P-3058

HUMAN COMPUTER INTERFACE

A PROJECT REPORT

Submitted by

HAARISH.J	71206106020
MOHAMMED IMRAN.R	71206106032
RAMESH RAJ.S	71206106041
SIVAPRAKASH.S	71206106049

P-3058

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2010

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"HUMAN COMPUTER INTERFACE"** is the bonafide work of "**HAARISH.J, MOHAMMED IMRAN.R, RAMESH RAJ.S, SIVAPRAKASH.S**" who carried out the project work under my supervision.

[illegible]

SIGNATURE 13/4/10

Dr.RAJESWARI MARIAPPAN Ph.D.,

HEAD OF THE DEPARTMENT

Electronics & Communication

Engineering Department

Kumaraguru College of Technology,

Coimbatore-641006

[illegible]

SIGNATURE

Prof. S Govindaraju, M.E

SUPERVISOR

Electronics & Communication

Engineering Department

Kumaraguru College of Technology,

Coimbatore-641006

The candidates with the university register number **71206106020**, **71206106032**, **71206106041**, and **71206106049** was examined by us in the project viva voce examination held on 16.04.2010

[illegible] [illegible]

ACKNOWLEDGEMENT

We are greatly indebted to our beloved Principal **Dr. S. Ramachandran, Ph.D.,** who has been the backbone of all our deeds.

We profusely thank **Dr. Rajeswari Mariappan, Ph.D.,** Head of Department, Department of Electronics and Communication Engineering, for lending a help hand in this project.

We are grateful to our beloved Project coordinator **Ms. A. Vasuki, M.E.,** Assistant Professor, and Project guide **Mr. S. Govindaraju,** Professor, Electronics and Communication Engineering Department for their valuable guidance, timely helps, constant encouragement and advice rendered throughout the project period for the successful completion of the project.

We are also grateful to the faculty members of Electronics and Communication Engineering Department, who have helped us in innumerable ways.

ABSTRACT

In recent years, scientists are aiming to develop non-intrusive man-machine interface with vision systems in a constrained domain. This application aims to replace the most common hardware or instrument used today, the mouse. This project mainly allows the people to interact with the computer for their everyday life by means of mouse pointer.

Among all the image capturing techniques scanning method is used in this project because the sensor moves across the focal plane much like the sensor of a desktop scanner. We first Interface the Camera then we set the required attributes (size, pixels). The image is saved in the specified path. This saved image is read from the path using IMAQ. Then the single color-green is extracted after that Edge detection using Prewitt filter is carried out. Thresholding is done using IMAQ. The region of interest using annulus tool is selected. Finally obtaining X-Y co-ordinates of ROI, the X-Y co-ordinates given as input for setting the cursor position. Cursor moves to the corresponding X-Y co-ordinates.

This project aids the Physically Challenged and the Tetraplegic people to use the mouse with ease. Reduce the dependency on hardware by controlling the cursor by means of eyeball. This method is computationally efficient and cost effective.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

CHAPTER 1
INTRODUCTION

1.1 BRIEF OVERVIEW

In recent years, scientists are aiming to develop non-intrusive man-machine interface with vision systems in a constrained domain. This has led to the emergence of the field of automatic face processing and identification of visual facial behaviors (such as blinking, smiling, frowning etc.), which are instinctively inherent to every human being and thus bringing cognitive sciences closer to computer science. This application aims to replace the most common hardware or instrument (if we may call it so) used today, the mouse. The application reduces the dependency on the hardware by using only two hardware devices that is the CPU and Camera it and demonstrates the importance of the software. It shows how efficiently the software can replace the hardware. It helps to develop an image acquisition and processing framework to track the human eye motion for mouse control. Developing a computationally efficient and cost effective solution for controlling the cursor of the mouse on the screen using the eye ball to aid the physically handicapped and to use it in automotive industries.

1.2 NEED FOR THE PROJECT

The need for this project is to develop a new human-machine interface particularly conceived for people with severe disabilities (specifically Tetraplegic people), that allows them to interact with the computer for their everyday life by means of mouse pointer. This project shall become an indispensable aid for their everyday life. With the help of this project, the physically handicapped people (Tetraplegic) can access the mouse cursor with using their eyeball movement and LabVIEW. This kind of projects in house, interacting with it by means of either remote controls or computers, directly from their bed or their wheel chair. For example, they can open and close doors and windows, switch the appliances on and off, write a letter, use a PC, and so on.

1.3 OBJECTIVE OF THE PROJECT

- To aid the Physically Challenged and the Tetraplegic people to use the mouse with ease.
- Reduce the dependency on hardware.
- Controlling the cursor by means of eyeball.
- Provide the method computationally efficient and cost effective.

IMAGE PROCESSING REQUIREMENTS

CHAPTER 2

IMAGE PROCESSING REQUIREMENTS

2.1 IMAGE CAPTURING

There have been three main methods of capturing the image, each based on the hardware configuration of the sensor and color filters.

2.1.1 CAMERA TYPES

The first method is often called single-shot, in reference to the number of times the camera's sensor is exposed to the light passing through the camera lens. Single-shot capture systems use either one CCD with a Bayer filter mosaic, or three separate image sensors (one each for the primary additive colors red, green, and blue) which are exposed to the same image via a beam splitter.

The second method is referred to as multi-shot because the sensor is exposed to the image in a sequence of three or more openings of the lens aperture. There are several methods of application of the multi-shot technique. The most common originally was to use a single image sensor with three filters (once again red, green and blue) passed in front of the sensor in sequence to obtain the additive color information. Another multiple shot method utilized a single CCD with a Bayer filter but actually moved the physical location of the sensor chip on the focus plane of the lens to "stitch" together a higher resolution image than the CCD would allow otherwise. A third version combined the two methods without a Bayer filter on the chip.

The third method is called scanning because the sensor moves across the focal plane much like the sensor of a desktop scanner. Their linear or tri-linear sensors utilize only a single line of photo sensors, or three lines for the three colors. In some cases, scanning is accomplished by rotating the whole camera; a digital rotating line camera offers images of very high total resolution. For image capturing, we use a digital camera that takes video or still photographs, or both, digitally by recording images via an electronic image sensor.

The main types of camera are broken into three distinct groups. They are progressive area scan, interlaced area scan and line scan.

2.1.2 AREA SCAN

- **Progressive Area Scan**

  If the object is moving quickly, the progressive area camera is to be used. These cameras operate by transferring an entire captured frame from the image sensor, and as long as the image is acquired quickly enough, the motion will be frozen and the image will be a true representation of the object. Progressive scan cameras incorporate direct digital output and eliminate many time consuming processing steps associated with interlacing.

- **Interlaced Area Scan**

  To minimize the amount of video data transmitted over airwaves and to provide a satisfactory resulting picture, interlaced area scan camera is used. The standard interlaced technique is to transmit the picture in two pieces or fields called 2:1 Interlaced Scanning. One artifact of interlaced scanning is that each of the fields is acquired at a slightly different time.

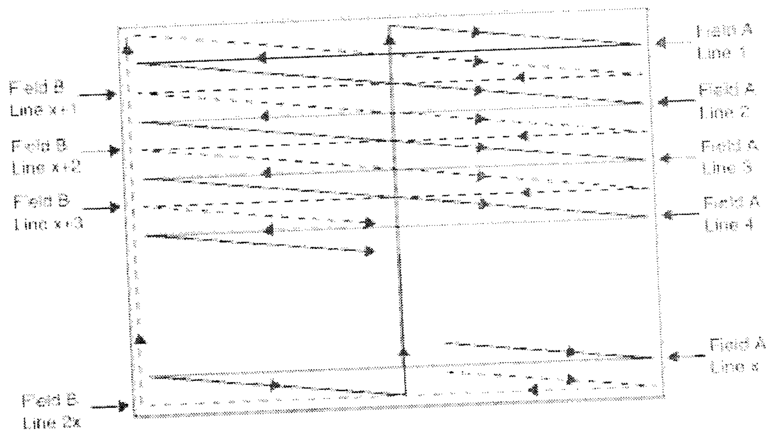


Fig. 2.1.2 Interlaced Scanning Schematic

2.1.3 LINE SCAN

A line-scan camera is a camera device containing a line-scan image sensor chip, and a focusing mechanism. These cameras are almost solely used in industrial settings to capture an image of a constant stream of moving material. Unlike video cameras, line-scan cameras use a single array of pixel sensors, instead of a matrix of them. Data coming from the line-scan camera has a frequency, where the camera scans a line, waits, and repeats. The data coming from the line-scan camera is commonly processed by a computer, to collect the one-dimensional line data and to create a two-dimensional image. The collected two-dimensional image data is then processed by image-processing methods for industrial purposes.

Line-scan technology is capable of capturing data extremely fast, and at very high image resolutions. Usually under these conditions, resulting collected image data can quickly exceed 100 MB in a fraction of a second. Line-scan-camera–based integrated systems therefore are usually designed to streamline the camera's output in order to meet the system's objective, using computer technology which is also affordable.

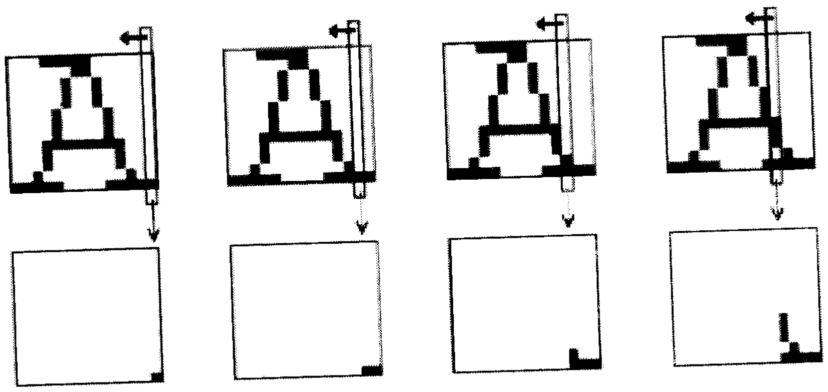


Fig 2.1.3 Line Scan Progression

2.1.4 USB CAMERA

Due to the non availability of area scan camera, we used USB camera.

to computers via USB port. They are well known for their low manufacturing costs and flexible applications. They typically include a lens, an image sensor, and some support electronics. Various lenses are available, the most common being a plastic lens that can be screwed in and out to set the camera's focus. Fixed focus lenses, which have no provision for adjustment, are also available. Support electronics are present to read the image from the sensor and transmit it to the host computer. Some cameras - such as mobile phone cameras - use a CMOS sensor with supporting electronics "on die", i.e. the sensor and the support electronics are built on a single silicon chip to save space and manufacturing costs. The USB video device class (UVC) specification allows for interconnectivity of web cams to computers even without proprietary drivers installed. Microsoft Windows Vista, Linux[7] and Mac OS X 10.4 & 10.5 have UVC drivers built in and do not require extra drivers, although they are often installed in order to add additional features.

2.2 IMAGE FORMAT USED

The USB camera produces still JPEG images which cannot be used for image processing. So they are converted to the compatible PNG format for further processing. The description about the PNG image format is explained below.

2.2.1 PNG FORMAT

Portable Network Graphics (PNG) is a bitmapped image format that employs lossless data compression. PNG was created to improve upon and replace GIF (Graphics Interchange Format) as an image-file format not requiring a patent license. The PNG acronym is optionally recursive, unofficially standing for PNG's Not GIF.PNG supports palette-based (palettes

le or RGB images. PNG was designed for

transferring images on the Internet, not professional graphics, and so does not support other color spaces.

2.2.2 FILE HEADER

A PNG file starts with an 8-byte signature. The hexadecimal byte values are 89 50 4E 47 0D 0A 1A 0A. Each of the header bytes is there for a specific reason.

Table 3.1 Header Bytes and their Function

Bytes	**Purpose**
89	Has the high bit set to detect transmission systems that do not support 8 bit data and to reduce the chance that a text file is mistakenly interpreted as a PNG, or vice versa.
50 4E 47	In ASCII, the letters "PNG", allowing a person to identify the format easily if it is viewed in a text editor.
0D 0A	A DOS style line ending (CRLF) to detect DOS-UNIX line ending conversion of the data.
1A	A byte that stops display of the file under DOS when the command type has been used – the end-of-file character
0A	A UNIX style line ending (LF) to detect UNIX-DOS line ending conversion.

2.2.3 COLOR DEPTH

PNG images can either use palette-indexed color or be made up of one or more channels (numerical values directly representing quantities about the pixels). When there is more than one channel in an image all channels have the same number of bits allocated per pixel (known as the bit depth of the channel). Although the PNG specification always talks about the bit depth of channels, can affect a single pixel, the number of bits per pixel is often higher than the number of bits per channel, as shown in the illustration below.

The number of channels will depend on whether the image is grayscale or color and whether it has an alpha channel. PNG allows the following combinations of channels:

- indexed (channel containing indexes into a palette of colors)
- grayscale
- grayscale and alpha (level of transparency for each pixel)
- red, green and blue (rgb/true color)
- red, green, blue and alpha

Table 3.2 PNG color options, cell values are total bits per pixel for that combination

Type	Bit depth per channel				
	1	**2**	**4**	**8**	**16**
indexed (color type 3)	1	2	4	8	No
grayscale (color type 0)	1	2	4	8	16
grayscale & alpha (color type 4)	No	No	No	16	32
true color (RGB: color type 2)	No	No	No	24	48
true color & alpha (RGBA: color type 6)	No	No	No	32	64

With indexed color images, the palette is always stored in RGB at a depth of 8 bits per channel (24 bits per palette entry). The palette must not have more entries than the image bit depth allows for but it may have fewer (so if an image for example only uses 90 colors there are no need to have palette entries for all 256).

per pixel. Everything else uses a bit depth per channel of either 8 or 16. The combinations this allows are given in the table above. The standard requires that decoders can read all supported color formats but many image editors can only produce a small subset of them.

P-3058

2.2.4 TRANSPARENCY OF IMAGE

PNG offers a variety of transparency options. With true color and greyscale images either a single pixel value can be declared as transparent or an alpha channel can be added. For paletted images, alpha values can be added to palette entries. The number of such values stored may be less than the total number of palette entries, in which case the remaining entries are considered fully opaque.

The scanning of pixel values for binary transparency is supposed to be performed before any color reduction to avoid pixels becoming unintentionally transparent. This is most likely to pose an issue for systems that can decode 16 bits per channel images (as they must be compliant with the specification) but only output at 8 bits per channel (the norm for all but the highest end systems).

2.2.5 IMAGE COMPRESSION

PNG uses a non-patented lossless data compression method known as DEFLATE, which is the same algorithm used in the zlib compression library. This method is combined with prediction, where for each image line, a filter method is chosen that predicts the color of each pixel based on the colors of previous pixels and subtracts the predicted color of the pixel from the actual color. An image line filtered in this way is often more compressible than the raw image line would be, especially if it is similar to the line above (since DEFLATE has no understanding that an image is a 2D entity, and instead just sees the image data as a stream of bytes). Compression is further improved by and a heuristic method of implementing it commonly used by PNG-writing software, were created by Lee Daniel Crocker, who tested the methods on many images during the creation of the format.

2.2.6 COMPARISON WITH GRAPHICS INTERCHANGE FORMAT (GIF)

- On most images, PNG can achieve greater compression than GIF (see the section on file size, below).
- PNG gives a much wider range of transparency options than GIF, including alpha channel transparency.
- PNG gives a much wider range of color depths than GIF (true color up to 48-bit compared to 8-bit 256-color), allowing for greater color precision, smoother fades, etc.[11]
- GIF intrinsically supports animated images. PNG only supports animation using an unofficial extension (see the section on animation, above).
- PNG images are widely supported (for instance, with modern web browsers and office software), but not as widely supported as GIF images.

