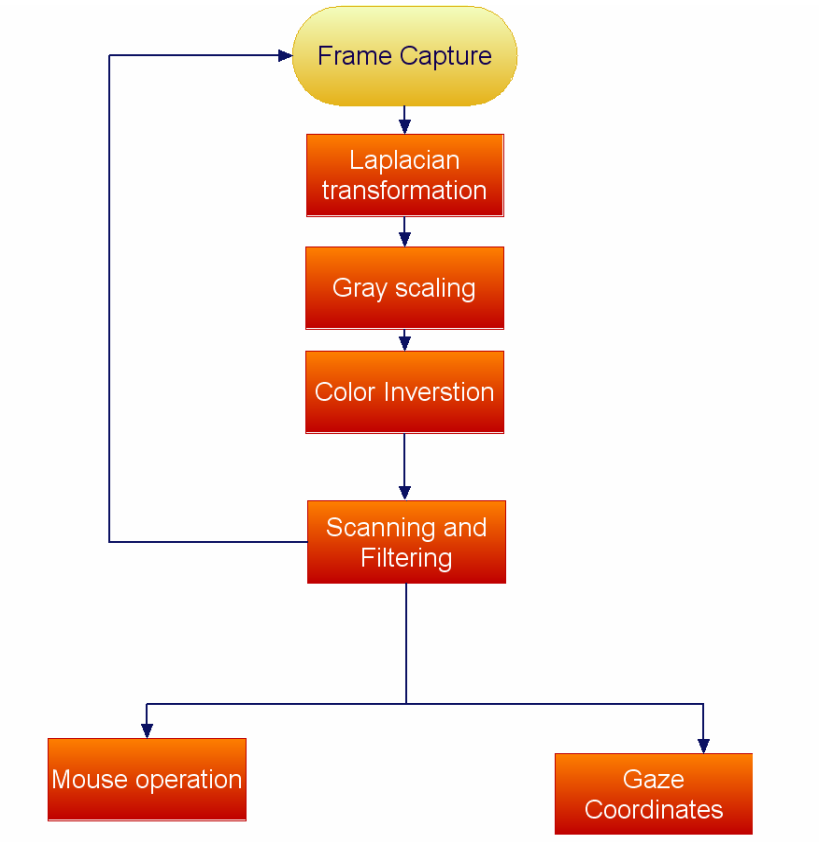

Fig. 4.4: Live frame processing stages flow chart

4.1.5. Mapping Coordinates:

4.1.5.1. The Mapping Process:

We now have a list of x and y coordinates that will be used to map user gaze movement into the screen, the coordinates of the gaze can't be directly mapped to the screen due to the nature of the eye in which it will cover many spots at the screen without focusing at a certain region so we need a sort of fixed point to map movements according to the distance from it.

In order to do this task we will use a set of reference points at the screen to move the courser according to the distance from the present location to the reference point, each time the user starts using our system, he\she will be prompt to look for the specified locations of screen (for this moment we suggest looking at the 4 corners of the screen). The process of setting the reference points is managed by a software guidance program that will be developed by us.

For example take the frame at figure 4-5, this frame has the 4 constant values that are common with this runtime, the number that varies from frame to frame is the point (140,100), once the program receives this coordinate it will check to which corner it is closer, this is done by calculating the distance form all the corners and the received coordinate, then the smallest value distance implies that it is the closest.

Fig.4.5: mapping calculated coordinates to the monitor dimensions.

Once the closest corner is discovered a direct subtraction is performed between the specified corner and the present coordinate, the output of this operation is 3 numbers:

1. The closest corner ID: if the closest corner is the top left then " closest corner" ID would be 1, if it was the top right then the ID would be 2, if it was the bottom left then the ID would be 3 other wise ID would be 4.
2. The X coordinate difference: it is the amount of shift from the corner by mean of X direction movement multiplied by the factor that can be calculated by multiplying the ration between X-length of the frame captured above that is (1123-130) with the X-length of the screen resolution (e.g. 600), the calculated ratio means how the captured frame is related to the real screen size.
3. The Y coordinate difference: it is the amount of shift from the corner by mean of Y direction movement multiplied by the factor that can be calculated by multiplying the ration between Y-length of the frame captured above that is (933 - 87) with the Y-length of the screen resolution (e.g. 800).

