MIRROR

(Vision Based Human Computer Interaction)

A PROJECT REPORT

Submitted By:

VISHNU.S

VIVEK MOHAN

SUDHEESH V SOMAN

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLGY

in

COMPUTER SCIENCES & ENGINEERING

SCHOOL OF ENGINEERING

COCHIN UNIVERSITY OF SCIENCE &TECHNOLOGY

KOCHI- 682022

APRIL, 2012

DIVISION OF COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

COCHIN UNIVERSITY SCIENCE & ENGINEERING

Kochi-682022

CERTIFICATE

Certified that this is a bonafide record of the project work titled

MIRROR

(Vision Based Human Computer Interaction)

Done by

VIVEK MOHAN (12100096)

VISHNU.S (12100086)

SUDHEESH V SOMAN (12100075)

of VI semester Computer Science & Engineering in the year 2012 in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science & Engineering of Cochin University of Science & Technology

Dr.David Peter S
Head of Division

Mrs.Preetha S
Project Guide

ACKNOWLEDGEMENT

We like to thank the Almighty without whom we could not have completed this project successfully .We would like to also convey our heartfelt thanks to our project guide Preetha.S, Department of Computer Science SOE,CUSAT, for her support and guidance throughout the project. The support and guidance helped us in understanding the way we handled the project.

We would also like to thank all the members of the Department of Computer Science for their while hearted valuable support and encouragement throughout our project.

Last but not the least we would like to thank our parents, who taught us the value of hard work by their own example and friends who encouraged us by giving motivation to complete the project.

Vivek Mohan

Vishnu.s

Sudheesh V Soman

Dept. Computer Science,

School of Engineering,

CUSAT-22

ABSTRACT

Mirror (Vision Based Human Computer Interaction)

Purpose:

This is an application that is mainly intended implement gesture based implementation of mouse

Scope:

The project can be implemented in personal computers to provide cursor control using hand movement. This implementation gives a new experience of computing .Fingers are tracked using color bands and optical flow tracking algorithms

MODULES:

The project is mainly dividing into following modules/functions.

1. Color Detection Module

2. Tracking Module

3. Mouse Control Module

LIST OF FIGURES

Fig 4.1) Level zero DFD for Tracking and processing 19

Fig 4.2) DFD for Tracking of Different types of Point 20

Fig 4.3) DFD for Library Support 21

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
	ACKNOWLEDGEMENT	I
	ABSTRACT	II
	LIST OF FIGURES	III
1	**INTRODUCTION**	**1**
	1.1 Objectives	1
	1.2 Modules	2
	1.2.1 Frame Extraction	2
	1.2.2 Color Space Analysis	3
	1.2.3 Mouse Control	3
2	**SYSTEM STUDY**	**5**
	2.1 Introduction	5
	2.2 Feasibility study	5
	2.2.1 Economic Feasibility	5
	2.2.2 Technical Feasibility	6
	2.2.3 Behavioral Feasibility	6
3	**SYSTEM ANALYSIS**	**7**
	3.1 Existing System	7
	3.1.1 Problems with existing system	7
	3.2 Proposed System	7
	3.2.1 Advantages	8
4	**SYSTEM DESIGN**	**9**
	4.1 Programming Language Specification	9
	4.2 Algorithms	11

4.2.1 Tracking 11
4.2.1.1 Lucas-Kanade Method 11
4.2.1.2 Color Space Detection 16
4.3 Data Flow Diagrams 19
4.3.1 LEVEL-0 19
4.3.2 LEVEL-1 20
4.3.3 LEVEL-2 20
5 SYSTEM DESCRIPTION 21
5.1 System Requirement 21
5.1.1 SOFTWARE CONFIGURATION 21
5.1.2 HARDWARE SPECIFICATION 21
5.2 SCREENSHOTS 22
5.3 Essential OpenCv Library Functions Used 26
5.3.1 Lucas-Kanade code 26
6 SYSTEM TESTING 29
6.1 System Testing 29
6.2 Types of testing 29
6.2.1 White box Testing 29
6.2.2 Black box Testing 29
6.2.3 Unit Testing 30
6.2.4 Integration Testing 30
6.2.5 Validation Testing 30
6.2.6 System Testing 30
6.2.7 Output Testing 30
6.2.8 User Acceptance Testing 31
7 CONCLUSION AND FUTURE SCOPE 33
REFERENCES 35

CHAPTER 1

1. INTRODUCTION

This project is designed to provide virtual mouse which works by tracking hand motion through webcam. The highlight of this project is the use of webcam to track user hand and according to the position and gestures of hand implement appropriate mouse functions.

1.1 Objectives:

The main objective of our project is to successfully track user hand without any conflicts caused by noise or other image impairments. Fingers are tracked using color bands and optical flow tracking algorithms so the same color pixel(same as color bands) that might come in the background should not defect tracking. Image can be stored in memory in different color spaces. In order to processing on those image data and track color distribution lots of algorithms and equations are developed and currently available .To track a points optical flow that's the path it moves from frame to frame we here use Lucas Kanade method

Video Recognition has been implemented for vast range of applications such - as face recognition, extraction of text from pictures, augmented reality etc. here we are trying to implement functions of mouse through video recognition.OpenCV was designed for computational efficiency and with a strong focus on realtimeapplications. OpenCV is written in optimized C and can take advantage of multicore processors. One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly.