2.2.7 COMPRESSION WITH JOINT PHOTOGRAPHY EXPERTS GROUP (JPEG)

JPEG (Joint Photography Experts Group) can produce a smaller file than PNG for photographic (and photo-like) images, since JPEG uses a lossy encoding method specifically designed for photographic image data. Using PNG instead of a high-quality JPEG for such images would result in a large increase in file size (often 5–10 times) with negligible gain in quality.

PNG is a better choice than JPEG for storing images that contain text, line art, or other images with sharp transitions. Where an image contains both sharp transitions and photographic parts a choice must be made between the large but sharp PNG and a small JPEG with artifacts around sharp transitions. JPEG also does not support transparency.

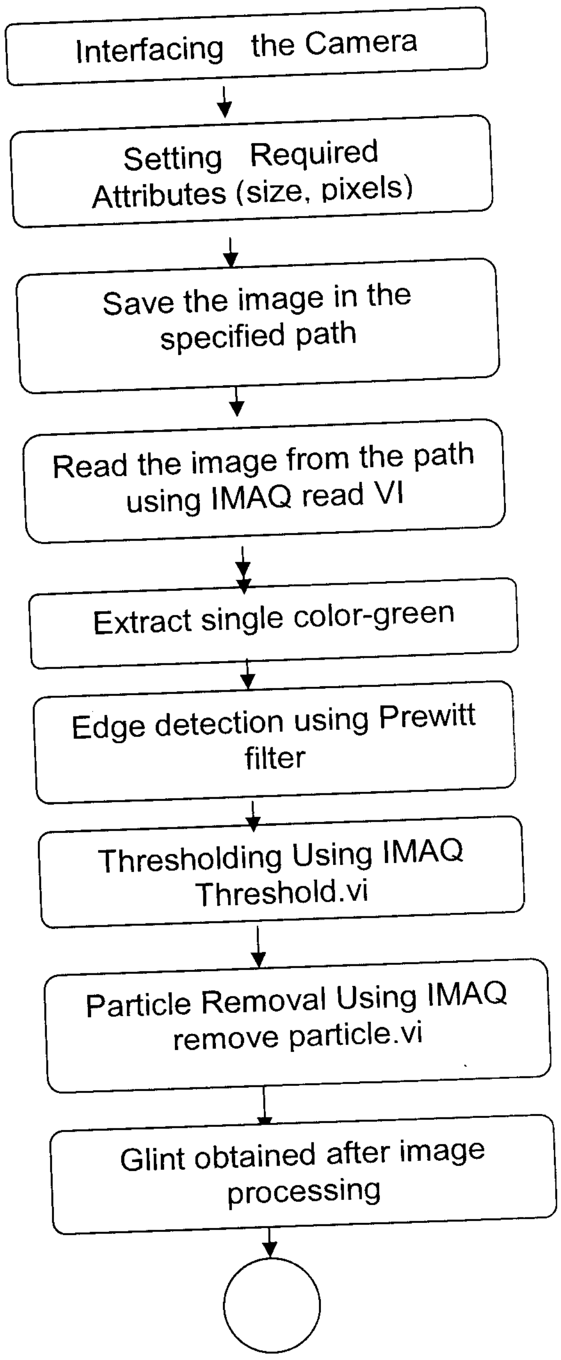
JPEG is a worse choice for storing images that require further editing as it suffers from generation loss, whereas lossless formats do not. This makes PNG useful for saving temporary photographs that require successive editing. When the photograph is ready to be distributed, it can then be saved as a JPEG, and this limits the information loss to just one generation. That said, PNG does not provide a standard means of embedding Exif image data from sources such as digital cameras, which makes it problematic for use amongst photographers, especially professionals. TIFF does support it as a lossless format.

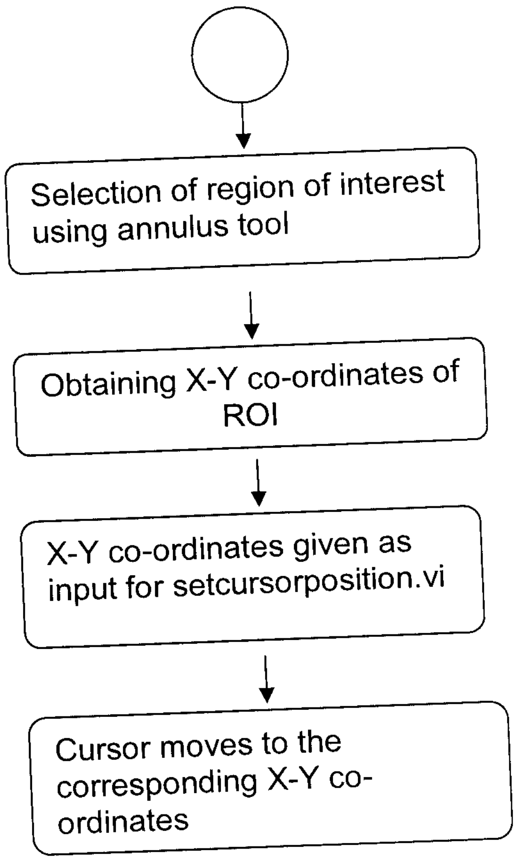
JPEG has historically been the format of choice for exporting images containing gradients, as it could handle the color depth much better than the GIF format. However, any compression by the JPEG would cause the gradient to become blurry, but a 24-bit PNG export of a gradient image often comes out identical to the rasterization of the source vector image, and at a small file size. As such, the PNG format is the optimal choice for exporting small, repeating gradients for web usage.

SINGLE FRAME

CHAPTER 3
SINGLE FRAME

3.1 METHOLOGY


Interfacing the Camera
Setting Required Attributes (size, pixels)
Save the image in the specified path
Read the image from the path using IMAQ read VI
Extract single color-green
Edge detection using Prewitt filter
Thresholding Using IMAQ Threshold.vi
Particle Removal Using IMAQ remove particle.vi
Glint obtained after image processing


Selection of region of interest using annulus tool
Obtaining X-Y co-ordinates of ROI
X-Y co-ordinates given as input for setcursorposition.vi
Cursor moves to the corresponding X-Y co-ordinates

3.2 IMAGE ACQUISITION

LabVIEW image acquisition is highly effective and enables the user to create systems which can analyze and recognize patterns or other recurrent features in analogue or digital images. The block diagram for the Image acquisition is given below.

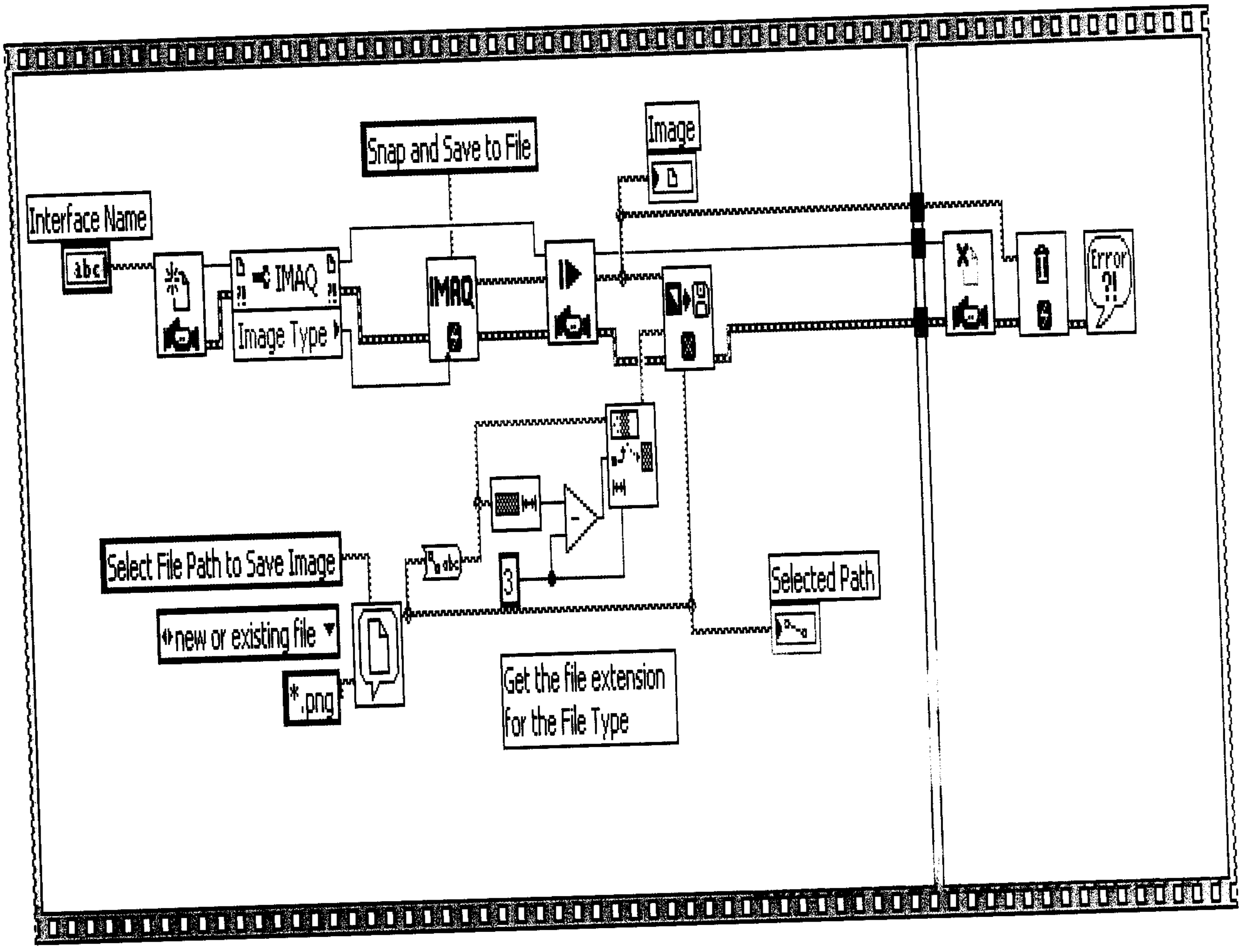
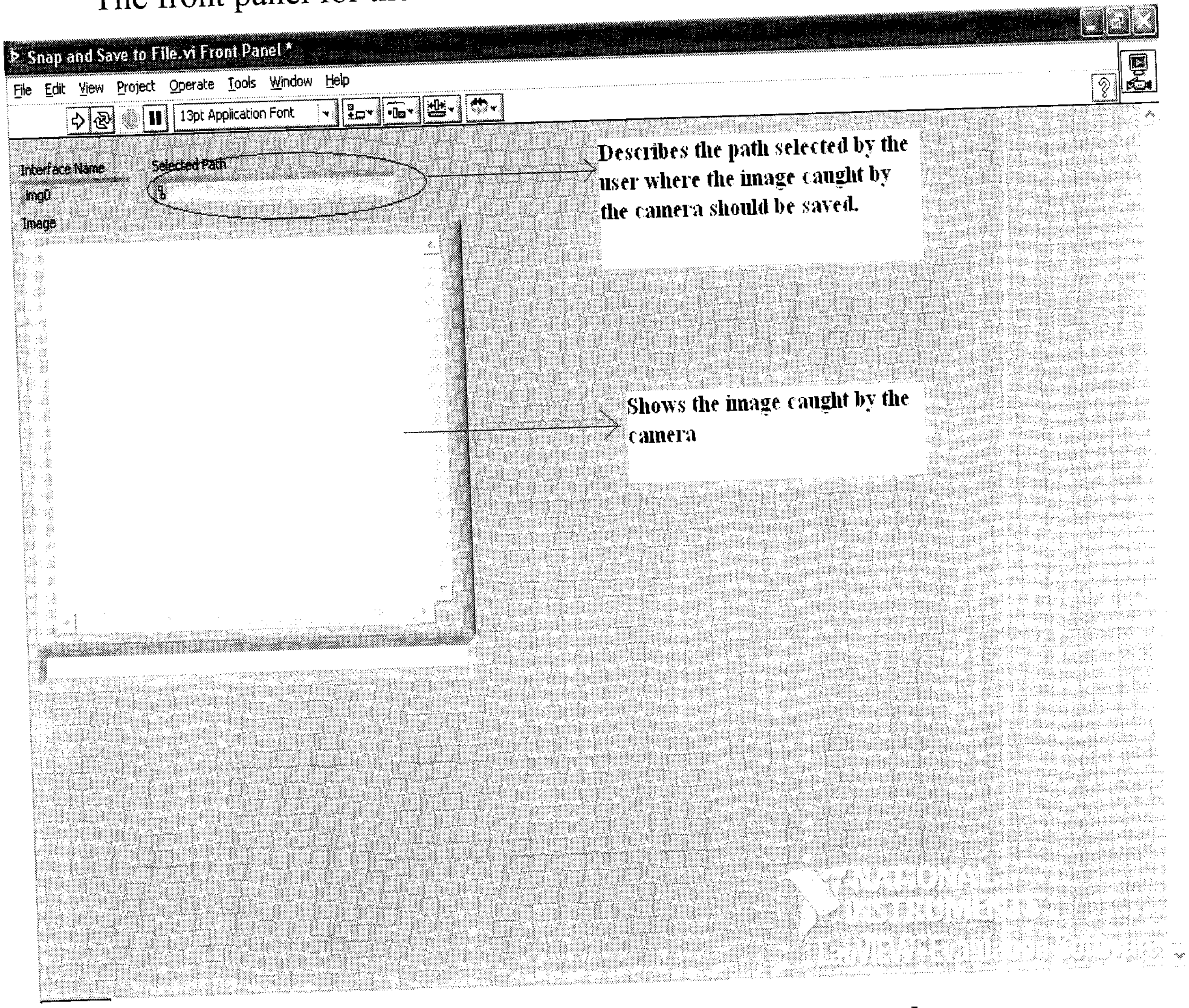


Fig 3.2(a) Block Diagram for Image Acquisition

The block diagram is used to obtain the required image from the USB webcam. First the availability of the interface is checked and after that its configuration such as type of image, pixels size etc are obtained which is fed to the IMAQ create VI. Then using the image type and the border size determined by the user in the IMAQ create VI, the IMAQ snap VI obtains the image in the required attributes. Then this image is saved in the specified location in the specified type (PNG) determined by the user using the IMAQ write VI. After the image is saved, the resources used by the application are released by the IMAQ close and IMAQ dispose VI's.

Suppose if any error occurs such as no interface found or no path handler VI and also specifies the type of error to the user. After successfully acquiring the image in the specified location, it can be used for further processing.

The front panel for the above block diagram is shown below.



Snap and Save to File.vi Front Panel *
File Edit View Project Operate Tools Window Help
13pt Application Font
Interface Name
Img0
Selected Path
Image
Describes the path selected by the user where the image caught by the camera should be saved.
Shows the image caught by the camera
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software

Fig 3.2(b) Image Acquisition front panel

3.3 IMAGE PROCESSING AND MOUSE CONTROL

An image is an array, or a matrix, of square pixels (picture elements) arranged in columns and rows. The image acquired from the camera is processed using the following VI files and functions to obtain the glint of the eye for further process.