These values are passed to the function that will map them to the screen; the mapping process will check whether the corner is 1, 2, 3 or 4, treating each scenario as a separate case, by that we mean if the resolution of the screen is N * M , the x-coordination is X and the y-coordination is Y then:

- If the corner was 1 then the movment is set to be (0 +X, 0+Y).
- If the corner was 2 then the movment is set to be (N-X, 0+Y).
- If the corner was 3 then the movment is set to be (0 +X, M-Y).
- If the corner was 4 then the movment is set to be (N-X, M-Y).

The mapping function is an assembly written code that will convert the resulted numbers above into cursor movements at the screen.

4.1.5.2 Head Movement Treatment:

The reference point is a critical node, losing it will result in an inaccurate readings, the losing of the reference point can occur due to the change of user position (e.g. adjusting the seat or random head movement), slight change of gaze position will affect the accuracy in a critical way, we had developed a checking techniques that will validate the present reference point:

1. if the user change his/her location then the gaze coordinates related to the present frame will vary in a huge amount (25 pixels in the X direction and 5 pixels in the Y direction) so if the gaze coordinates had varied with that amount or more then the reference point must be adjusted.
2. Another auxiliary test will ensure more reliability is based on the idea that the gaze coordinates will never exceeds the values of the corners, and if a violation for this rule occurred then adjustment the reference point is required.

All the validity checking above will result in rapid and sudden variation in the gaze coordinates, this variation will ease the detection process also it will ease the solution, the amount of variation can be calculated by subtracting the gaze locations among a set of consecutive frames (e.g. 5 frames), once a rapid change is detected (more than 25 pixel at the X-coordinates and more than 5 pixels at the Y-coordinate) a calibration for the fixed reference points is performed by adding the values of the rapid change to the fixed points (rapid change in X-coordinate is added to the X-pair of the reference point and the Y-coordinate is added to the Y-pair of the reference

point) causing shift to the main reference point's square, this modification will keep the accuracy and precision level acceptable offering good aiming results.

4.1.6. Translating Main Operations:

The process of translating operation is simpler than translating coordinates: When a single click is detected (blink for 1 second) then the "active" eye coordinate is stored and those 2 parameters (click type and coordinates of active eye) are passed to the assembly code that will translate these actions into events at the screen.
When a double click is detected (blink for 2 second at the left eye) then the right eye coordinate is set and those 2 parameters (click type and coordinates of right eye) are passed to the assembly code that will translate these actions into events at the screen.
When a scroll is detected (blink for 2 second at the right eye) then the left eye coordinates is monitored whether it is upward or downward, these data are passed to the assembly code that will translate these actions into events at the screen.
Those assembly codes are easily implemented and will take the coordinates as parameters and will perform the specified task.

4.1.7. Design Output:

Our system will be based on a stable environment supported by Intel, allowing us to perform many actions over the captured frame; we had divided our system into stages that will result in a system with:

- Accurate readings in which we will get the coordinates of gaze moving pixels that are very small related to the entire region therefore more accurate.
- Good speed while the system passes through functions that are considered to be a light weight functions relative to other techniques that uses a heavy methodologies that don’t focus primarily at the high speed property.
- Errors will be minimized through 2 filtration process; these filtrations are enough to obtain the gaze coordinates and removing the majority of surrounding noise.
- Performing many mouse main operations will be implemented while taking care of any event that may influence one of the operations (e.g. involuntary blink).

Figure 4-6 shows the general design for system from the moment of set up to cursor response to the user request.