1.2 Modules:

1. **Frame Extraction**
2. **Color Space Analysis&Tracking**
3. **Mouse Control**

1.2.1 Frame Extraction:

The Webcam is used for capturing video. The users hand movement and color band tracking are performed on this video. The software reads this visual by taking frame by frame pictures very fast .each frame contains an instance of movement. This frame is a metrics of color values alpha-RGB space. This frame can be converted into a mirror image if the webcam is placed in front of user, this option is enabled by user for better controlling of mouse. The frame size and resolution vary with webcam model. Thus this frame is x, y coordinates are calculated with respect to Screen Resolution. The webcam resolution by default is 640X420 this can be tweaked to 1200x1024 as the standard model support. The frame extracted in Alpha-RGB form may have to be converted in to other color space for better processing since RGB is not good color space for computer processing as it is for humans. The other Format mainly used here is GRAY SCALE, HSV.HSV means Hue-Saturation-Value, where the Hue is the color. And since color is not an easy thing to separate or compare, Hue is often represented as a circular angle (between 0.0 to 1.0 when stored as floats). Being a circular value means that 1.0 is the same as 0.0. For example, a Hue of 0.0 is red, a Hue of 0.25 would be green, a Hue of 0.5 is blue, a Hue of 0.75 is pink, and a Hue of 1.0 would be the same as a Hue of 0.0 which is red (again). Saturation is the greyness, so that a Saturation value near 0 means it is dull or grey looking whereas as a Saturation value of 0.8 might be a very strong color (eg: red if

Hue is 0). And Value is the brightness of the pixel, so 0.1 is black and 0.9 is white

1.2.2 Color Space Analysis & Tracking:

In Color based Tracking the software analyses the captured frame and the image is stored in HSV color space. Then we will specify the hsv range of color we want to track . A function defined will extract the pixel in that range. This extracted image will be analyzed whether it contain traceable amount of pixels and then tracks it.The position of a point in the current frame(with respect to previous one) is calculated using lucas-kanade algorithm . The next time we will only analyse partial portion of image for color with respect to current optical flow of pixel.

1.2.3 MOUSE CONTROL

This part provides all the procedures for performing mouse operations such as click, double click, right click, drag &drop Move cursor to a desired point etc. This Module is developed to work on Linux Systems. It's implemented with the help of X11 Library.

CHAPTER 2

2. SYSTEM STUDY

2.1 Introduction

Presently cursors are controlled by mouse and touchpads. Computers main input devices are keyboard, mouse. This Physical Devices Is Essential for the working of Computer. At the same time it causes lack if portability and unwanted wires. So we understood if it is possible to implement the functions of these devices in a better way then we could provide the user with completely new and efficient way of computing which opens a path for vast advance application.

2.2 Feasibility Study

- Economic feasibility
- Technical feasibility
- Behavioral feasibility

2.2.1 Economic feasibility

A fully matured version of this project has the power to provide the functions of all standard input devices available now. This causes reduction in computer parts, interfacing units, accessories. For Developing this project the only feasible component required is a standard quality camera (webcam).and may be some proper lighting. All the software components are Open-Source, thus software components are free this is a main advantage in matter of economic feasibility.

2.2.2 Technical feasibility

Computer Vision based technologies are very advanced and vast. So technical personals required for the development process requires lot of background knowledge and need to spend more time in understanding and studying the concepts of many different algorithms and techniques. Thus Time required for research is high.

2.2.3 Behavioral Feasibility

Users have been using mouse in computing for decades. So another technology developed for replacing this should give more freedom, efficiency, usability, understandability. These are risks that have to be dealt in Design Phase.

CHAPTER 3

3. SYSTEM ANALYSIS

Here in the **Cursor Control through Web Camera** project, a detailed study of existing system is carried along with all the steps in system analysis. An idea for creating a better project was carried and the next steps were followed.

3.1 Existing System

The Existing System Uses Input Devices such as mouse, Touch pads to control the movement of cursor though the vision based human computer interaction is still on research level.

3.1.1 Problems with Existing System

1. Less Flexible
2. Extra Hardware Cost
3. Wired or User needs to physically operate that device all the time
4. User can't move away from computer and operate (operate while standing)

3.2 Proposed System

The aim of proposed system is to develop a system of improved facilities. The software is suitable for better human computer interaction.

3.2.1 Advantages of the Proposed System

The system is very simple in design and to implement. The system requires very low system resources and the system will work in almost all configurations. The system is designed to work with Open source platform. It has the Following features:

- Ensure tracking
- Minimize computation
- No strain for user.
- Greater efficiency
- Better service

CHAPTER 4

4. SYSTEM DESIGN

4.1 Programming Language Specification: Python,Opencv Library

Python is a remarkably powerful dynamic programming language that is used in a wide variety of application domains. Python is often compared to Tcl, Perl, Ruby, Scheme or Java. Some of its key distinguishing features include:

- Very Clear, Readable Syntax
- Strong Introspection Capabilities
- Intuitive Object Orientation
- Natural Expression Of Procedural Code
- Full Modularity, Supporting Hierarchical Packages
- Exception-Based Error Handling
- Very High Level Dynamic Data Types
- Extensive Standard Libraries And Third Party Modules For Virtually Every Task
- Extensions And Modules Easily Written In C, C++ (Or Java For Jython, Or .NET Languages For Ironpython)
- Embeddable Within Applications As A Scripting Interface

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real time computer vision, developed by Intel and now supported by Willow Garage. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. If the library finds Intel's Integrated Performance Primitives on the system, it will use these proprietary optimized routines to accelerate itself.

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and Recognition
- Stereopsis Stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking

The OpenCV functions that allow us to interact with the operating system, the file system, and hardware such as cameras are collected into a

library called HighGUI (which stands for "high-level graphical user interface"). HighGUI allows us to open windows, to display images, to read and write graphics-related files (both images and video), and to handle simple mouse, pointer, and keyboard events. We can also use it to create other useful doodads like sliders and then add them to our windows. If you are a GUI guru in window environment of choice, then you might find that much of what HighGUI off ers is redundant. Yet even so you might find that the benefit t of cross-platform portability is itself a tempting morsel.