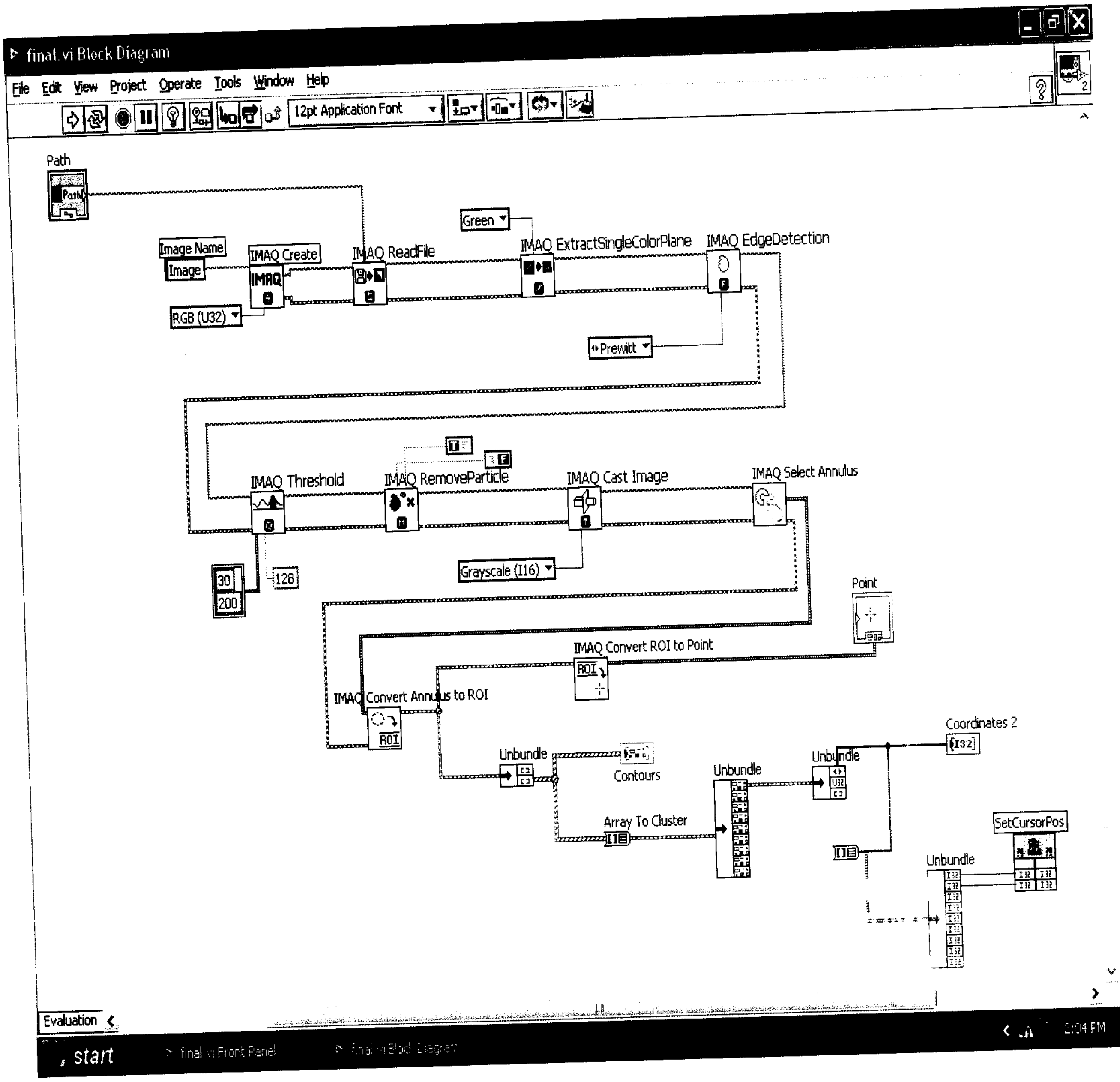
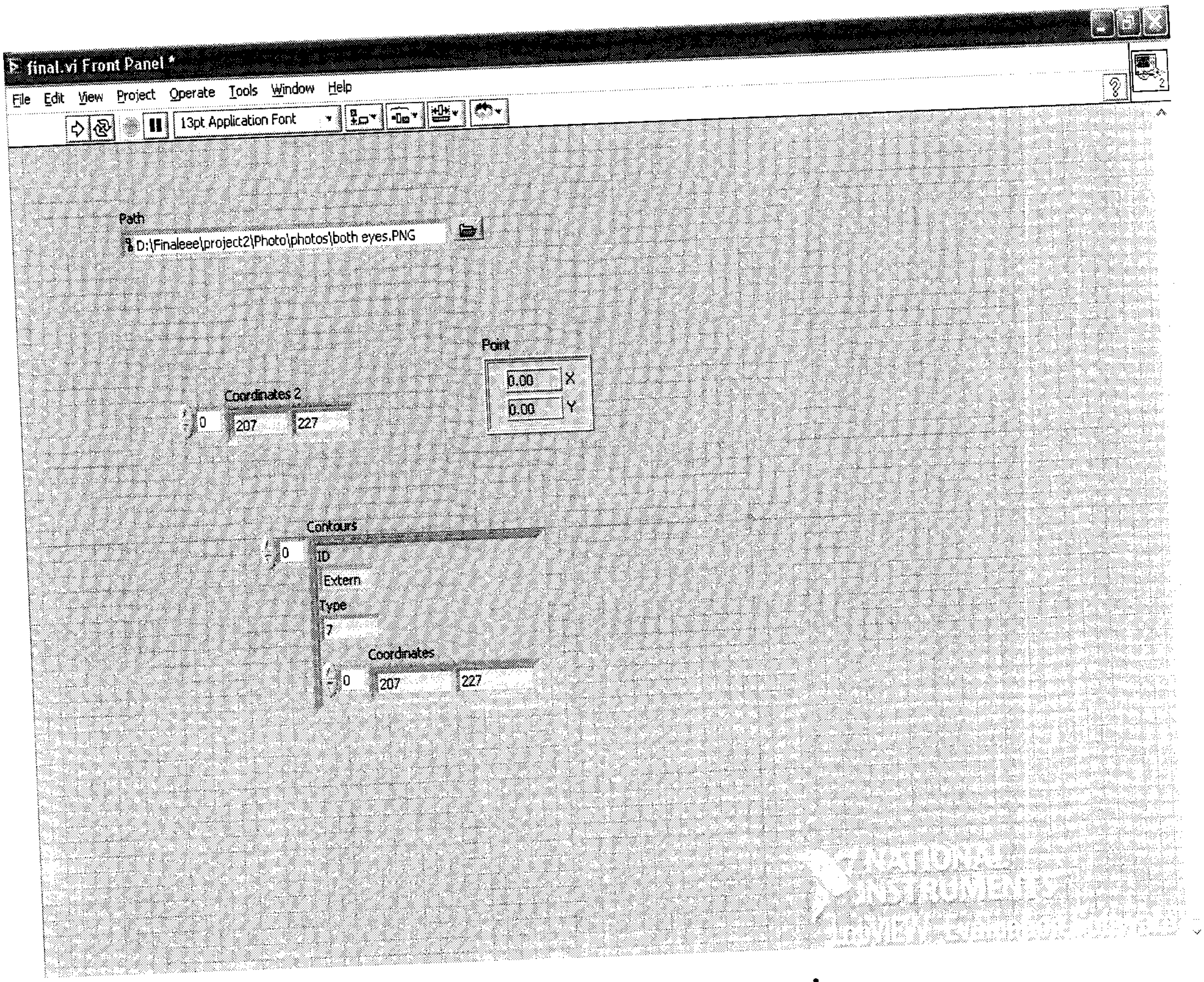


Fig 3.3(a) Block diagram for the image processing.

The above block diagram is used for image processing and cursor movement. First the image is read from the specified path described by the user using the IMAQ read file VI, then using IMAQ extract single color plane VI we extract green color alone out of the image and then this extracted image is used for edge detection by Prewitt filter using IMAQ edge detection VI and the unwanted particles are removed using IMAQ remove particle VI and the obtained image is converted into gray scale type using IMAQ cast image VI.


final.vi Front Panel *
File Edit View Project Operate Tools Window Help
13pt Application Font
NATIONAL INSTRUMENTS LabVIEW Evaluation Software

Path
D:\Finaleee\project2\Photo\photos\both eyes.PNG

Point
0.00 X
0.00 Y

Coordinates 2
0 207 227

Contours
0
ID
Extern
Type
7
Coordinates
0 207 227

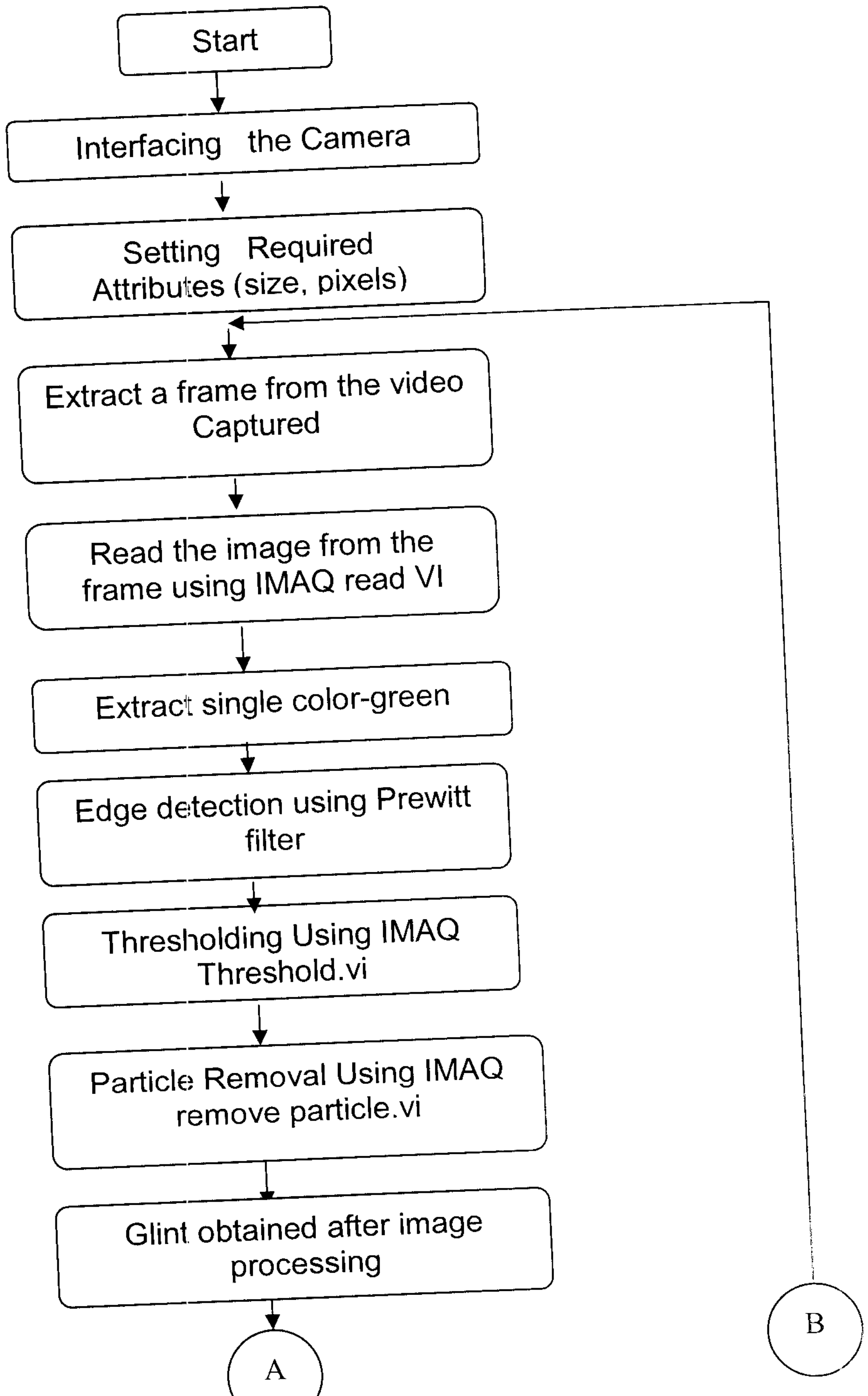
Fig 3.3(b) Front panel of the image processing

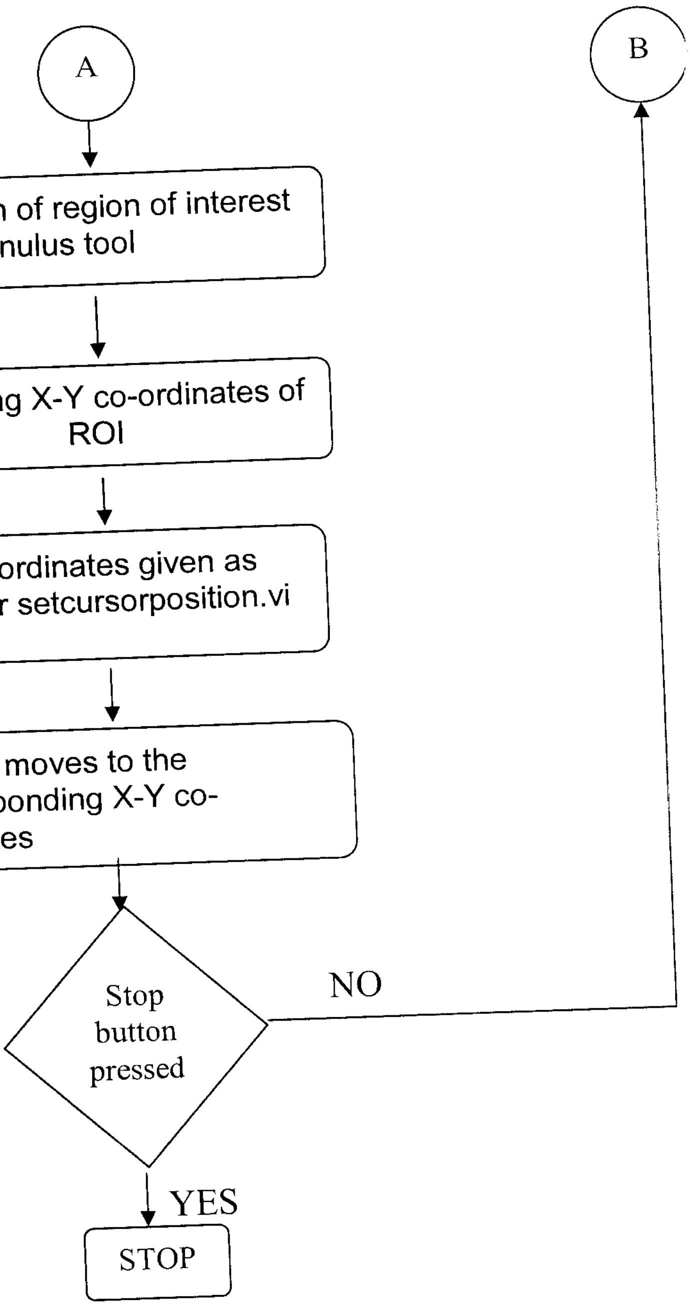
Then the glint is obtained which is selected as the Region Of Interest (ROI) using the annulus tool using the IMAQ select annulus VI and IMAQ convert annulus to ROI VI. Then the XY co-ordinates obtained by selecting the ROI is fed as the input to the mouse cursor co-ordinates using the set cursor position VI as shown above. And thus according to the XY co-ordinates fed the mouse cursor moves.