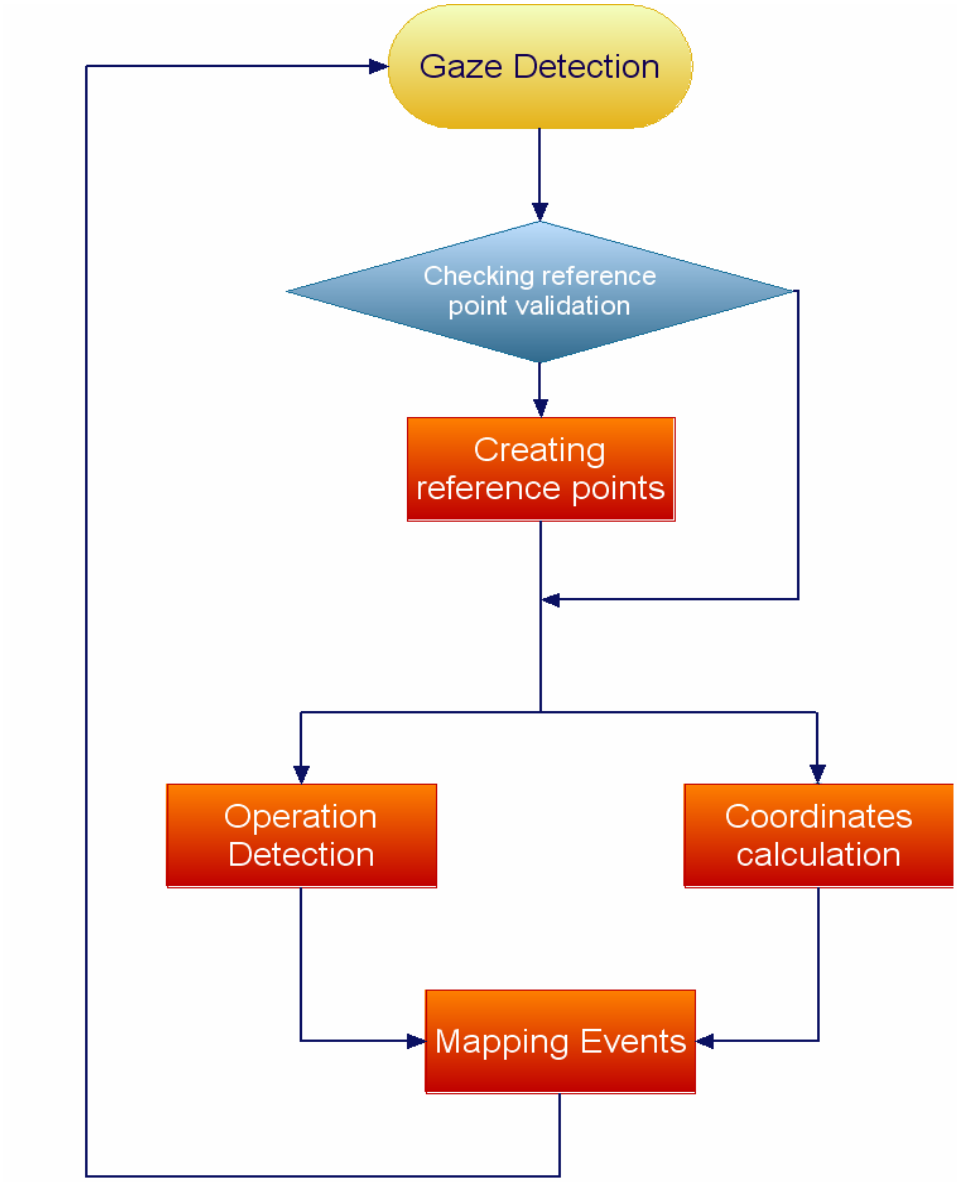

Fig.4.5: General design flow chart.

Chapter 5

Results:

5.1. Speed:

One of our main objectives through this project is to ensure high processing speed while processing user input, by analyzing our system we can note that the only part that we can modify its timing and performance is the Frame processing part, other parts like mapping and frame capturing are un-upgradeable, figure 5-1 shows speeds of execution for each stage in the detection process that we used to determine gaze location:

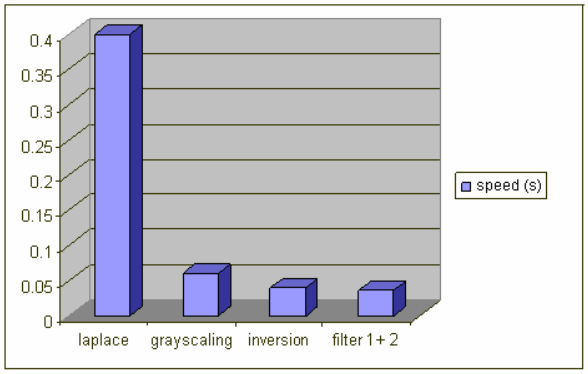

Fig.5.1: Time consumption over system stages.

The calculations are based on measurement that we did of our own and obtained from Intel computer vision websites, tested over real images "not frames" knowing that accessing an image is much slower than accessing a frame because it is a memory access while frame requires accessing registers which is much faster, therefore the results above is considered as maximums and cant be exceeded per frame.