Python have been updated to meet the development needs. new language, These languages leverage the functionality of the OpenCV library, which provides access to key technologies that simplify the development of Vision Based Systems such as Robots, computer vision technologies, Graphics applications.

4.2 ALGORITHMS

4.2.1 Tracking

4.2.1.1 Lucas-Kanade Method

The Lucas-Kanade (LK) algorithm [Lucas81], as originally proposed in 1981, was an attemptto produce dense results. Yet because the method is easily applied to a subset of the points in the input image, it has become an important sparse technique. The LK algorithm can be applied in a sparse context because it relies only on local information that is derived from some small window surrounding each of the points of interest. This is in contrast to the intrinsically global nature of the Horn and Schunck algorithm (more on this shortly). The disadvantage of using small local windows in Lucas-Kanade is that large motions can move points outside of the local window and thus become impossible for the algorithm to find. This problem led to development

of the "pyramidal"LK algorithm, which tracks starting from highest level of an image pyramid (lowest detail) and working down to lower levels (finer detail). Tracking over image pyramids allows large motions to be caught by local windows.

The basic idea of the LK algorithm rests on three assumptions.

1. *Brightness constancy*. A pixel from the image of an object in the scene does not

change in appearance as it (possibly) moves from frame to frame. For grayscale images

(LK can also be done in color), this means we assume that the brightness of a

pixel does not change as it is tracked from frame to frame.

2.Temporal persistence or "small movements".: The image motion of a surface patchchanges slowly in time. In practice, this means the temporal increments are fast enough relative to the scale of motion in the image that the object does not move much from frame to frame.

3. Spatial coherence. Neighboring points in a scene belong to the same surface, have similar motion, and project to nearby points on the image plane.

The first requirement, brightness constancy, is just the Requirement that pixels in one tracked patch look the same over time:

f (x, t)≡ I(x(t), t)= I(x(t +dt), t +dt)

$$I(x+\Delta x, y+\Delta y, t+\Delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t +$$

H.O.T.

From these equations it follows that:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

or

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0$$

Which results in

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0$$

Where V_x, V_y are the x and y components of the velocity or optical flow of $I(x, y, t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at (x, y, t) in the corresponding directions. I_x, I_y and I_t can be written for the derivatives in the following.

Thus:

$$I_x V_x + I_y V_y = -I_t$$

or

$$\nabla I^T \cdot \vec{V} = -I_t$$

This is an equation in two unknowns and cannot be solved as such. This is known as the *aperture problem* of the optical flow algorithms. To find the

optical flow another set of equations is needed, given by some additional constraint. All optical flow methods introduce additional conditions for estimating the actual flow.

The Lucas-Kanade method assumes that the displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the point *p* under consideration. Thus the optical flow equation can be assumed to hold for all pixels within a window centered at *p*. Namely, the local image flow (velocity) vector must satisfy

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

where $q_1, q_2, \ldots, q_n$ are the pixels inside the window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the partial derivatives of the image I with respect to position *x*, *y* and time *t*, evaluated at the point q_i and at the current time.

These equations can be written in matrixform $Av = b$, where

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \qquad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

This system has more equations than unknowns and thus it is usually over-determined. The Lucas-Kanade method obtains a compromise solution by the least squares principle. Namely, it solves the 2×2 system

$A^T A v = A^T b$or

$\mathrm{v} = (A^T A)^{-1} A^T b$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i w_i I_x(q_i)^2 & \sum_i w_i I_x(q_i) I_y(q_i) \\ \sum_i w_i I_x(q_i) I_y(q_i) & \sum_i w_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i w_i I_x(q_i) I_t(q_i) \\ -\sum_i w_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

with the sums running from i=1 to n.The matrix A^T A is often called the structure tensor of the image at the point p.

The plain least squares solution above gives the same importance to all *n* pixels q_i in the window. In practice it is usually better to give more weight to the pixels that are closer to the central pixel *p*. For that, one uses the weighted version of the least squares equation,

$$A^T W A v = A^T W b$$

or

$$\mathrm{v} = (A^T W A)^{-1} A^T W b$$

Where W is an *n*×*n* diagonal matrix containing the weights $W_{ii} = w_i$to be assigned to the equation of pixel q_i. That is, it computes

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i w_i I_x(q_i)^2 & \sum_i w_i I_x(q_i) I_y(q_i) \\ \sum_i w_i I_x(q_i) I_y(q_i) & \sum_i w_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i w_i I_x(q_i) I_t(q_i) \\ -\sum_i w_i I_y(q_i) I_t(q_i) \end{bmatrix}$$

The weight $w_i$is usually set to a Gaussian function of the distance between q_i and *p*.

4.2.1.2 Color Space Detection

Blob detection is a fast and simple method that can be used for many machine vision tasks, such as tracking a red ball, finding a blue marker or detecting a person's skin To find colored blobs, you should convert your color image from BGR to HSV format so that the colors are easier to separate. HSV format, each pixel in image is represented using three digits varying from 0-255.In HSV H for Hue,S for Saturation, V for Value or Colorfulness. Each color unique color has a unique hue valu and saturation and value is used to tell how much of that color is contained in that image. In a three channel image maximum number of colors that can be presented is 255^255^255 that about six million . So Processing of images in RGB is not efficient for computers . HSV in this case plays a simple and efficient way. "cvThreshold()" on a Hue image (single-channel image) to find the blue pixels. But since black, white and grey can also have any Hue values, you

should also threshold the Saturation and Value components of the HSV image to ignore black, white and grey. Depending on how bright or dark or dull you expect your objects and your background environment to be, you might consider for example: anything with a "Value" (brightness) below 60 to be too dark (black), and anything with a Saturation below 30 to be too dull (grey or white). And after you have got your program working, you should play around with these HSV thresholds until you are happy with the results in your images, because they will vary a lot, and you might want to use a more complicated formula such as ignoring anything that has V above 200 if it also has S below 50 (white), etc.More powerful functions are the cvCmp, cvCmpS, inRange and inRangeS function. The functions ending in an S let you compare the image against a particular value (like 20). The others let you compare the image against another image (so you could have different comparison values for different pixels).