MULTI FRAMES

CHAPTER 4
MULTI FRAMES

4.1 METHOLOGY


Start
Interfacing the Camera
Setting Required Attributes (size, pixels)
Extract a frame from the video Captured
Read the image from the frame using IMAQ read VI
Extract single color-green
Edge detection using Prewitt filter
Thresholding Using IMAQ Threshold.vi
Particle Removal Using IMAQ remove particle.vi
Glint obtained after image processing
A
B


A
B
Selection of region of interest using annulus tool
Obtaining X-Y co-ordinates of ROI
X-Y co-ordinates given as input for setcursorposition.vi
Cursor moves to the corresponding X-Y co-ordinates
Stop button pressed
NO
YES
STOP

4.2 IMAGE ACQUISITION

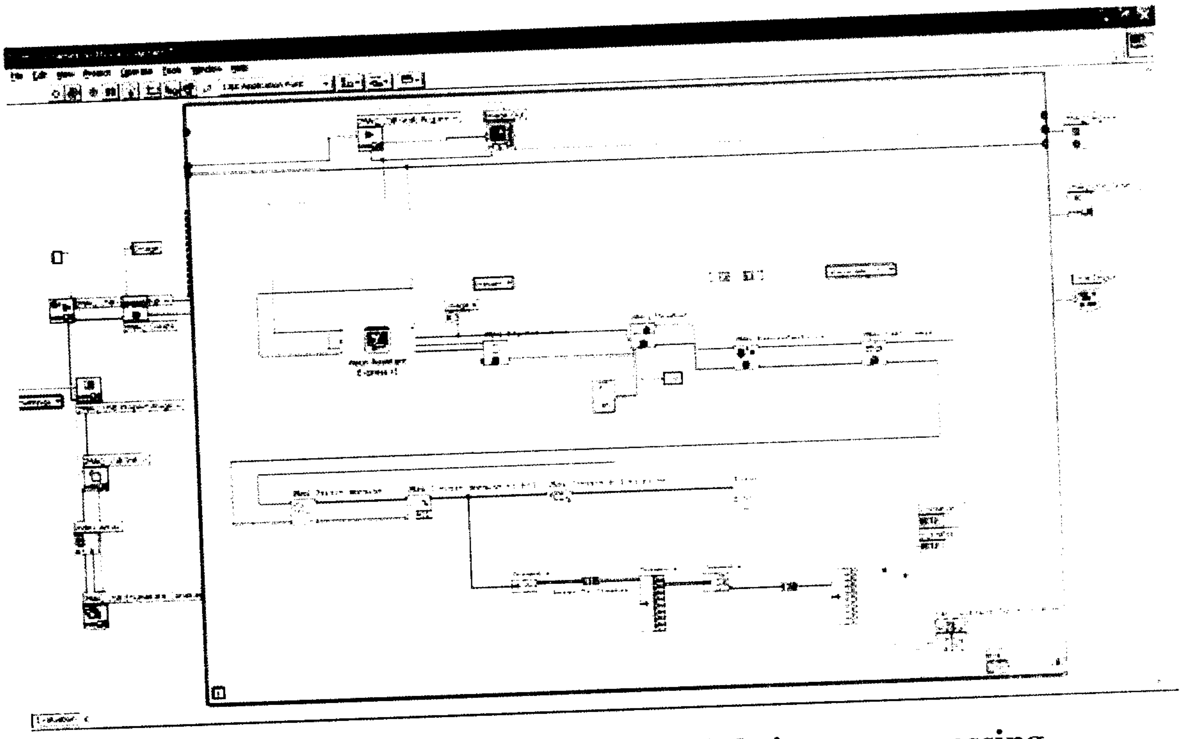


Fig 4.2(a) Block diagram for the multiple image processing.

The block diagram is used to obtain the required image from the USB web cam. First the availability of the interface is checked and after that its configuration such as frame rate, resolution, output size, compression, pixels size etc are obtained which is fed to the IMAQ create VI. Then using the image type and the border size determined by the user in the IMAQ create VI, the IMAQ Grab VI obtains the image in the required attributes. Then this image is determined by the user using the IMAQ write VI. After the image is saved, the resources used by the application are released by the IMAQ close and IMAQ dispose VI's.

Suppose if any error occurs such as no interface found or no path specified to save the image etc, then these errors are handled by the simple error handler VI and also specify the type of error to the user. After successfully

ified location, it can be used for further

The front panel for the above block diagram is shown below.

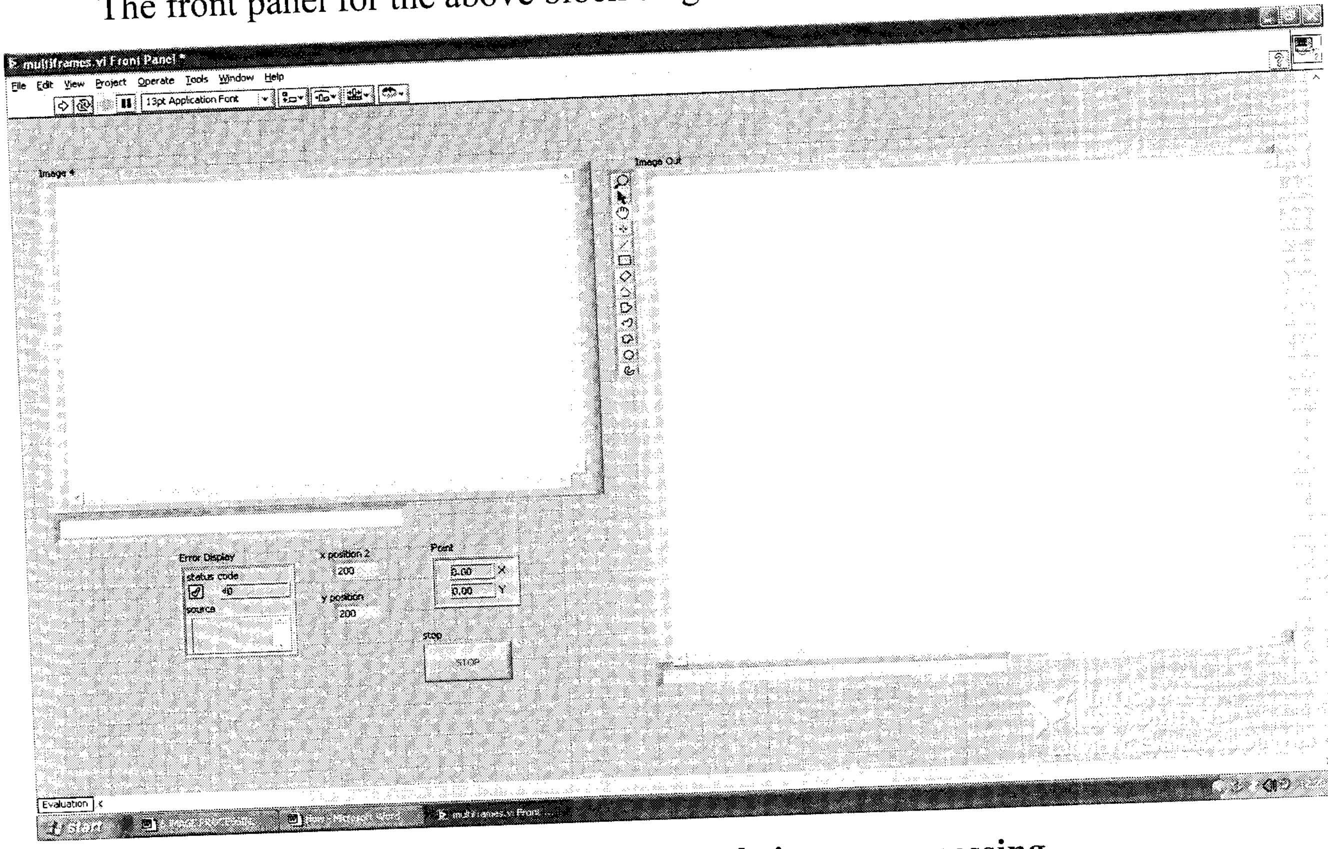


Fig4.2 (b) Front panel of multiple image processing

4.3 IMAGE PROCESSING

Single frame is obtained from the video and then it is passed through image processing and cursor movement sections. First the image is read from the specified path described by the user using the IMAQ read file VI, then using the IMAQ extract single color plane VI we extract green color alone out of the image and then this extracted image is used for edge detection by Prewitt filter using IMAQ edge detection VI and the obtained image is used for thresholding using IMAQ threshold VI. Then the unwanted particles are removed using IMAQ remove particle VI and the obtained image is converted into gray scale type using IMAQ cast image VI.

Then the glint is obtained which is selected as the Region Of Interest (ROI) using the annulus tool using the IMAQ select annulus VI and IMAQ

ROI is fed as the input to the mouse cursor co-ordinates using the set cursor position VI as shown above. And thus according to the XY co-ordinates fed the mouse cursor moves.

VIRTUAL INSTRUMENTATION – LABVIEW

CHAPTER 5

VIRTUAL INSTRUMENTATION – LABVIEW

5.1 INTRODUCTION

A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as plug-in boards, and driver software, which together perform the functions of traditional instruments. Virtual instruments represent a fundamental shift from traditional hardware-centered instrumentation systems to software-centered systems that exploit the computing power, productivity, display, and connectivity capabilities of popular desktop computers and workstations.

Although the PC and integrated circuit technology have experienced significant advances in the last two decades, software truly offers the flexibility to build on this powerful hardware foundation to create virtual instruments, providing better ways to innovate and significantly reduce cost. With virtual instruments, engineers and scientists build measurement and automation systems that suit their needs exactly (user-defined) instead of being limited by traditional fixed-function instruments (vendor-defined).The project is implemented with the help of the LabVIEW, virtual instrumentation software.

5.2 PRODUCTS USED

LabVIEW 8.5.1

NI Vision Assistant 8.5

5.3 THE CHALLENGE

Developing a computationally efficient and cost effective solution for

on the screen using the eye ball to aid the

5.4 THE SOLUTION

Developing a highly accurate and flexible image processing application using NI Vision Assistant and Lab VIEW to control the movement of the cursor by capturing real time images of the eyeball.

5.5. FEATURES OF LabVIEW

LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution. In LabVIEW, we can build a user interface with a set of tools and objects. LabVIEW programs are called virtual instruments (VIs). Controls are inputs and indicators are outputs.

Each VI contains three main parts:

- Front Panel – How the user interacts with the VI.
- Block Diagram – The code that controls the program.
- Icon/Connector – Means of connecting a VI to other VIs.

The user interface is known as the front panel. We can then add code using the graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart. LabVIEW also includes several wizards to help us quickly configure the DAQ devices and computer based instruments and build applications.

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as ... Thermometers, lights, and other indicators display output values from

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.

LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

5.5.1 CONTROL PALETTE

The control palette is used to place the controls and indicators on the front panel. The controls palette is available only on the front panel. To view the palette, we should select Window>>Show Controls Palette. We can also display the controls palette by right clicking an empty area on the front panel. Tack down the control palette by clicking the pushpin on the top left corner of the palette.

5.5.2 FUNCTIONS (AND STRUCTURES) PALETTE

The Functions palette is used to build the block diagram. The Functions palette is available only on the block diagram. To view the palette, we should select Window>>Show Functions Palette. We can display the Functions palette by right-clicking an open area on the block diagram. Tack down the Functions palette by clicking the pushpin on the top left corner of the palette.

5.5.3 TOOLS PALETTE

If automatic tool selection is enabled and if we move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the Tools palette. Toggle automatic tool selection by clicking the Automatic Tool Selection button in the Tools palette. We can use the Operating tool to change the values of a control or select the text within a control. We can use the Positioning tool to select, move, or resize objects. The Positioning tool changes shape when it moves over a corner of a resizable object. We can use the Labeling tool to edit text and create free labels. The Labeling tool changes to a cursor when you create free labels. We can use the Wiring tool to wire objects together on the block diagram. The other important tools are Scrolling Tool, Breakpoint Tool, Probe Tool, Color Copy Tool, Coloring Tool, and Shortcut Menu Tool.

5.6 ADVANTAGES OF LabVIEW

- Modular.
- Easier to debug.
- Don't have to recreate code.
- Require less memory.
- Self learning with context HELP

IMAGE ACQUSITION TOOLS

CHAPTER 6
IMAGE ACQUSITION TOOLS

6.1 IMAQ CREATE

Creates a temporary memory location for an image. Use IMAQ Create in conjunction with the IMAQ Dispose VI to create or dispose of NI Vision images in LabVIEW.

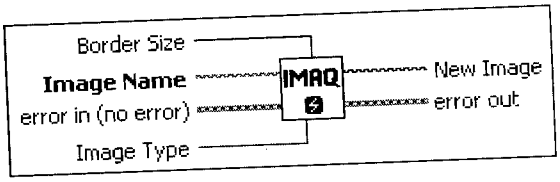


Fig 6.1 IMAQ Create

Border Size: It determines the width, in pixels, of the border to create around an image. These pixels are used only for specific VIs. Create a border at the beginning of your application if an image is to be processed later using functions that require a border (for example, labeling and morphology). The default border value is 3. With a border of three pixels, you can use kernels up to 7 × 7 with no change. If a kernel larger than 7 × 7 is needed in the process, then specify a larger border when creating the image.

Image Name: It is the name associated with the created image. Each image created must have a unique name.

Error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. Use the Simple Error Handler or General Error Handler VIs used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Image Type: It specifies the image type. Choose from the following values:

Grayscale (U8) (0)	8 bits per pixel (unsigned, standard monochrome)
Grayscale (16) (1)	16 bits per pixel (signed)
Grayscale (SGL) (2)	32 bits per pixel (floating point)
Complex (CSG) (3)	2 × 32 bits per pixel (floating point)
RGB (U32) (4)	32 bits per pixel (red, green, blue, alpha)
HSL (U32) (5)	32 bits per pixel (hue, saturation, luminance, alpha)
RGB (U64) (6)	64 bits per pixel (red, green, blue, alpha)

New Image: It is the **Image** reference that is supplied as input to all subsequent (downstream) functions used by NI Vision. Multiple images can be created in LabVIEW applications

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.2 IMAQ DISPOSE

Destroys an image and frees the space it occupied in memory. This VI is required for each image created in an application to free the memory allocated to the IMAQ Create VI. Execute IMAQ Dispose only when the image is no longer needed in your application. The IMAQ Dispose can be used for each call [illegible] all images created with IMAQ Create

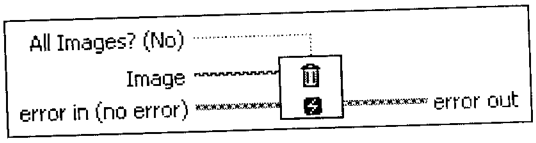


Fig 6.2 IMAQ Dispose

All Images? (No): It specifies whether to destroy a single image or all previously created images. Giving a TRUE value on input destroys all images previously created. The default is FALSE. Be sure to use this function at the end of an application to free the memory occupied by the images.

Image: It specifies the reference to the image to destroy.

Error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.3 IMAQDX CLOSE CAMERA

Stops an acquisition in progress, releases resources associated with the acquisition, and closes the specified session.

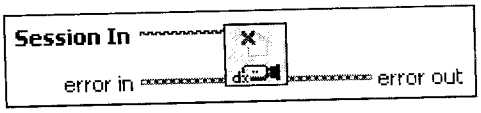


Fig 6.3 IMAQdx Close Camera

Session In: It is a unique reference to the camera, which you can obtain with IMAQdx Open Camera.vi.

Error in: It describes error conditions that occur before this VI or function runs.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.4 IMAQDX CONFIGURE GRAB

Configures and starts a grab acquisition. A grab performs an acquisition that loops continually on three buffers. The grab function has to be used for high-speed image acquisition. The IMAQdx Grab.vi is used to copy an image out of the buffer. If this VI is called before calling IMAQdx Open Camera.vi, IMAQdx Configure Grab.vi uses cam0 by default. IMAQdx Unconfigure Acquisition.vi is used to unconfigure the acquisition.