5.2 Accuracy:

Accuracy is another objective that we aim to achieve, to do so we avoided the usage of skin filters that detects eyes only (not the gaze) and may cause errors when the surrounding environment includes other people, while we are capturing smallest part of the eye rather than capturing the entire eye we will ensure that any slight

movement will be detected providing our mapping functionality with an accurate readings, this reading also may be affected with the following factors:

- Light level from light emitting diodes, the diode should emit light using its maximum power, while extra emitting will cause bad effects at the user eyes so we should adjust the LED's emitting power to a level that is not high in order not to hurt the user eyes and not low to in order to ensure good accuracy.
- Most metallic parts should be removed from the scene as possible because it reflects the same amount of light that the human eyes reflect.
- The speed of eye movement affects the accuracy to a high degree; this is because we are processing only 15-20 frames per second at maximum levels in order to avoid extra overhead, so rapid eye movement (e.g. right, left then right) may not be detected, in order to ensure accuracy and efficiency we had set the frame rate to this optimum level.

The accuracy and repeatability directly impacted on the application of the system design. Hence, the effects listed above were minimized.
Finding the blink was straightforward, due to the use of light reflection. Consequently x, y coordinates will be quite accurate.

5.3 Comparison between algorithms of gaze detection stages:

Usually the detection of eye/gaze movement algorithms that is based on light reflection pass through common stages which may not be within the same order from algorithm to another, these stages are:

- Averaging and Edge Detection: this stage includes transforming the frame into

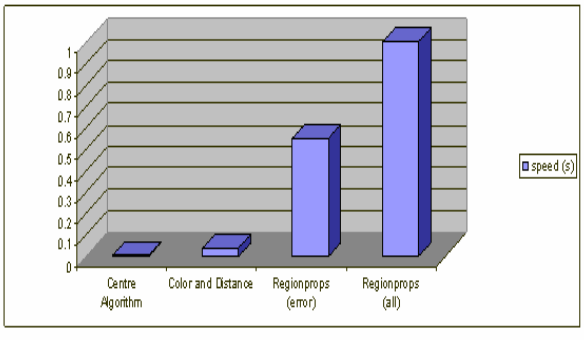

Fig5.2: Averaging and edge detection comparisons

another form, this transformation will cause sort of approximation that make the eye clearer, Edge Detection is used to find the edge of the eye (bigger than

the pupil). Figure 5-2 shows some algorithms that perform this step with there timing in seconds.

- Pupil Detection also known as the gaze detection, it is the aim of all algorithms to detect the gaze because it is the smallest region at eye that implies a movement, figure 5-3 shows the used algorithms at this field.

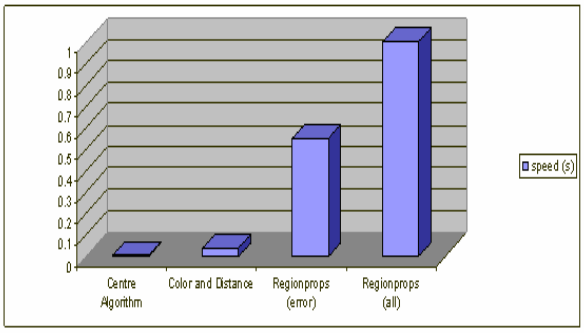

Fig.5.3: Pupil detection algorithm comparison.

- XY Calculation: this step is embedded at the filtration process of our algorithm, so we had avoided this step and gained extra time, figures 5-4 shows the used algorithms at this area.

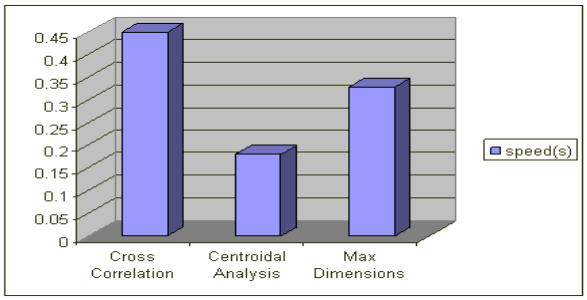

Fig.5.4: X, Y calculations algorithms comparison

- Error Detection: this process contains filtration from noise that came with the frame form the background; at our case we used color and distance filtrations. Figure 5-5 shows the used algorithms at this step.