The cvCmp function lets you specify the type of comparison (greater than, greater or equal, etc). Its syntaxes are:

```
cvCmp(src1, src2, dst, cmp_op);

cvCmpS(src, value, dst, cmp_op);
```

cmp_op can take these values:

- CV_CMP_EQ – equal to
- CV_CMP_GT – greater than
- CV_CMP_GE – greater or equal
- CV_CMP_LT – less than
- CV_CMP_LE – less or equal
- CV_CMP_NE – not equal to

The inRange function lets you specify a range of values (a minimum and a maximum) which is converted to a 255. Any value outside the range is set to 0. Its syntaxes are:

cvInRange(src1, lowersrc, uppersrc, dst);

cvInRangeS(src, scalarLower, scalarHigher, dst);

Pretty much self-explanatory. This function can be used to threshold out an HSV

image.where the H channel holds the colour image.

4.3 DATA FLOW DIAGRAMS

4.3.1 LEVEL-0

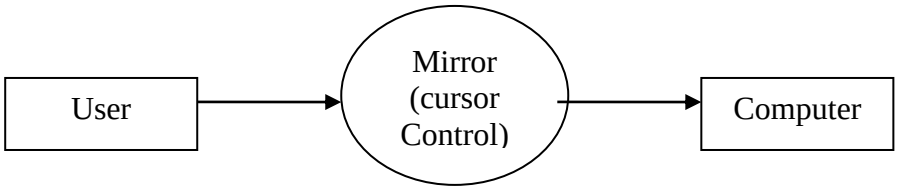

Fig 4.1 Level zero DFD for Tracking and processing