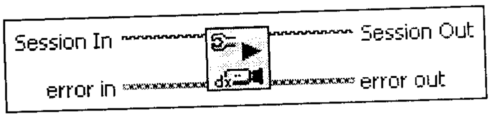


Fig 6.4 IMAQdx Configure Grab

Session In: It is a unique reference to the camera, which can be obtained with IMAQdx Open Camera.vi.

Error in: It describes error conditions that occur before this VI or function runs.

Session Out: It is a unique reference to the camera. **Session Out** is the same as Session In.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.5 IMAQDX ENUMERATE CAMERAS

Returns a list of all interface files on the host computer.

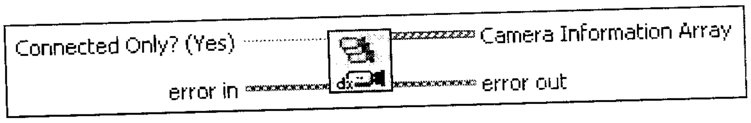


Fig 6.5 IMAQdx Enumerate Cameras

Connected Only? (Yes) if the **Connected Only?** Value is **Yes**, then the Camera Information Array only contains cameras that are currently connected to the host computer. If the **Connected Only?** Value is **No**, then the Camera Information Array contains cameras that are currently connected, and were

Error in: It describes error conditions that occur before this VI or function runs.

Camera Information Array: It is an array of interface files that are on the host computer. This includes cameras that are currently connected or that were previously connected.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.6 IMAQDX GRAB

Acquires the most current frame into **Image Out**. This VI has to be called only after calling IMAQdx Configure Grab.vi. If the image type does not match the video format of the camera, this VI changes the image type to a suitable format.

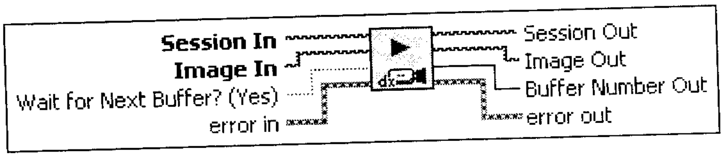


Fig 6.6 IMAQdx Grab

Session In: It is a unique reference to the camera, which can be obtained with IMAQdx Open Camera.vi.

Image In: It is the reference to the image that receives the captured pixel data.

Wait for Next Buffer? (Yes) if the value is **Yes**, the driver will wait for the next available buffer. If the value is **No**, the driver will not wait for the next available buffer, and will instead return the last acquired buffer.

Error in: It describes error conditions that occur before this VI or function runs.

Session Out: It is a unique reference to the camera. Session Out is the same as Session In.

Image Out: It's the reference to the captured image.

Buffer Number Out: It is the actual acquired buffer number returned.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.7 IMAQDX OPEN CAMERA

Opens a camera, queries the camera for its capabilities, loads a camera configuration file, and creates a unique reference to the camera. The IMAQdx Close Camera.vi is used when the reference is finished.

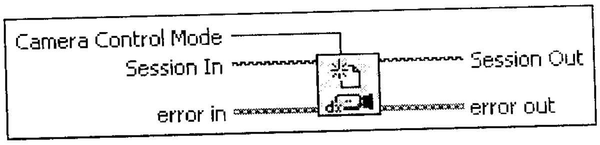


Fig 6.7 IMAQdx Open Camera

Camera Control Mode is the control mode of the camera used during image broadcasting. The default value is **Controller**.

Controller Mode	Actively configures and acquires image data
Listener Mode	Passively acquires image data from a session opened in controller mode on a different host or target computer.

Session In: It specifies the name of the camera you wish to open. The default value is **cam0**.

Error in: It describes error conditions that occur before this VI or function runs.

Session Out: It is a unique reference to the camera. Session Out is the same as Session In.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

6.8 INDEX ARRAY

Returns the element or subarray of n-dimension array at index.

When an array is wired to this function, the function resizes automatically to display **index** inputs for each dimension in the array, wired to **n-dimension array**. Additional **element or subarray** terminals can be added by resizing the function. The connector pane displays the default data types for this polymorphic function.

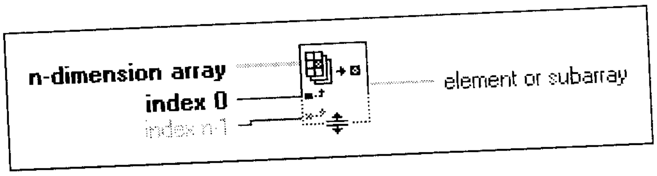


Fig 6.8 Index Array

N-dimension array: It can be an n-dimensional array of any type. If n-dimension array is an empty array, element or subarray returns the default value of the defined data type for the array.

Index 0...n-1: It must be numeric. The number of index inputs matches the number of dimensions in n-dimension array.
If the **index** is out of range (<0 or N, where N is the size of **n-dimension array**), **element or subarray** returns the default value of the defined data type. A subarray of the array can be extracted by leaving one or more of the index terminals unwired.

IMAGE PROCESSING TOOLS

CHAPTER 7
IMAGE PROCESSING TOOLS

7.1 IMAQ EDGE DETECTION VI:

This extracts the contours (detects the edges) in grey-level values. The connected source image must have been created with a border capable of supporting the size of the processing matrix. For example, a 3*3 matrix has a minimum border size of 1. The border size of the destination image is not important.

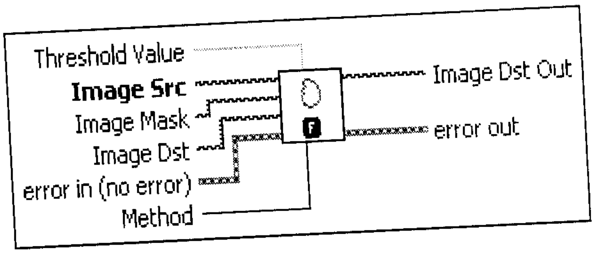


Fig 7.1 IMAQ edge detection VI

Image Scr: This is a reference to the source image.

Image Dst out: This is a reference to the destination image.

Image Mask: It is an 8-bit image that specifies the region of the small image that will be copied. Only those pixels in the Image Src (Small) image that correspond to an equivalent non-zero pixel in the mask image are copied. All other pixels keep their original values. The entire image is processed if Image Mask is not connected.

Error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

7.1.1 EDGE DETECTION TECHNIQUE USED

Edge detection is a terminology in image processing and computer vision, particularly in the areas of feature detection and feature extraction, to refer to algorithms which aim at identifying points in a digital image at which the image brightness changes sharply or more formally has discontinuities. Edge detection is an effective tool for many machine vision applications. It provides your application with information about the location of object boundaries and the presence of discontinuities.

We can use edge detection in the following three application areas: gauging, detection, and alignment. Filters smooth, sharpen, transform, and remove noise from an image so that you can extract the information you need. Nonlinear filters either extract the contours (edge detection) or remove the isolated pixels. NI Vision has six different methods you can use for contour extraction (Differentiation, Gradient, Prewitt, Roberts, Sigma, or Sobel).

7.1.2 PREWITT FILTER

The Prewitt filter is a high pass filter that extracts the outer contours of local edge orientation for each pixel. Prewitt edge detection produces an image where higher grey-level values indicate the presence of an edge between two objects. It highlights significant variations of the light intensity along the vertical and horizontal axes. This example uses the following source image.

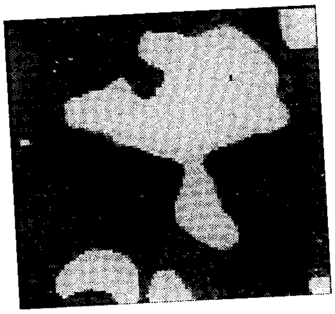


Fig 7.1(a) Source image

A Prewitt filter produces the following image.

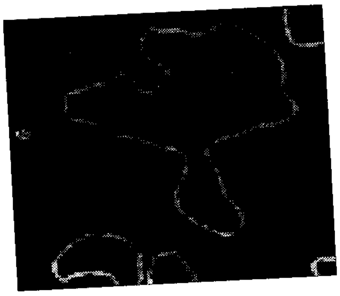


Fig 7.1(b) Image after prewitt filtering

There are many ways to perform edge detection. However, the most may be grouped into two categories, gradient and Laplacian. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. The Laplacian method searches for zero crossings in the second derivative of the image to find edges. This first figure shows the edges of an image detected using the gradient method (Roberts, Prewitt, Sobel) and the Laplacian method (Marrs-Hildreth).

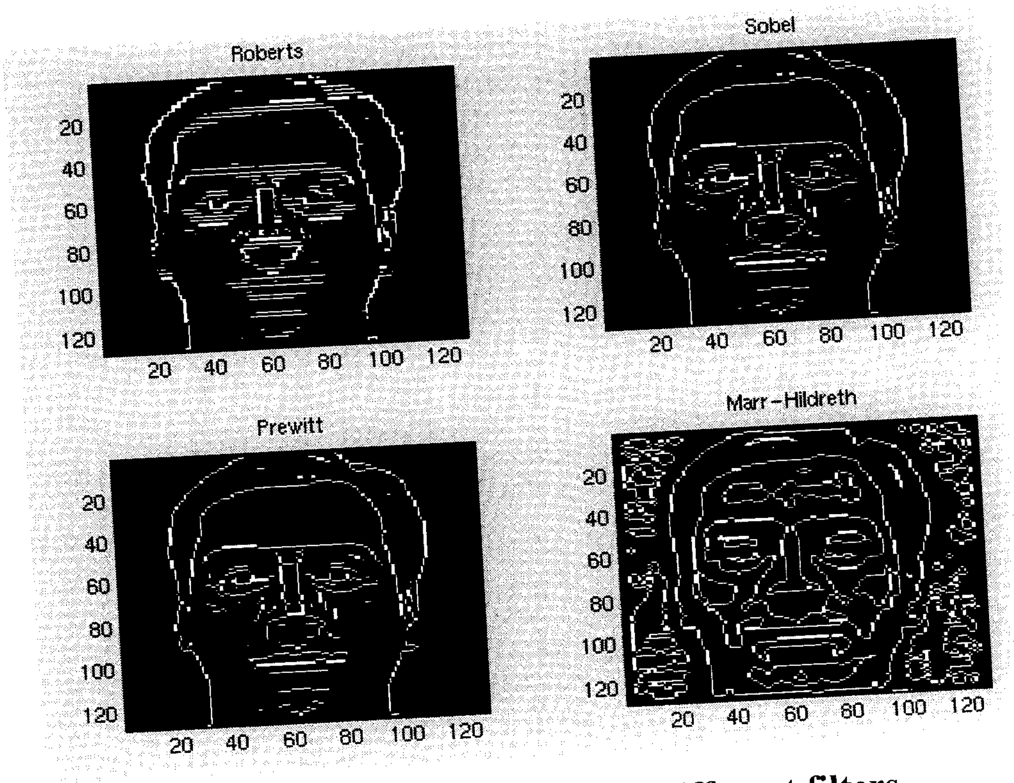


Fig 7.1(c) Images obtained by different filters

Thus from the above image we can see that prewitt filter performs better edge detection than other type of filters in edge detection.

Notice that the facial features (eyes, nose, mouth) have very sharp edges. Notice also that the Marr-Hildreth not only has a lot more noise than the other methods. Due to the nature of the Sobel and Prewitt filters we can select out only vertical and horizontal edges of the image.

7.2 IMAQ THRESHOLD VI:

Applies a threshold to an image.

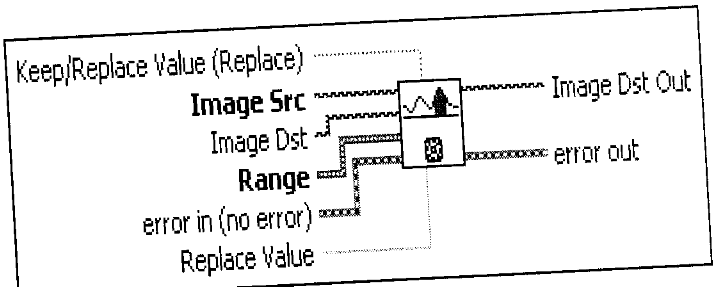


Image Scr: It is a reference to the source image.

Image Dst out: It is a reference to the destination image.

Replace value: It is the value used to replace pixels between the lower value and upper value. This operation requires that *keep/replace value* is true.

Keep/Replace Value (Replace): It determines whether to replace the value of the pixels existing in the range between Lower value and Upper value. The default status, TRUE, replaces these pixel values, and the status FALSE keeps the original values.

Range: It is a cluster specifying the lower and upper value of the threshold range.

- **Lower value**

It is the lowest pixel value used during a threshold. Default is 128.

- **Upper value**

It is the highest pixel value used during a threshold. Default is 255.

Error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

7.3 IMAQ REMOVE PARTICLE VI:

It Eliminates or keeps particles resistant to a specified number of 3*3 erosions. The particles that are kept are exactly the same shape as those found in the original source image .The source image must be an 8-bit binary image.

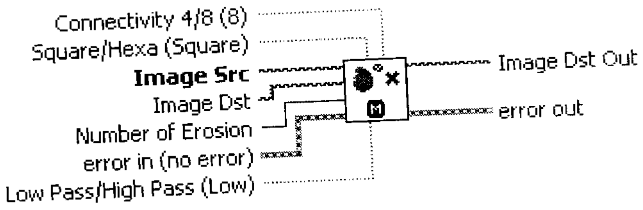


Fig 7.3 IMAQ Remove Particle VI

Image Src: It is a reference to the source image

Image Dst: It is a reference to the destination image

Number of erosion: It specifies the number of 3*3 erosions to apply to the image. The default is 2.

Error in (no error): It describes the error status before this VI or function runs. The default is no error. If an error occurs before this VI or function runs, the VI or function passes the error in value to the error out. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs when this VI or function runs, it runs normally and sets its own error status in error out.

objects resistant to n

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

7.4 IMAQ CAST IMAGE VI:

This VI converts the current image type to the image type specified by Image Type. If a lookup table is specified, the IMAQ Cast Image VI converts the image using a lookup table. If converting from a 16-bit image to an 8-bit image, the VI executes this conversion by shifting the 16-bit pixel values to the right by the specified number of shift operations and then truncating to get an 8-bit value.

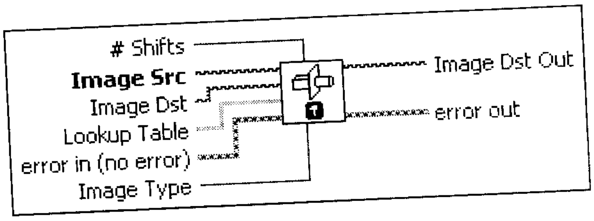


Fig 6.9 IMAQ Cast Image VI

#Shifts: It specifies the number of right shifts by which each pixel value in the input image is shifted. This is valid only when converting from a 16-bit image to an 8-bit image. This VI executes this conversion by shifting the 16-bit pixel values to the right by the specified number of shift operations, up to a maximum of 8 shift operations, and then truncating to get an 8-bit value. To ignore the bit depth and shift 0, the value of -1 must be entered. To use the bit depth to cast the image, the value of 0 must be entered.

It is a reference to the source image.

Lookup Table: It is an array containing a maximum of 256 elements if Image Src is an 8-bit image or a maximum of 65,536 elements if Image Src is a 16-bit image. The array contains values equal to the index if there are fewer than the amounts determined by the image type in Image Src. Lookup Table can be used to calculate a function that gives a relation between a gray level value and a user value. This input is valid only when converting from an 8-bit image to a 16-bit image, from a 16-bit image to an 8-bit image or from an 8-bit or 16-bit image to a 32-bit floating point image.