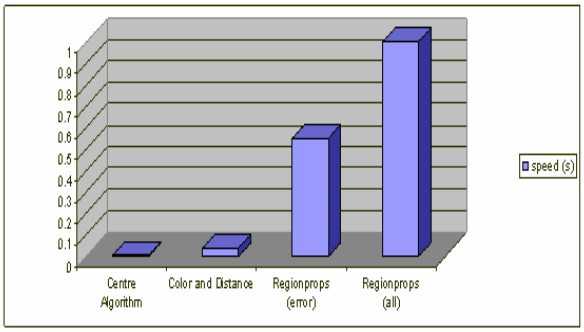

Fig.5.5: Error Detection algorithim comparision.

The values we obtained for other algorithms are precise and accurate; they are obtained from an independent source from the internet, by that we can see that we had developed the optimum solution for each step to ensure the accuracy and the high performance.

5.4. Budget:

The webcam is becoming more integrated at the world of computing with its good quality and high resolution in an affordable price, our recommenced webcam should be with 2 mega pixels and with auto focus property this will cost the user 90 $, on the other hand our system works fine (but with higher error rate) using the 1.3 mega pixels webcam that costs 12$.

Note that the error difference between the 2 webcams is not that much, but it is recommended to use a webcam with higher resolution.

Chapter 6

Conclusion:

6.1 Review:

In this project, the basic ideas of building a real-time system capable of performing eye tracking, blink detection and gaze detection was developed. The system of gaze detection is based on determining the gaze according to ranges of colors after performing several analyses on each frame. The first step after capturing a frame is to perform laplace transformation on it, the main reason behind this step is to eliminate the high frequencies that the present frame contain , the result of this stage will be an image that has the night vision style. Still the image contains some high frequencies so we can't use the resulted image to detect the gaze. The next step after Laplace transformation is to take the grayscale for the resulted image and after this take the inverted version which will make the image ready to be scanned and detect the gaze. The system works well with cheap cameras and does not require camera lens calibration; also the system performance is highly dependent on the system settings.

6.2 Work Results:

At the present time we are able to perform eye detection within the eye, obtain the gaze location and detect the blink within the captured frame, we believe that the algorithm we built is capable to carry out the required task within an acceptable timing offering light load at the system with accurate readings.

The coding progress can be found within the appendix B.

Bibliography

[1] Eyegaze Eyetracking Systems. What is Eye Tracking good for?
http://world.std.com.
Retrieved Septemper. 2007

[2] History of Eye Tracking. LC Technologies
http://www.eyegaze.com
Retrieved Septemper. 2007

[3] Quick Cam® Orbit AF. logitech Webcam
http://www.logitech.com
Retrieved October. 2007

[4] A4 technology. Night vision webcam.
http://www.a4tech.com/en/product2.asp?CID=77&SCID=89&MNO=PK-333MB.
Retrieved Sept. 2007.

[5] Features of C++ Language. The C++ Resources
http://default.co.yu/~xxx/C++/description
Retrieved Septemper. 2007

[6] Programming with OpenCV. Introduction to OpenCV
http://developer.intel.com
Retrieved November. 2007

[7] EYE STRUCTURE AND FUNCTION
http://www.myeyeworld.com/files/eye_structure.htm
Retrieved Septemper. 2007

[8] The Human Visual System
http://www.physics.utoledo.edu/~lsa/_color/17_eye.htm
Retrieved Septemper. 2007

[9] Evolution of the eye
http://en.wikipedia.org/wiki/Evolution_of_the_eye
Retrieved Septemper. 2007

[10] Discrete Fourier Transform
http://local.wasp.uwa.edu.au/~pbourke/other/dft/
Retrieved January. 2008

[11] Mathematical Sciences
http://www.maths.abdn.ac.uk/
Retrieved January. 2008

[12] M. Sonka, V. Hlavac, and R. Boyle, (1999). Image Processing, Analysis, and Machine Vision. Brooks/Cole Publishing Company, second ed

[13] Todd Randall Reed, Boca Raton: CRC Press ‹ c2005 272 p.: ill. ; 25 cm. Digital image sequence processing, compression, and analysis

[14] OpenCV
http://www710.univ-lyon1.fr/~bouakaz/OpenCV-0.9.5/
Retrieved October. 2007

[15] Linear Image Processing
http://www.dspguide.com/CH24.PDF
Retrieved October. 2007

APPENDIX A

Mouse function movements:

Controlling mouse movements software was developed in Microsoft Visual C++ 6.0 environment .In this section details of the functions are presented.