4.3.2 LEVEL-1

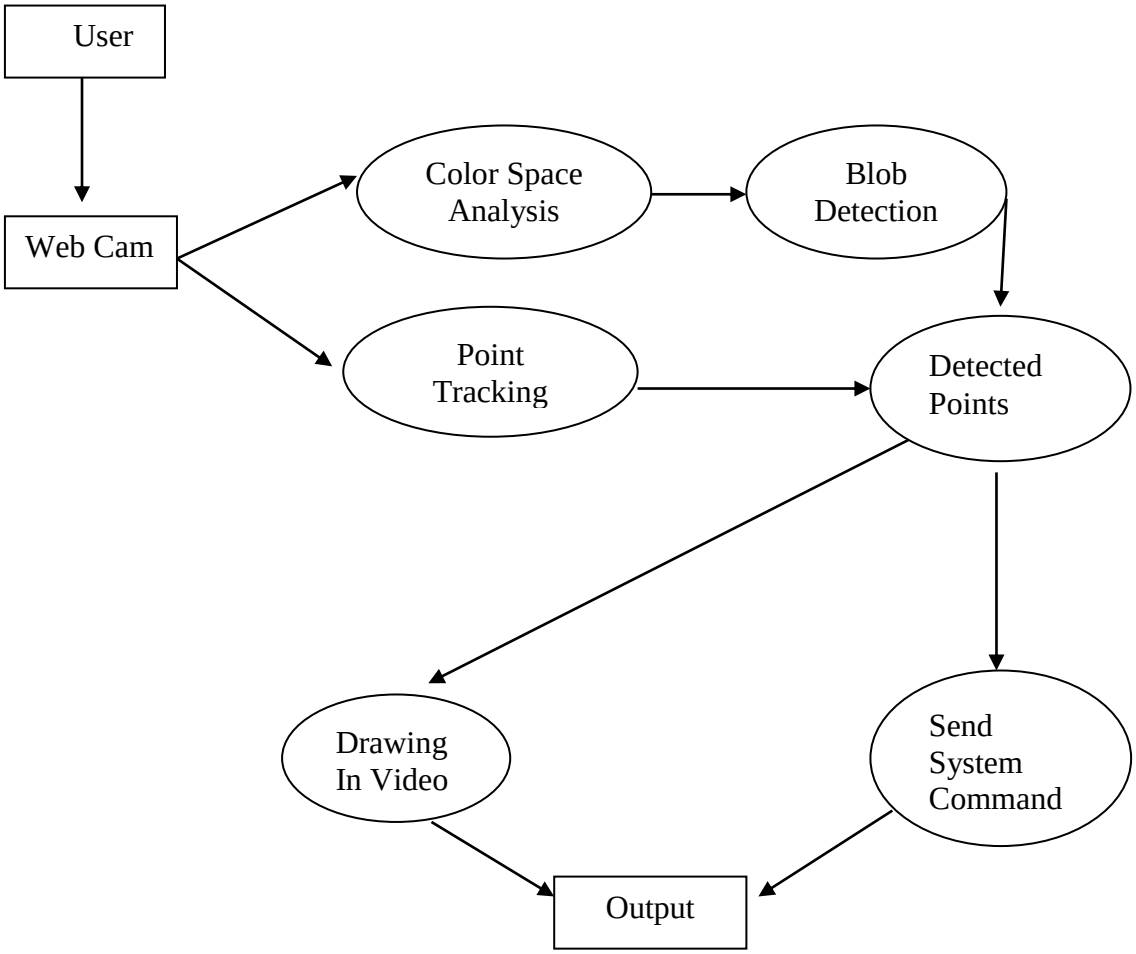

Fig 4.2 DFD for Tracking of Different types of Points

4.3.3 LEVEL-2

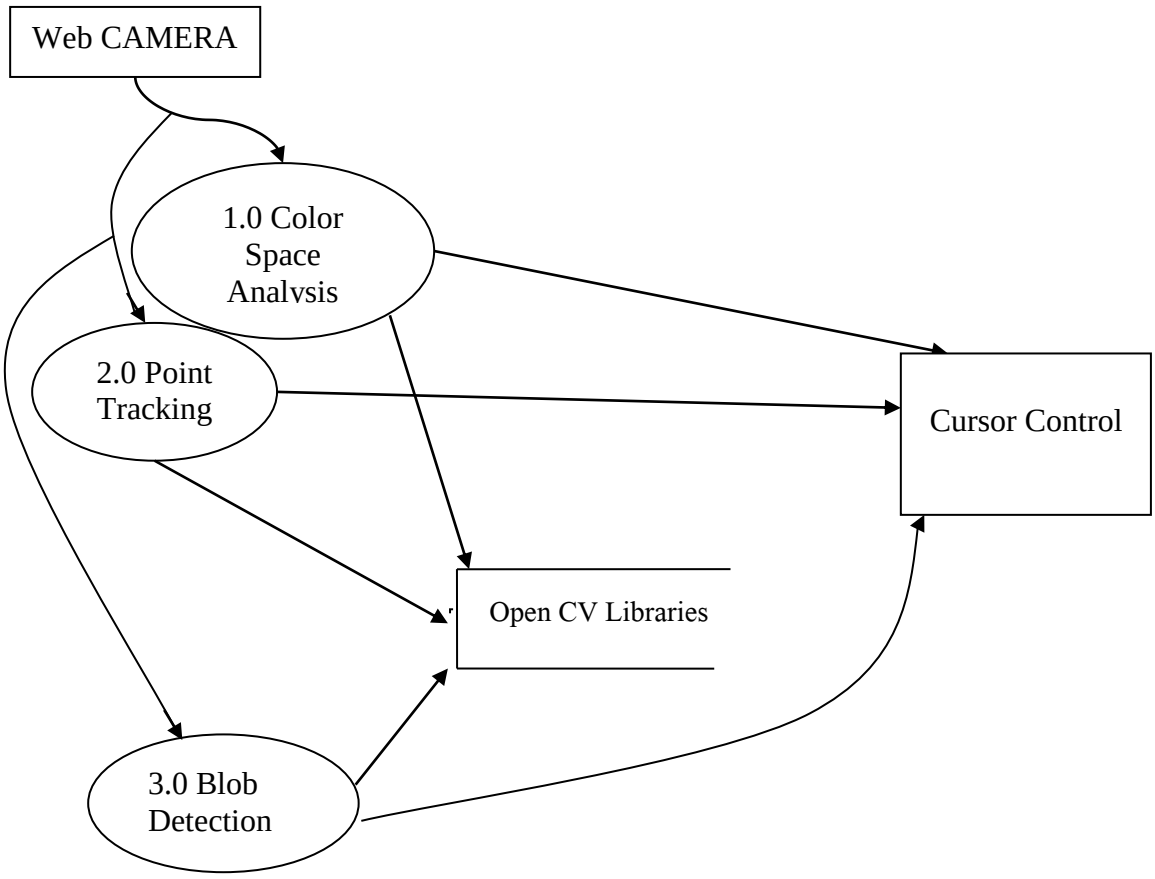

Fig 4.3 DFD for Library Support

CHAPTER 5

5. SYSTEM DESCRIPTION & IMPLEMENTATION

5.1 System Requirement

A system requirements specifications (SRS) is a complete description of the behavior of the system to be developed .It includes a set of use cases that describe all of the interactions that the users will have with the software.

5.1.1 SOFTWARE CONFIGURATION

Development Platform	:	Linux
Programming Language	:	Python/C
Library Support	:	OpenCV

5.1.1 HARDWARE SPECIFICATION (Recommended)

System	:	IBM-Compatible PC
Processor	:	Intel Pentium D or Greater
Speed	:	2.4 GHz.
Memory	:	512 MB RAM
Hard Disk Drive	:	20 GB
Monitor	:	15``SVGA Digital Color Monitor
Webcam	:	1 Standard Quality webcam