Error in (no error): It describes the error status before this VI or function runs. The default is no error. If an error occurs before this VI or function runs, the VI or function passes the error in value to the error out. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs when this VI or function runs, it runs normally and sets its own error status in error out.

Image Type: It is used to specify the image type from the following types:

Table 6.2 Image types

Grayscale(U8)	8 bits per pixel(unsigned, standard monochrome)
Grayscale(16)	16 bits per pixel(signed)
Grayscale(SGL)	32 bits per pixel(floating point)
Complex(CSG)	2 x 32 bits per pixel(floating point)
RGB(U32)	32 bits per pixel(red, green, blue, alpha)
HSL(U32)	32 bits per pixel (hue, saturation, luminance, alpha)
RGB(U64)	64 bits per pixel(red, green, blue, alpha)
Grayscale(U16)	16 bits per pixel(unsigned, standard monochrome)

Error out: It contains error information. If *error in* indicates that an error information. Otherwise, it describes the error status that this VI or function produces.

Image Dst Out: It is a reference to the destination image. If Image Dst is connected, Image Dst Out is the same as Image Dst. Otherwise; Image Dst Out refers to the image referenced by the Image Src.

7.5 WHILE LOOP:

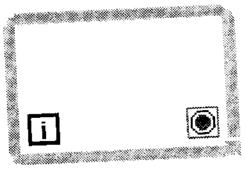


Fig 7.5 While Loop

Repeats the subdiagram inside it until the conditional terminal, an input terminal, receives a particular Boolean value. The Boolean value depends on the continuation behavior of the While Loop. Right-click the conditional terminal and select **Stop if True** or **Continue if True** from the shortcut menu. The While Loop always executes at least once.

The iteration (**i**) terminal provides the current loop iteration count, which is zero for the first iteration. If iteration count exceeds 2,147,483,647, or 2^{31}, the iteration terminal remains at 2,147,483,647 for all further iterations. If the counting for more than 2,147,483,647 iterations is needed, then shift registers with a greater integer range can be used.

After the While Loop is created, the shift registers can be used to pass values from one iteration to the next. If an array is wired to a While Loop, the process for every element in that array by enabling auto-indexing can be read.

7.6 IMAQ EXTRACT SINGLE COLOR PLANE

Extracts a single plane from a color image.

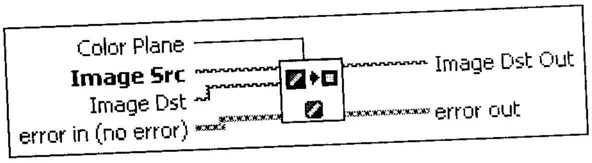


Fig 7.6 IMAQ Extract Single Color Plane

Color Plane defines the color plane to extract. Choose from the following values:

Red (0)	(Default) Extracts the red color plane
Green (1)	Extracts the green color plane
Blue (2)	Extracts the blue color plane
Hue (3)	Extracts the hue color plane
Saturation (4)	Extracts the saturation color plane
Luminance (5)	Extracts the luminance color plane
Value (6)	Extracts the value color plane
Intensity (7)	Extracts the intensity color plane

Image Src: It is the reference to a color image that has one of its color planes extracted. If **Image Dst** is not connected, the source image is converted to an image that contains the extracted plane.

Image Dst: It is a reference to the destination image. If **Image Dst** is connected, it must be the same size as the **Image Src**.

Error in (no error): This describes the error status before this VI or function If an error occurred before this VI or function runs,

runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Image Dst Out: It is a reference to the destination image. If **Image Dst** is connected, **Image Dst Out** is the same as **Image Dst**. Otherwise; **Image Dst Out** refers to the image referenced by **Image Src**.

error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

MOUSE CONTROL TOOLS

CHAPTER 8
MOUSE CONTROL TOOLS

8.1 IMAQ SELECT ANNULUS:

It allows the user to specify an annulus area in an image. IMAQ Select Annulus displays the image in the specified window, provides the annulus tool, and returns the coordinates of the annulus selected when the user clicks OK in the window.

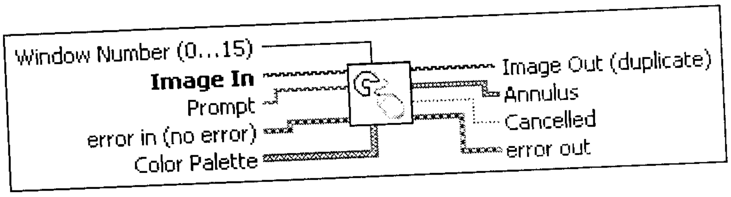


Fig 8.1 IMAQ Select Annulus

Window Number (0…15): It specifies the window which displays the image. When the default value is selected, this VI uses a modal dialog window, centered in the screen. When the regular NI Vision Window number (0…15) is selected, the VI displays Image in the specified window and temporarily sets the NI vision window to modal mode. When the user clicks OK or Cancel in the window, the attributes of the window are set back to their initial values.

Image In: It is a reference to the image on which the user selects the Annulus.

Prompt: It specifies a message string to display in the title bar of the window. This control is used to provide the user with instructions about selecting the object.

he VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Color Palette: It is used to apply a color palette to the image window. Color Palette is an array of clusters constructed by the user or supplied by the IMAQ GetPalette VI. This palette is composed of 256 elements for each of the three color planes (red, green, and blue). A specific color is the result of applying a value between 0 and 255 to each of the three color planes. If the three planes have identical values, a gray level is obtained (0 specifies black and 255 specifies white). If the image type requires a color palette and it is not supplied, a grayscale color palette is generated and written.

Image Out (duplicate): It is a reference to Image In. This VI does not modify the image connected to the Image In input.

Annulus: It specifies the coordinates of the annulus chosen by the user. Its various attributes are given as

- Center X specifies the x-coordinate of the center of the circle or annulus.
- Center Y specifies the y-coordinate of the center of the circle or annulus.
- Inner Radius is the radius (in pixels) of the inner circle defining one edge of the circular strip
- Outer Radius is the radius (in pixels) of the outer circle defining the

- Start Angle is the first angle of the circular strip. This angle is measured counterclockwise from the x-axis of the coordinate system centered on the center of the annulus.
- End Angle is the second angle of the circular strip. This angle is measured counterclockwise from the x-axis of the coordinate system centered on the center of the annulus.

Cancelled: It returns TRUE if the user ends the selection by clicking Cancel in the window.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

8.2 IMAQ CONVERT ANNULUS TO ROI VI:

It converts an annulus to an ROI descriptor.

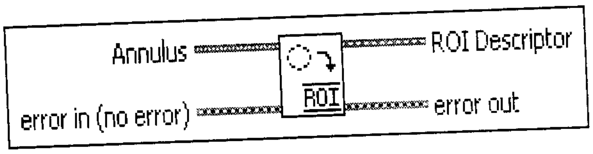


Fig 8.2 IMAQ Convert Annulus to ROI

Annulus: It specifies the coordinates of the annulus chosen by the user. Its various attributes are given as

- Center X specifies the x-coordinate of the center of the circle or annulus.
- Center Y specifies the y-coordinate of the center of the circle or annulus.
- Inner Radius is the radius (in pixels) of the inner circle defining one edge

- Outer Radius is the radius (in pixels) of the outer circle defining the second edge of the circular strip.
- Start Angle is the first angle of the circular strip. This angle is measured counterclockwise from the x-axis of the coordinate system centered on the center of the annulus.
- End Angle is the second angle of the circular strip. This angle is measured counterclockwise from the x-axis of the coordinate system centered on the center of the annulus.

error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

ROI Descriptor: It is a descriptor that defines an annulus.

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

8.3 IMAQ CONVERT ROI TO POINT VI:

It converts an ROI descriptor to a point element.

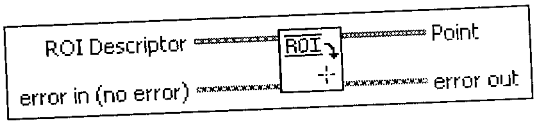


Fig 8.3 IMAQ Convert ROI to Point

ROI Descriptor: It is a descriptor that defines an annulus.

Error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

Point: It is a point-coordinate cluster.

- X is the x-coordinate of the point
- Y is the y-coordinate of the point

Error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

8.4 UNBUNDLE FUNCTION:

This function splits a cluster into each of its individual elements. When a cluster is wired to this function, the function resizes automatically to display outputs for each element in the cluster that is wired. The connector pane displays the default data types for this polymorphic function.

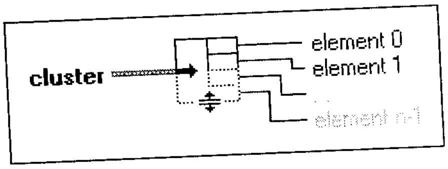


Fig 8.4 Unbundle Function

This function produces an output these elements in the same order they appear in the cluster. The number of outputs for this function must match the number of elements in the cluster. When two or more elements of the same type are present, keep track of their order in the cluster.

Cluster: It is the cluster whose elements we have to access.

Element (0…n-1): These are the elements of the cluster.

8.5 ARRAY TO CLUSTER FUNCTION:

It converts a 1D array to a cluster of elements of the same type as the array elements. The number of elements in the cluster is set by right-clicking the function and selecting the Cluster Size from the shortcut menu. The default is nine and the maximum cluster size is 256.

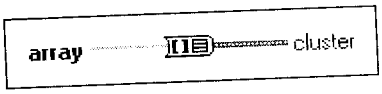


Fig 8.5 Array to Cluster Function

Array: It is a one-dimensional array of any type.

Cluster: Each element in cluster is the same as the corresponding element in array. The cluster order matches the order of the elements in the array.

8.6 CALL LIBRARY FUNCTION NODE:

It calls a DLL or shared library function directly.

The Call Library Function Node supports a large number of data types and calling conventions. It can be used to call functions in most standard and custom-made DLLs and shared libraries. If a DLL that contains ActiveX objects has to be called, use the Automation Open VI with the Property Node and the Invoke Node. The Call Library Function Node is expandable and shows data types for the wired inputs and outputs, similar to the Bundle function.

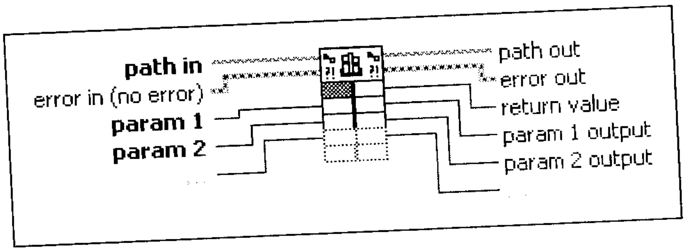


Fig 8.6 Call Library Function Node

path in: It identifies the path to the DLL or shared library you want to call. The checkmark must be placed in the **Specify path on diagram** checkbox in the Call Library Function dialog box for this input to appear on the connector pane.

error in: This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

param 1...n: These are example input parameters of the library function.

path out: It returns the path to the called DLL or shared library. The checkmark must be placed in the **Specify path on diagram** checkbox in the Call Library Function dialog box for this output to appear on the connector pane.

error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

return value: It is an example return value of the library function.

param 1...n output: These are example output parameters of the library function.

8.7 IMAQ CLAMP HORIZONTAL MIN

Measures a distance in the horizontal direction, from the center of the search area towards the vertical sides of the search area. This VI locates edges along a set of parallel search lines, or rake. The edges are determined based on their contrast and slope.

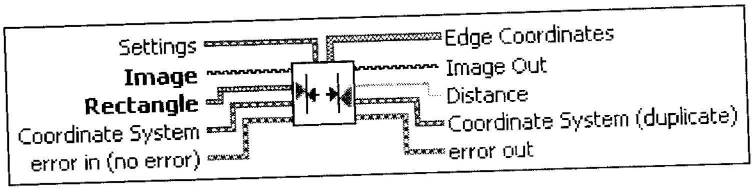


Fig 8.7 IMAQ Clamp Horizontal Min

Settings is a cluster defining the parameters of the edge detection algorithm and the information that is overlaid on the result image. The first three parameters specify the filter used to detect the edges.

- **Contrast** specifies the threshold for the contrast of the edge. Only edges with a contrast greater than this value are used in the detection process. **Contrast** is defined as the difference between the average pixel intensity before the edge and the average pixel intensity after the edge.
- **Filter width** specifies the number of pixels that the VI averages to find the contrast at either side of the edge.
- **Steepness** specifies the slope of the edge. This value represents the number of pixels that correspond to the transition area of the edge.
- **Subsampling Ratio** specifies the number of pixels that separate two consecutive search lines of the rake.
- **Show Search Area** determines whether the search area is overlaid on the image.
- **Show Search Lines** determines whether the search lines used to locate the edges are overlaid on the image.
- **Show Edges Found** determines whether the locations of the edges found are overlaid on the result image.
- **Show Result** determines whether the hit lines to the object are overlaid on the result image.

Rectangle: It specifies the coordinates of a rectangular search area.

- **Left** is the x-coordinate of the upper left corner of the rectangle
- **Top** is the y-coordinate of the upper left corner of the rectangle
- **Right** is the x-coordinate of the bottom right corner of the rectangle.
- **Bottom** is the y-coordinate of the bottom right corner of the rectangle
- **Rotation** specifies the rotation angle in degrees of the rectangle with its center as point of rotation. If the rotation angle does not equal zero, the **Left**, **Top**, **Right**, and **Bottom** coordinates are not the actual coordinates of the upper left and bottom right corner of the rectangle, but their position if the rotation angle equals zero.

Coordinate System specifies the coordinate system to which the **Rectangle** is linked. If the **Coordinate System** input is used (connected), the location of the intensity measurement is shifted and rotated by the difference between the reference position of the coordinate system and its new location.

error in (no error): This describes the error status before this VI or function runs. The default is no error. If an error occurred before this VI or function runs, the VI or function passes the *error in* value to *error out*. This VI or function runs normally only if no error occurred before this VI or function runs. If an error occurs while this VI or function runs, it runs normally and sets its own error status in error out. The Simple Error Handler or General Error Handler VIs are to display the description of the error code. The *error in* and *error out* is used to check errors and to specify execution order by wiring error out from one node to error in of the next node.

It is an array of point clusters consisting of the spatial

Image Out: It is a reference to the destination image. The search area and/or the result of the measurement may be overlaid on the image according to the settings

Distance: It returns the distance measured between the hit lines

Coordinate System (duplicate): It is a reference to the coordinate system.

error out: It contains error information. If *error in* indicates that an error occurred before this VI or function ran, *error out* contains the same error information. Otherwise, it describes the error status that this VI or function produces.

Elapsed time (ms): It is the elapsed time, in ms, of the search.

RESULTS

CHAPTER 9
RESULTS AND DISCUSSIONS

9.1 PROCESSED OUTPUT

After getting the image of the eye and processing it using the above said techniques we get the output as follows.