A.1. MousePosition:

This function calculates current mouse position.

```
MousePosition ()
{
  union REGS i,o;

  clrscr();

  i.x.ax=0;
  int86(0x33,&i,&o);

  if(o.x.ax==0)
  {
    printf("No Mouse Available...");
    exit();
  }

  i.x.ax=1;
  int86(0x33,&i,&o);

  gotoxy(25,23);
  printf("Press any key to exit...");

  while(!kbhit())
  {
   i.x.ax=3;
   int86(0x33,&i,&o);
   gotoxy(2,2);
   printf("x->co-ordinate=%d     \n y->co-ordinate=%d      ",o.x.cx,o.x.dx);
  }
```

In the above program we have a while loop. This loop continues until a key is hit. In loop, we used sub-function 3 and invoked mouse interrupt. Sub-function 3 returns X->co-ordinate in cx register and Y->co-ordinate in dx register.printf statements prints x and y co-ordinates as long as the loop continues.Maximum screen resolution for mouse in text mode is 640x200 and in graphics mode is 640x480.

A.2. PressedButton:

This Function below determines which button is pressed as soon as we press any button.

```
PressedButton ()
    {
      union REGS i,o;
      int button;

      clrscr();

      i.x.ax=0;
      int86(0x33,&i,&o);

      if(o.x.ax==0)
      {
        printf("No mouse available....");
        exit();
      }

      i.x.ax=1;
      int86(0x33,&i,&o);

      gotoxy(24,23);
      printf("Press any key to exit....");

      while(!kbhit())
      {
        i.x.ax=3;
        int86(0x33,&i,&o);

        button=o.x.bx&7;

          gotoxy(23,11);
          switch(button)
          {
            case 1:
printf("Left button pressed ");
            break;
            case 2:
printf("Right button pressed  ");
            break;
            case 4:
printf("Middle button pressed");
            break;
            case 3:
printf("Left and Right buttons pressed");
            break;
            case 5:
printf("Left and Middle buttons pressed");
            break;
            case 6:
printf("Right and Middle buttons pressed");
            break;
            case 7:
printf("All the three buttons pressed");
            break;
            default:
          printf("No button pressed....");
          }
       }
         i.x.ax=2;
         int86(0x33,&i,&o);
      }
```

The above program is same as the previous program except we have little extra in while loop. In while we used the same sub-function 3 and invoked mouse interrupts. This subfunction even returns button press information in bx register. Entire button press information is stored in the first 3 bits of the bx register. So we ANDED bx with 7 to separate the first 3 bits and stored them in button variable.

If the first bit's value is 1 then the left button is pressed, if the value is 0 then it is not pressed. If the second bit's value is 1 then the right button is pressed, if value is 0 then it is not pressed. If the last bit's value is 1 then the middle button is pressed, if value is 0 then it is not pressed.

A.3. SetPosition:

This Function set the position of the mouse pointer on the screen

```
SetPosition ()
    {
      union REGS i,o;

      clrscr();

      i.x.ax=0;
      int86(0x33,&i,&o);

      if(o.x.ax==0)
      {
         printf("No mouse available");
         exit();
      }
      i.x.ax=1;
      int86(0x33,&i,&o);

      i.x.ax=3;
      int86(0x33,&i,&o);
      gotoxy(1,1);
      printf("Current Position:x=%d    y=%d    ",o.x.cx,o.x.dx);
      gotoxy(15,23);
      printf("Press any key to set the mouse pointer to (150,100)...");
      getch();

      i.x.ax=4;
      i.x.cx=150;
      i.x.dx=100;
      int86(0x33,&i,&o);
    gotoxy(15,23);
    printf("Cursor is set ... press a key to exit                    ");
    getch();
  }
```

In the above program, we use sub-function 4 to set the pointer's position. We set the X->co-ordinate by placing a value in the cx register and Y->co-ordinate by placing a value in the dx register