5.2 SCREENSHOTS

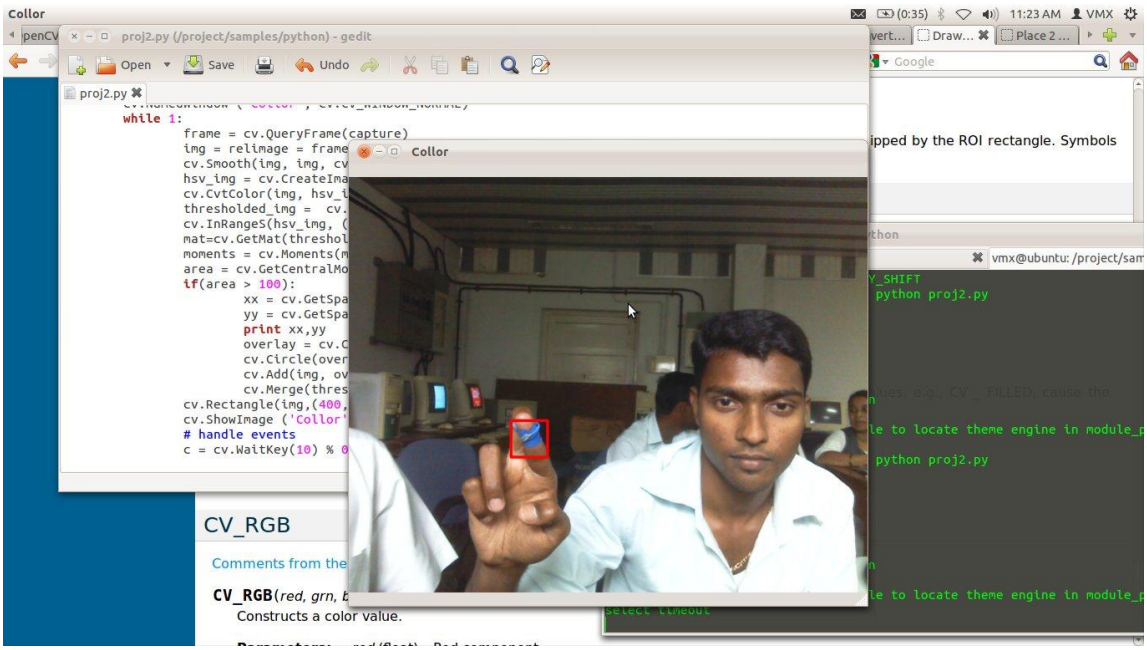

Fig 5.1.Screen shot of Cursor Control.

In The above Screenshot Shows a color band of blue detected using blob detection. After the color is once detected the centroid of this area is sent to l Lucas Kanade and then Lucas Kanade algorithm tracks that point but it doesn't search the entire image for tracking it creates a Region Of Interest in the First frame surrounding the detected color.

The above screen shot also shows distance between two tracked points determines left click and right click.

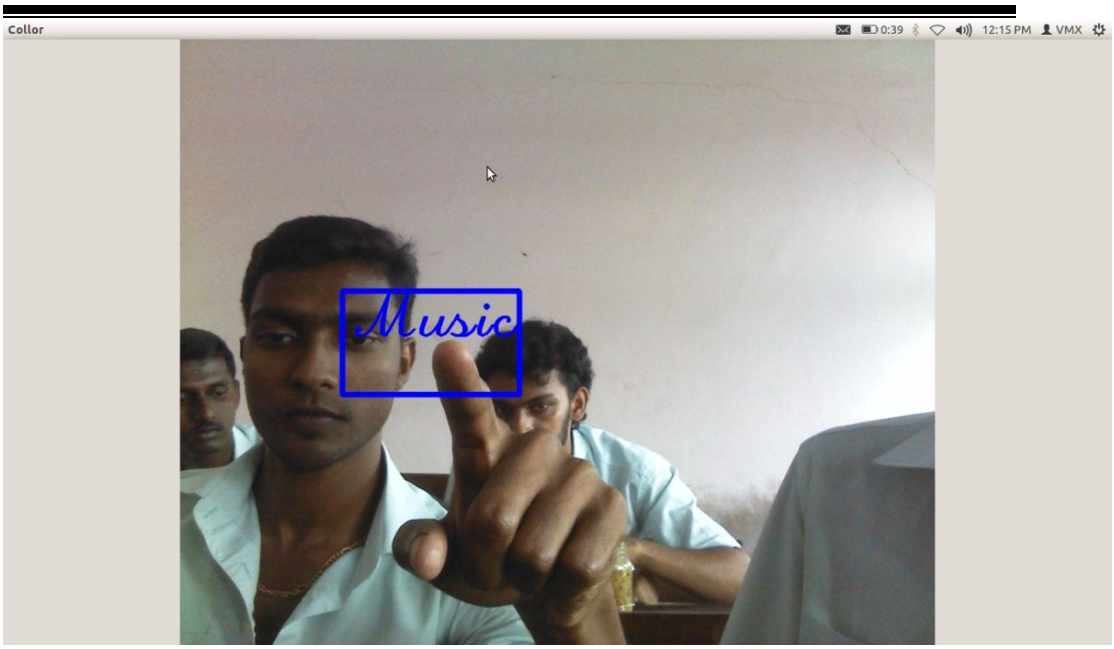

Fig 5.2 Screen Shot Of Implementing Touchscreen Apps.

The above Screen Shot shows the x,y coordinates of tracked points are used determine whether we have to start some Application.