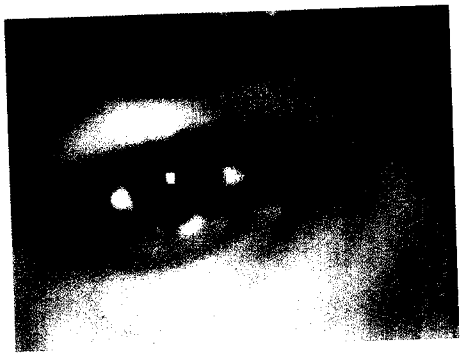


Fig 9.1(a) Acquired image

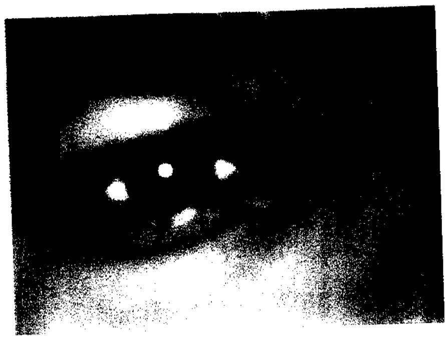
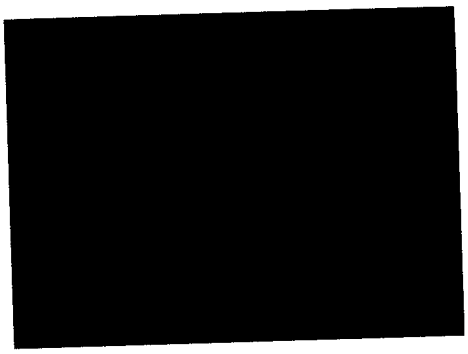


Fig 9.1(b) Green colour extracted image



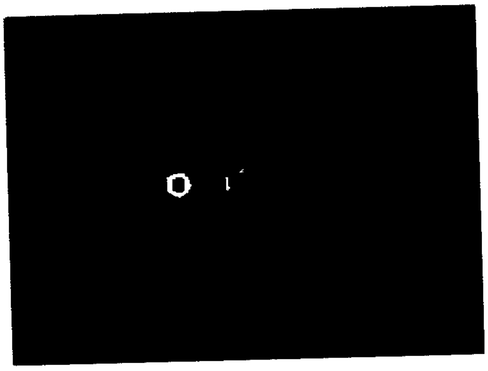


Fig 9.1(d) Image after thresholding

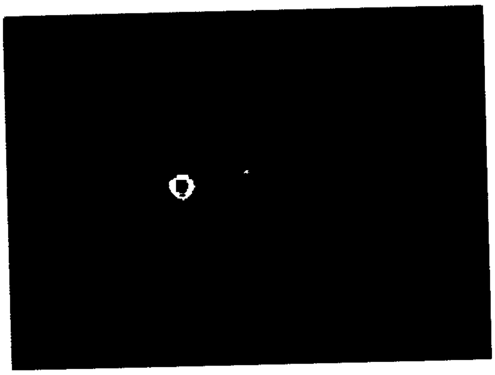


Fig 9.1(e) Image after particle removal

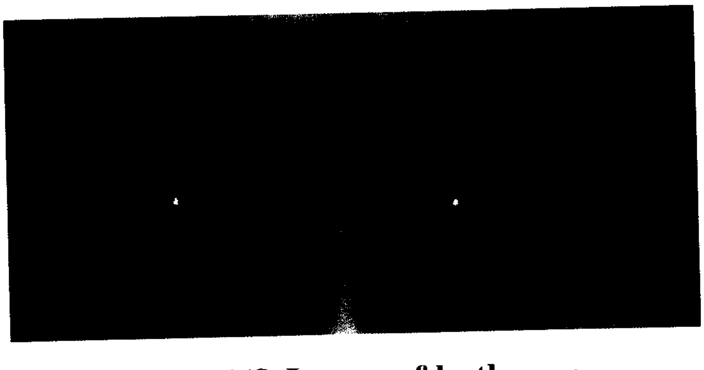
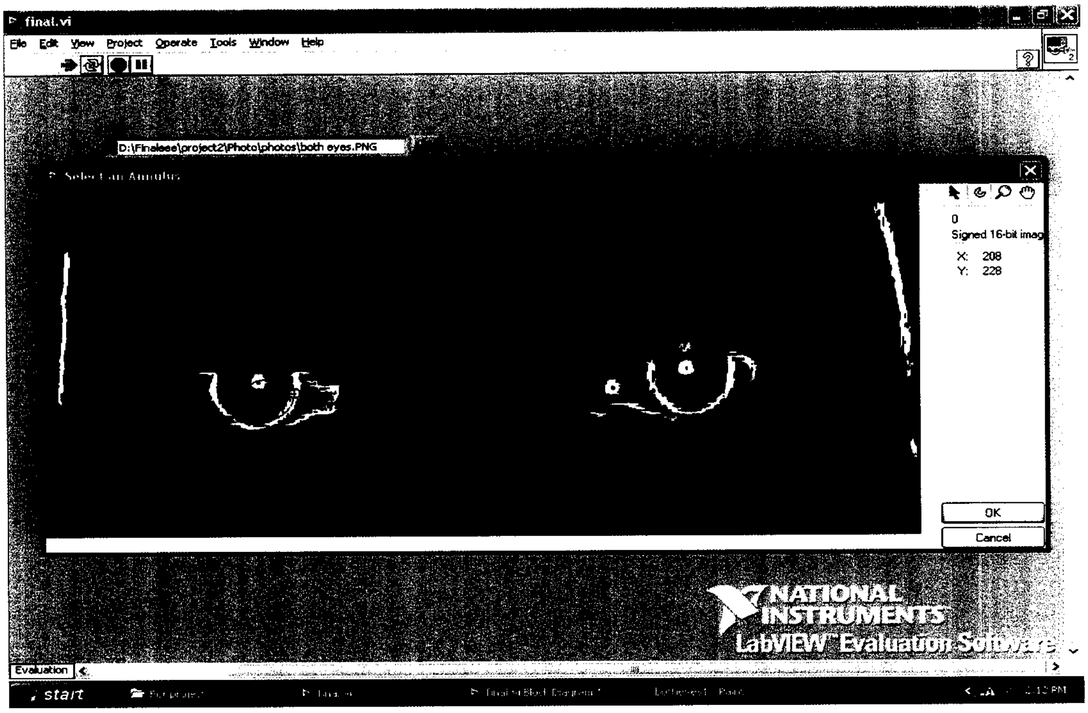


Fig 9.1(f) Image of both eyes


final.vi
File Edit View Project Operate Tools Window Help
Select an Annulus
OK
Cancel
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software
Evaluation
start

D:\Finaleee\project2\Photo\photos\both eyes.PNG

0
Signed 16-bit imag
X: 208
Y: 228

Fig 9.1(g) Processed Image

Here the threshold range given was- lower value=5, upper value as 255. Here as seen above the edges of face and eye with the glint is obtained. Thus we have got edges that are not of importance for us when the threshold range was in the above range.

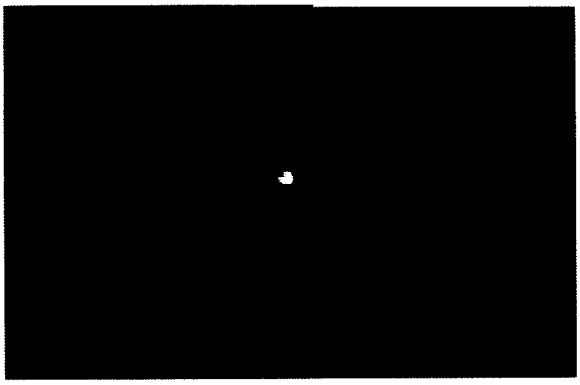


Fig 9.1(h) Image of Right eye

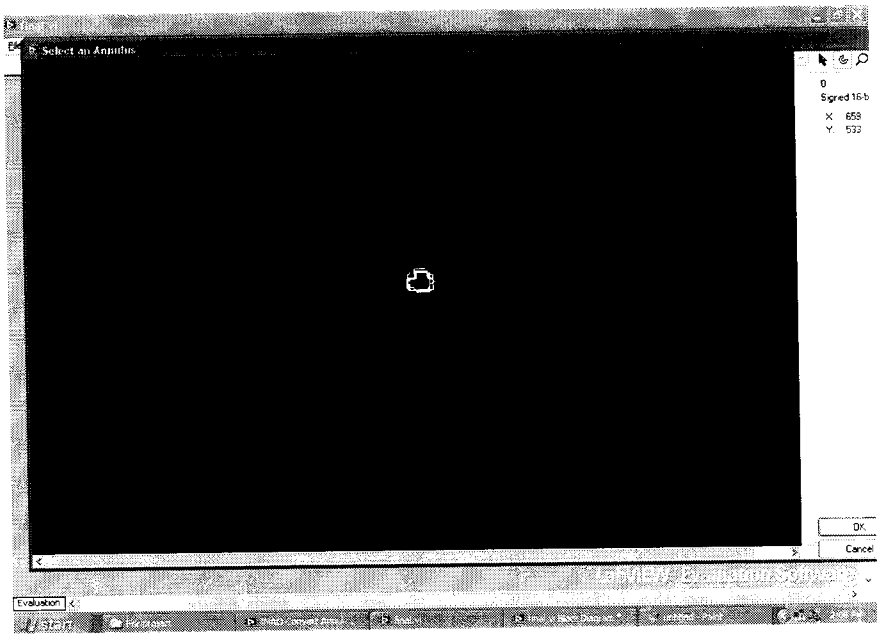
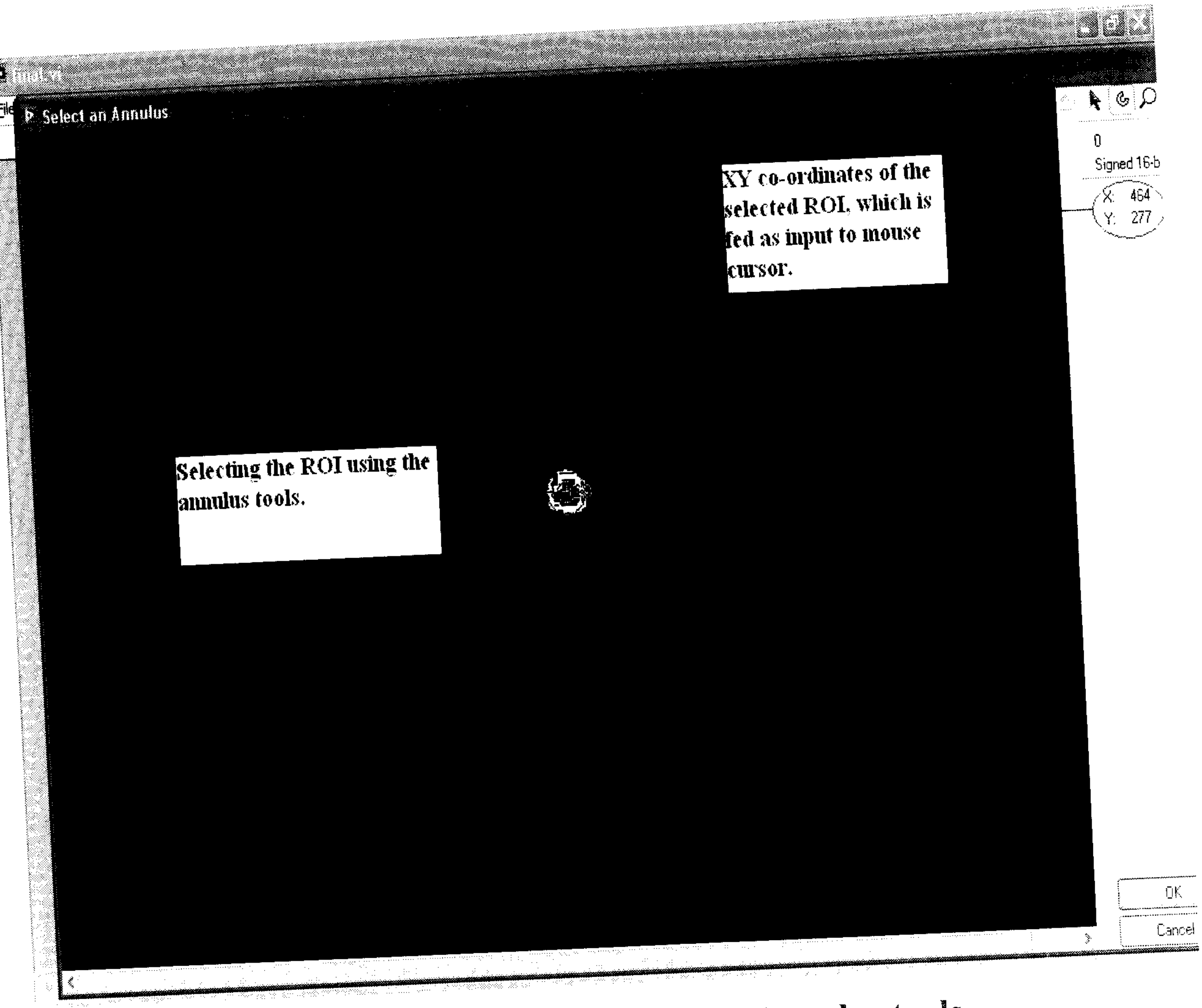


Fig 9.1(i) Glint obtained after processing

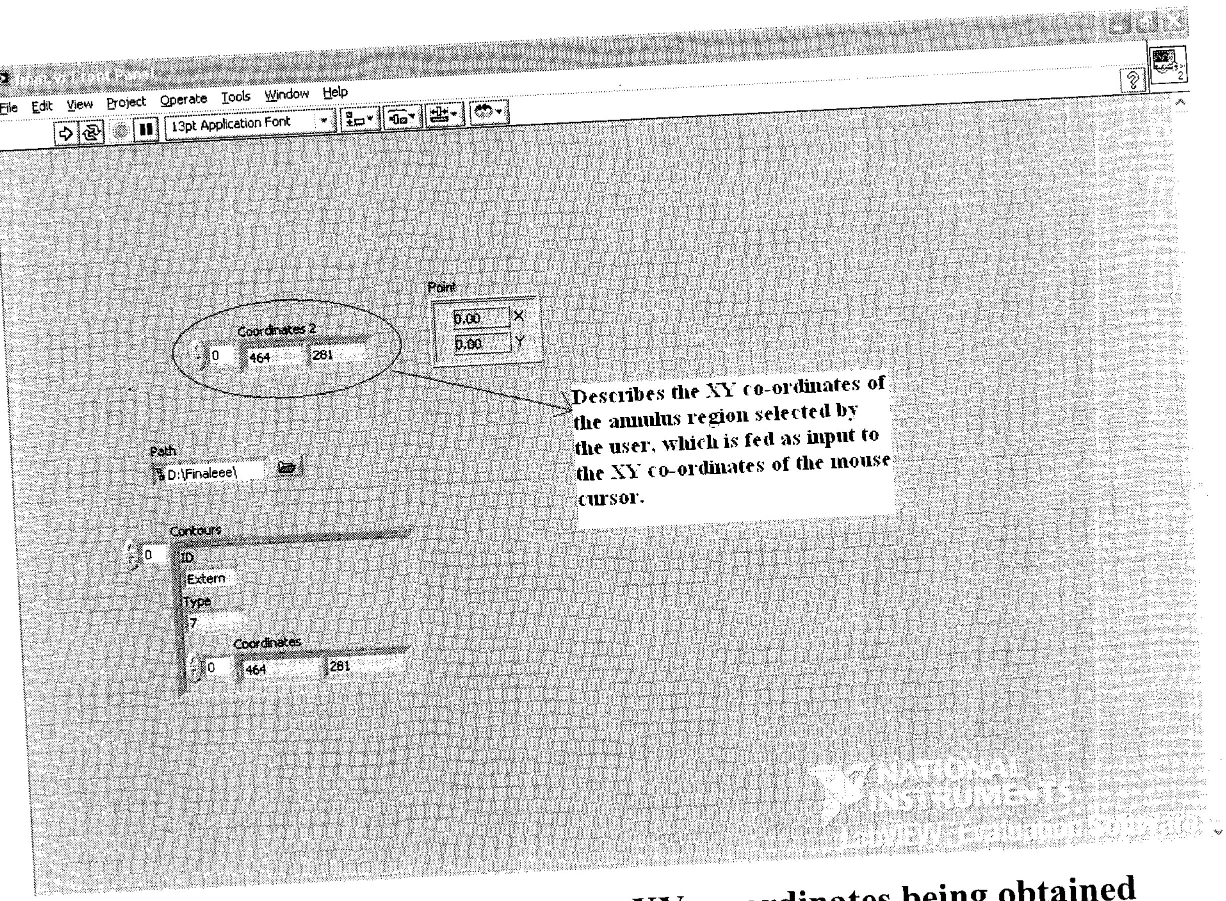
Here the threshold range given was, Lower value= 30, upper value =200. Thus the required glint of the eye was obtained for us in this threshold range. After experimenting we got the required glint of the eye alone in the range of, Lower value= 25-35 and upper value= 200-220.