APPENDIX B

Gaze detection within a picture:

```
/*****************************************************************************************/
#include <math.h>
#include <stdlib.h>
# include<iostream.h>                          //I/O lib
# include<fstream.h>                          //I/O lib
# include <string.h>
#include <stdio.h>
/*****************************************************************************************/
const int high =1062;          //global variable that determine the high   of the picture
const int width=724;           //global variable that determine the width  of the picture

/*****************************************************************************************/

int main ()
{
/*****************************************************************************************/

  FILE   * pFile;                    //pointer to open input image and scroll through it
  int    c,r,d,                      //c:dummy variable to rescive user input
                                     //r: the y coordinate movement notifier
                                     //d: the x coordinate movement notifier
         goal_ctr=0,                 //entire anticipated number of pixels
         comparator_hori,            //compare the horizental distance validation
         comparator_verti,           //compare the vertical distance validation
         pure_eye_ctr=0,             //store the number of pure pixels found
         dummy,                      //the data variable that temprorary stores each pixel data
         eyes_ctr=0,                 //store how many eyes is detected
         distance_res;
  int    goal[high][3],              //array to store the anticipated eye pixels
         pure_eye[high][3];          //store the pure eye location

/*****************************************************************************************/

    pFile=fopen ("sam2_1062_724.raw","r+b");       //open the raw image in the read binary mode
  if (pFile==NULL) perror ("Error opening file");
                                     //check opening validity
  else
  {
      for (r=0;r<width;r++){         //go through the image pixel by pixel
          for (d=0;d<high;d++){
        dummy=fgetc(pFile);          //get the new charachter

        if(dummy==-1 )               //if un-reconnized data occurs
           break;                    //stop the loop

    if((dummy >0 && dummy <10 ) ||(dummy >30 && dummy <55 )  || dummy==88 || dummy==86){
        /* the color of the inverted laplacian grayscalled eye will be within
        the ranges above according to the angle of light reflection */
        goal[goal_ctr][0]=d;         //store the anticipated X coordinate
        goal[goal_ctr][1]=r;         //store the anticipated Y coordinate
        goal[goal_ctr][2]=dummy;
        goal_ctr++;                  //increase the number of anticipation

        }
          }                          // closing of the first loop
        }                            //closing of the second loop



      for(int scan1=0;scan1<goal_ctr;scan1++)   {  //testing the distance validation
          for(int scan2=0;scan2<goal_ctr;scan2++){
             comparator_hori=abs(goal[scan1][0]-goal[scan2][0]);//horizental distance apart
             comparator_verti=abs(goal[scan1][1]-goal[scan2][1]);//vertical distance apart
                   distance_res=sqrt(pow(comparator_verti,2)+pow(comparator_hori,2));
                      if(distance_res>175 && distance_res<300 && goal[scan1][0]>75 && goal[scan1][1]>75 && goal[scan1][0]<
```

```
          {

                  pure_eye[pure_eye_ctr][0]=goal[scan1][0];       //store the pure X coordinate
                  pure_eye[pure_eye_ctr][1]=goal[scan1][1];       //store the pure Y coordinate
                  pure_eye_ctr++;
                         //we found another pixel
          }



        }

     }
       for( int viewer=0;viewer<pure_eye_ctr;viewer++)           //used to eleminate redundancy and NOISE
                 if(pure_eye[viewer][0]==pure_eye[viewer+1][0] && pure_eye[viewer][1]==pure_eye[viewer+1][1] )
        /*at this case we eleminate 2 source of noise:
        1)noise from redundant sources (2 pixel reveals at the same time with the same
        values of x,y and dummy).
        2)noise from luminating pixels that reflect the light as the eye do, so we apply filtering
        */
                  continue;        //skip the noise reading
                 else{
                 eyes_ctr++;
          cout<<"eye detected in location X:"<<pure_eye[viewer][0]<<" , Y:"<<pure_eye[viewer][1]<<endl;
                 }
                 if(eyes_ctr==0)
                 {   if(goal[0][0]>high/2)

                     cout<<"blink detected in left eye"<<endl;
                 else
                     cout<<"blink detected in  right eye"<<endl;
                 for(int recursive =0;recursive<goal_ctr;recursive++)
if((goal[recursive][2] >0 &&    goal[recursive][2] <10 ) ||(    goal[recursive][2] >30 &&   goal[recursive][2] <55 )
  || ( goal[recursive][2] >85 &&   goal[recursive][2] <95 ))
             cout<<"eye detected in location X:"<<goal[recursive][0]<<" , Y:"<<goal[recursive][1]<<endl;
                 }

 }

cin>>c;                              //show the output

    fclose (pFile);                  //close the opened
    return 0;


}
```