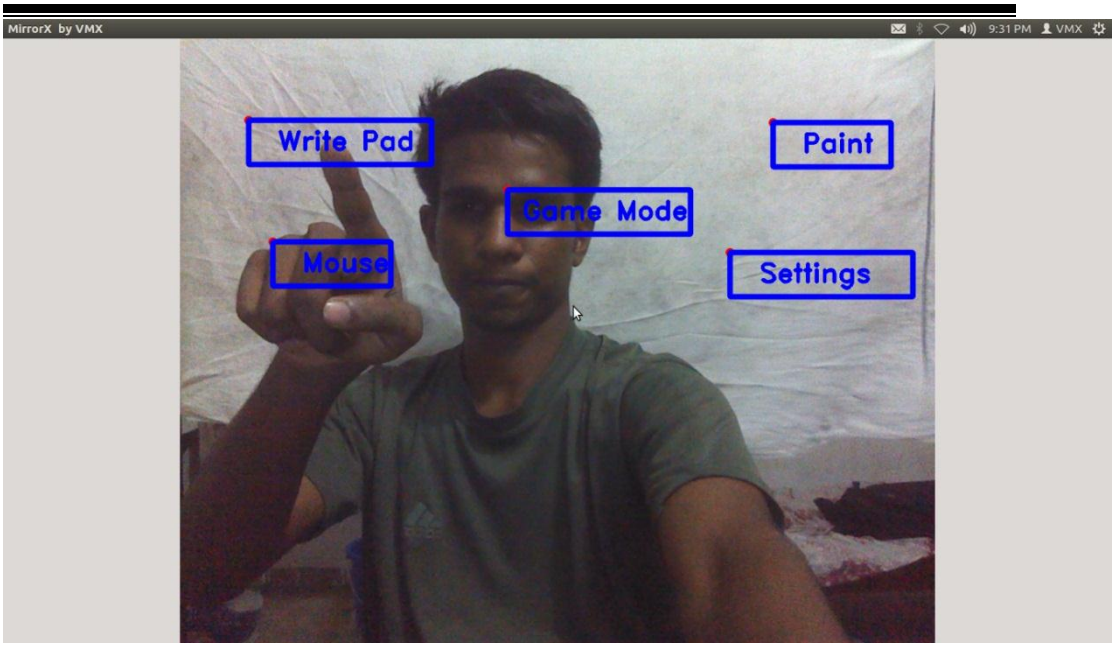

Fig 5.3 Screen Shot Of Mirror Home.

The above Screen Shot shows the Home screen of Mirror x.

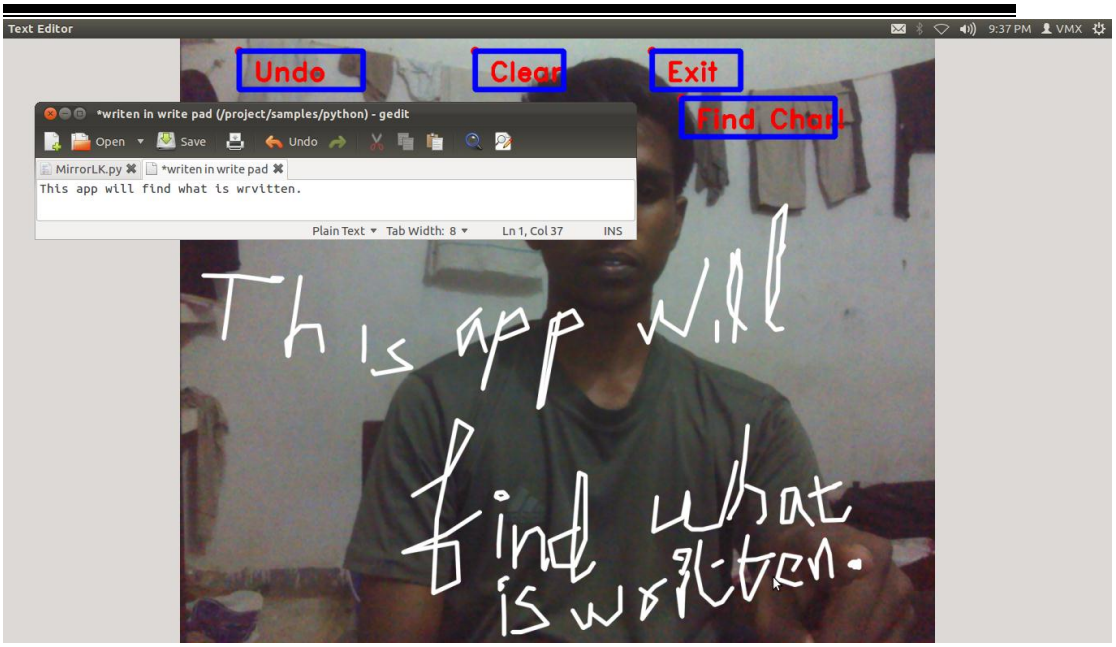

Fig 5.4 Screen Shot Of Implementing Write PAD.

The above Screen Shot shows the. Tracked movement coordinates are pushed in to stack and looking at the coordinates of handwriting the shape of drawing will be transferred to signature,this signature is looked up in our predefined dictary for letter shapes.

5.3 Essential OpenCv Library Functions Used:

5.3.1 Lucas-Kanade code

The routine that implements the no pyramidal Lucas-Kanade dense optical flow algorithms:

The following pyramid-based method is better for most situations most of the time.

Void CalcOpticalFlowPyrLK

(constCvArr imgA,*

constCvArr imgB,*

CvArr pyrA,*

CvArr pyrB,*

CvPoint2D32f featuresA,*

CvPoint2D32f featuresB,*

int count,

CvSizewinSize,

int level,

char status,*

float track_error,*

CvTermCriteria criteria,

int flags);

The first two arguments of cvCalcOpticalFlowPyrLK() are the initial and final images;both should be single-channel, 8-bit images. The next two arguments are buff ers allocated to store the pyramid images. The size of these buffers should be at least img.width mitigates the problems caused by violating our assumptions of small and coherent motion; the motion estimate from the preceding level is taken as the starting point for estimating motion at the next layer down. The array featuresA contains the points for which the motion is to be found, and featuresB is a similar array into which the computed new locations of the points from featuresA are to be placed; count is the number of points in the featuresA list. The window used for computing the local coherent motion is given by winSize. Because we are constructing an image pyramid, the argument level is used to set the depth of the stack of images. If level is set to 0 then the pyramids are not used. The array status is of length count; on completion of the routine, each entry in status will be either 1 (if the correspondingpoint was found in the second image) or 0 (if it was not). The track_error parameter is optional and can be turned off by setting it to NULL. If track_error is active then it is an array of numbers, one for each tracked point, equal to the difference between the patch around a tracked point in the first image and the patch around the location to which that point was tracked in the second image. You can use track_error to prune away points whose local appearance patch changes too much as the points move. The next thing we need is the termination criteria. This is a structure used by many OpenCV algorithms that iterate to a solution:

cvTermCriteria(int type, // CV_TERMCRIT_ITER, CV_TERMCRIT_EPS, or both intmax_iter, double epsilon);

Typically we use the cvTermCriteria() function to generate the structure we need. The first argument of this function is either CV_TERMCRIT_ITER or CV_TERMCRIT_EPS, which tells the algorithm that we want to terminate either aft er some number of iterations or whenthe convergence metric reaches some small value (respectively). The next two arguments set the values at which one, the other, or both of these criteria should terminate the algorithm. The reason we have both options is so we can set the type to CV_TERMCRIT_ITER |CV_TERMCRIT_EPS and thus stop when either limit is reached.