final.vi
Select an Annulus
XY co-ordinates of the selected ROI, which is fed as input to mouse cursor.
Selecting the ROI using the annulus tools.
0
Signed 16-b
X: 464
Y: 277
OK
Cancel

Fig 9.1(j) Selection of ROI using Annulus tools

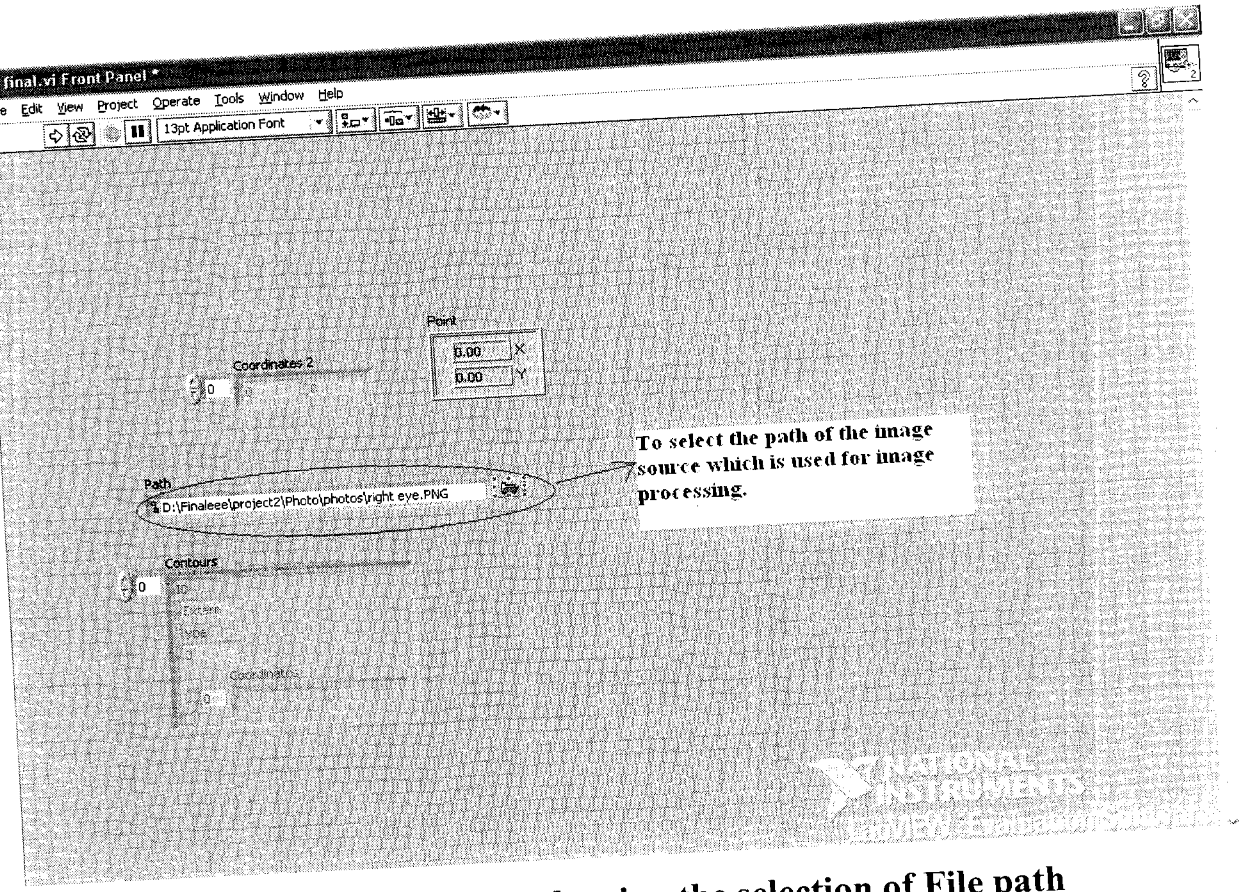
As seen above, after getting the required glint of the eye, we have selected the glint as the annulus region which is converted to ROI and the XY co-ordinates obtained when selecting the glint as shown above is fed as input to the XY co-ordinates of the mouse cursor.



Describes the XY co-ordinates of the annulus region selected by the user, which is fed as input to the XY co-ordinates of the mouse cursor.

Fig 9.1(k) Front panel showing XY co-ordinates being obtained

As seen above, the front panel indicates the user the current location of the mouse pointer after image processing. The XY coordinates correspondence to the position of the glint of the eye or the light.


final.vi Front Panel *
Edit View Project Operate Tools Window Help
13pt Application Font
Point
X
Y
Coordinates 2
Path
D:\Finaleee\project2\Photo\photos\right eye.PNG
Contours
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software

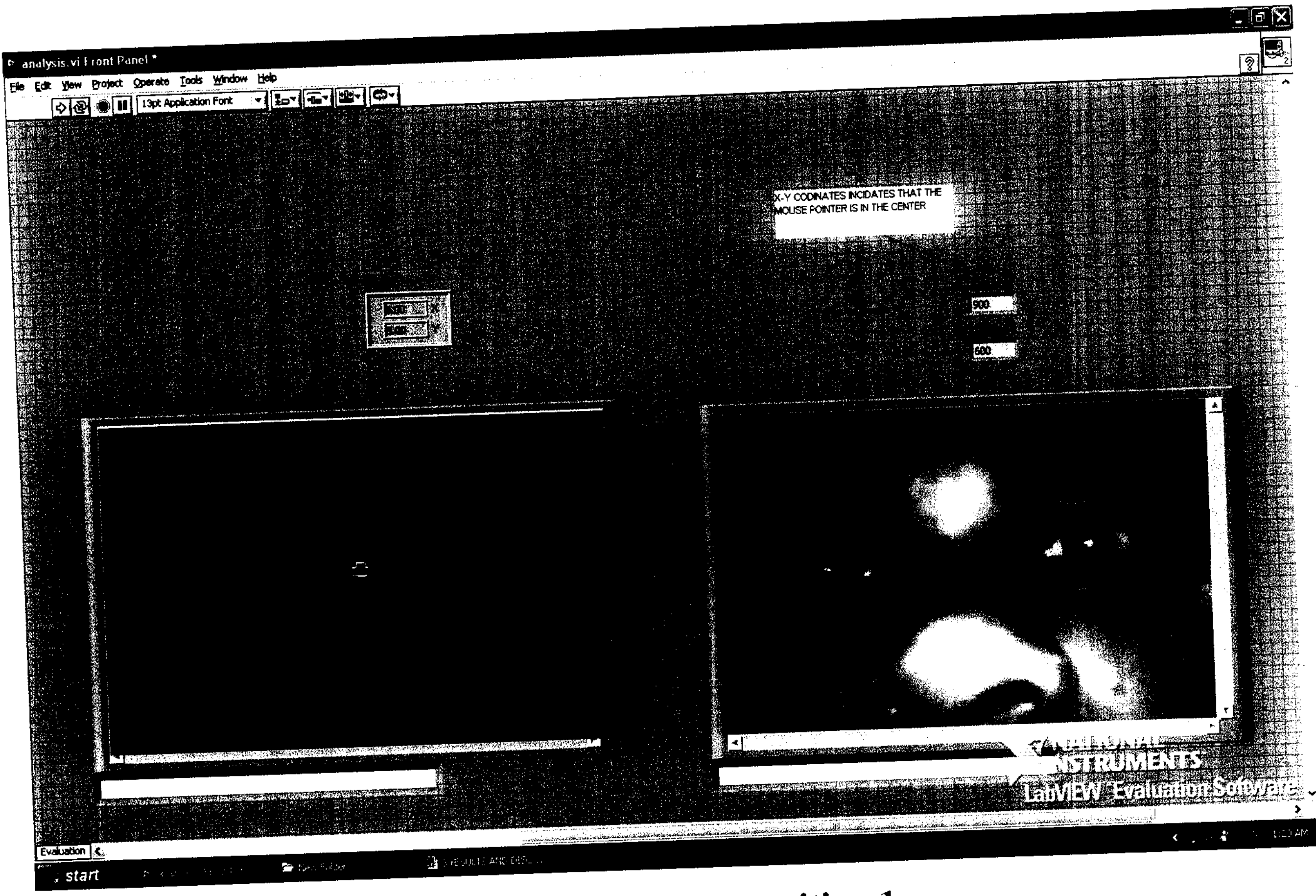
To select the path of the image source which is used for image processing.

Fig 9.1(l) Front panel showing the selection of File path

As seen above, the front panel indicates the user the location or the path of the image stored in the system memory.

9.2 MOVEMENT OF THE MOUSE POINTER

9.2.1 POSITION 1


analysis.vi Front Panel *
File Edit View Project Operate Tools Window Help
13pt Application Font
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software
Evaluation
start

X-Y CODINATES INCIDATES THAT THE MOUSE POINTER IS IN THE CENTER

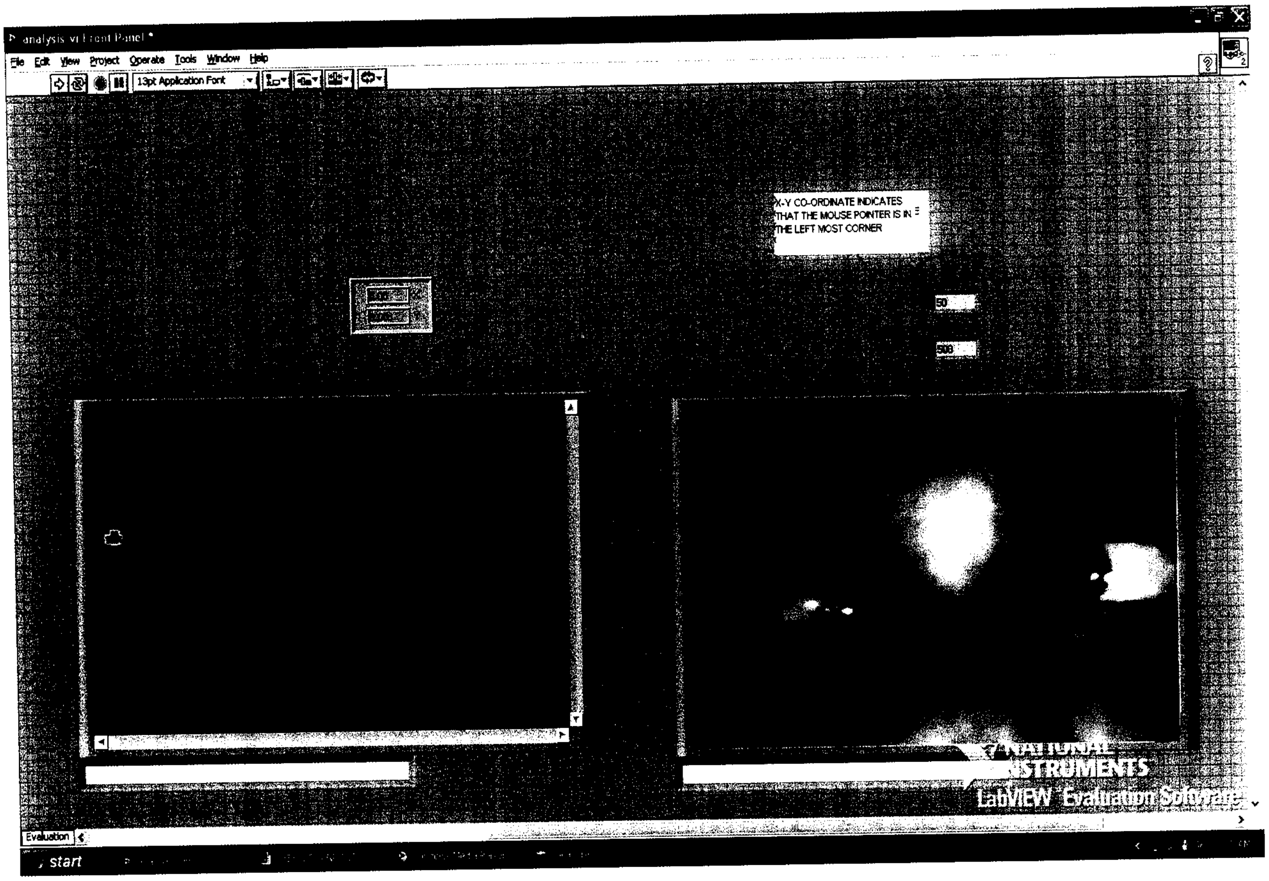
900

600

Fig 9.2.1 Mouse pointer –position 1

As seen above, the mouse pointer is in the centre of the display screen corresponding to the glint obtained through webcam.

9.2.2 POSITION 2


analysis.vi Front Panel *
File Edit View Project Operate Tools Window Help
13pt Application Font
Evaluation
start
NATIONAL INSTRUMENTS
LabVIEW Evaluation Software

X-Y CO-ORDINATE INDICATES THAT THE MOUSE POINTER IS IN THE LEFT MOST CORNER

50

503

Fig 9.2.2 Mouse pointer –position 2

As seen above, the mouse pointer is in the leftmost corner of the display screen corresponding to the glint obtained through webcam.

CONCLUSION

If this project is implemented, any human can control the position of the mouse pointer using the eye ball or using any other objects like light, pen and soon. Some of the advantages of implementing this project are listed below:

- It aids the Physically Challenged and the Tetraplegic people to use the mouse with ease.
- Reduce the dependency on hardware.
- Controlling the cursor by means of eyeball.
- Controlling the cursor by means of any objects like light, pen and soon.
- Provide the method computationally efficient and cost effective.
- Use of the computer by physically challenged people gets increased.

FUTURE EXPANSION

This project can be further enhanced as:

- Left click, right click can be included to make it more user-friendly.
- Up-down and left-right scrolling can be implemented.
- In future, it will lead them to do most of the normal things in their house, interacting with it by means of either remote controls or computers, directly from their bed or their wheel chair.
- They can open and close doors and windows, switch the appliances on and off, write a letter, use a PC, and so on.

REFERENCE

1. Christopher G. Relf, Image acquisition and processing with LabVIEW, Volume 1.
2. Jeffrey Travis, LabVIEW for Everyone: Graphical Programming Made Easy and Fun (3rd Edition)
3. Peter A. Blume, THE LabVIEW Style Book.
4. Rick Bitter, LabView: Advanced Programming Techniques, SECOND EDITION
5. IMAQ Vision for Labview User Manual, National Instruments
6. www.ni.com/labview
7. http://forums.ni.com/ni/board
8. http://www.pages.drexel.edu/~rs429/dasl/basicIMAQ/index.html