Finally, flags allows for some fi ne control of the routine's internal bookkeeping; it may be set to any or all (using bitwise OR) of the following.

CV_LKFLOW_PYR_A_READY: The image pyramid for the first frame is calculated before the call and stored inpyrA.

CV_LKFLOW_PYR_B_READY: The image pyramid for the second frame is calculated before the call and stored inpyrB.

CV_LKFLOW_INITIAL_GUESSES

The array B already contains an initial guess for the feature's coordinates when the routine is called.

CHAPTER 6

6. SYSTEM TESTING

6.1 System Testing

Since Our project is based on vision and the chances occuring.unwanted details in the frame is normal so project should developed in such a way that these background details should not arise against tracking.and the software should not fail on an external system. In order to make sure this we performed following tests.

6.2 Types of Testing

6.2.1 White Box Testing

W made sure the all the internal functions where working properly and the control flow instructions are transferring as intended. All feature tracking algorithms where tested. Intermediate values have been analyzed weather they are correct in each module.

6.2.2 Black Box Testing

We tested the program to meet whether the basic requirement, "Right output for right Input". So we Tested Running the program in Following cases

1. In a location where there wasn't sufficient lighting
2. Background contained too many colors
3. Other models of Webcam
4. Different Color Bands.

Though some test cases made problems with tracking in some cases. first and second cases were making unexpected output. Since in vision based

tracking lighting is the most essential part .we has incorporated noise reduction transforms on each frame to reduce the exceptions.

6.2.3 Unit Testing

We tested each Module independently and they were giving intended output for the proper input.Those Units were:

1.Mouse Control

2.Color based Tracking

3. Tracking Based on Lucas Kanade Algorithm

4. OnScreen Application Buttons

6.2.4 Integration Testing

We Interfaced each module to other and executed.The Programs were performing normally. Output From a module was successfully transferred to input of another Module.

Such as Cordinates of tracked points were transferred to mouse Control module Or System Command Module where inside these modules these points were considered as reference for decision making.

6.2.5 System Testing

The Software was tested on different linux operating systems and found stable.Diffrent models of Webcams were also tested. All standard exceptions were solved.

6.2.7 Output Testing

All the Desired Outputs were shown in screen according to hand movements and all the Gui and Drawing routines were suceesfully executed and output was shown in screen.

6.2.8 User Acceptance Testing

The User response was positive after using this new interactive computing it was found that they were actually delighted and the average feedback was ,discarding the anomalies existing in current video recognition technologies ,it is a highly useful application and new way of computing. The following features provided better usability

- Output Screen Design.
- On-Screen Widgets
- Virtual mouse

CHAPTER 7

7. CONCLUSION AND FUTURE SCOPE

This project was completed within the time span allotted. We are very satisfied to get this opportunity to do this project. All the knowledge we gained is full applied in the design of the mentioned system and application package is developer. All the suggestions forwarded in the software proposal have been completed. This system is developed in such a way that the modules developed in future can be linked easily to the system, without affecting the existing system, since it provides a hierarchical structure.

Top down programming approach has been adopted while developing the project; each task is divided into o separate modules. Hence modification and user-friendly and interactive. The performance of the system is provided efficiently.

The system was tested with all possible sample data and was found to have an effective planning of the functions or process with a high degree of accuracy and user friendliness.

To conclude this, We thank all people who help us to complete this project work successfully.

The Project is still on a prototype model. Lot of research is required to find the maximum potential of the application. Application is also highly prone to risk. Risk analysis and containment has to done perfectly. The successful

implementation of this concept will give a new experience of computing. Considering the fact webcams are cheaper than touchscreen and the area covered by both device differ significantly, by incorporating accuracy and convenience video recognition is the best choice for users

It is also possible to integrate the entire system by developing the existing software package to an intranet level application. This enables to control the geographical distributed organization units into one, which gives a far wide and consolidate view of the organization. New security methods should be developed to avoid the misuse of the system in a wide environment. The biometrics standard can be implemented for the purpose of high level of security.

Python 2.6 or above and Opencv are configurable and compliant to this new technology. In The future development the voice based data recognition module can be added. This reduces the typographical complexity and increases the high throughput by the end user. The voice recognition system can also be used as a message passing system to the end user as help. In the section of reports the inclusion of charts and graphs changes the entire appearance of reports to a new dimension. These are a few recommended future developments to the existing system.

The software is developed on Open Source Linux Platform which makes the system more reliable and compatible with the other environments.

APPENDIX -I

REFERENCES

BOOKS USED

[1] RobertLaganière (2011) , ' OpenCV 2 Computer Vision Application Programming Cookbook' - Packt Publishing Ltd.

[2] Gary Bradski ,and Adrian Kaehler (2008) 'Learning OpenCV' O'reilly Media.

[3] Alex Martelli ,Anna Martelli Ravenscroft, and David Ascher Mandriolli (2003) ,'Python Cookbook' , O'reilly Media.