# HAND GESTURE FOR SIMPLE USER INTERFACING

**A PROJECT REPORT**

***Submitted by***

**NANDHA KUMAR P R (21009104060)**

**SHYAM SUNDAR E H (21009104104)**

**VERGHESE KOSHY PUTHUKKERIL (21009104123)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**PANIMALAR ENGINEERING COLLEGE**

**ANNA UNIVERSITY :: CHENNAI 600 025**

**APRIL 2013**

# ANNA UNIVERSITY :: CHENNAI 600 025

## BONAFIDE CERTIFICATE

Certified that this project report **"HAND GESTURE USER INTERFACE SYSTEM"** is the bonafide work of "**NANDHA KUMAR P.R (21009104060), SHYAM SUNDAR E.H (21009104104), VERGHESE KOSHY PUTHUKKERIL (21009104123)",** who carried out the project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| **Dr. S. Murugavalli, M.E., Ph.D.,** | **Mrs. S. Malathi, M.E., (Ph.D.),** |
| **HEAD OF THE DEPARTMENT** | **SUPERVISOR** |
| DEPT. OF COMPUTER SCIENCE AND ENGINEERING | DEPT. OF COMPUTER SCIENCE AND ENGINEERING |
| PANIMALAR ENGINEERING COLLEGE | PANIMALAR ENGINEERING COLLEGE |
| POONAMALLEE, CHENNAI -600123 | POONAMALLEE, CHENNAI -600123 |

Certified that the candidate was examined in the Anna University Project Viva Voce held on ______________________.

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

# ACKNOWLEDGEMENT

A project of this magnitude and nature requires the kind co-operation and support from many for its successful completion. We wish to express our sincere thanks to all those who were involved in the completion of this project.

We would like to thank our esteemed and Honourable **Founder and Chairman**, **Thiru. Dr. JEPPIAR, M.A., B.L., Ph.D.,** for his sincere endeavour in educating us in his premier institution.

We would like to express our deepest gratitude to our beloved **Secretary and Correspondent, Thiru. Dr. P. CHINNADURAI, M.A., M.Phil., Ph.D.,** for his enthusiastic motivation which inspired us a lot in completing this project. Our sincere thanks also to our dynamic **Directors, Mrs. C. VIJAYA RAJESWARI** and **Mr. C. SAKTHI KUMAR, M.E., M.Phil.,** for providing us with the necessary facilities for the completion of this project.

We wish to express gratitude to our **Principal, Dr. K. MANI, M.E., Ph.D.,** for his encouragement and guidance. We would like to convey our sincere thanks to our **Head of the Department, Dr. S. MURUGAVALLI, M.E., Ph.D.,** and our **Project Supervisor** and **Mentor Mrs. S. MALATHI, M.E., (Ph.D.),** for their encouragement and valuable suggestions without which we would not have completed the project successfully.

We would also like to thank our **Parents** and **Family** for all the love, support and encouragement. Without them this project would not have event taken off.

# ABSTRACT

The requirement of a simple, yet sophisticated user interface has driven IT professionals to develop enhanced devices ranging from the basic punch-card system which were used for the earliest computer devices, to the widely used keyboard, mouse and the latest touch screen technology which is emerging on today's computers. These improved gadgets and technology basically reduce the invisible barrier between the everyday user and the system. However, these devices have their limitations and are inconvenient for the user to interact with the system. To simplify this procedure of interaction, a new system is developed which takes human gestures into consideration to perform the corresponding operation on the computer device.  New software is developed to capture the hand gesture of the user through web camera and interact with the system by using this primitive hand movement and gesture. The relative position of the hand is being used in the system to ensure that the gesture recognition is made accurately. The proposed hand gesture system simplifies the procedure and outperforms the mouse operation that is in vogue. The system also provides a low cost and easy to use environment for the user.

# TABLE OF CONTENTS

| CHAPTER | TITLE | PAGE NO. |
|---|---|---|
| | **ABSTRACT** | **i** |
| | **LIST OF FIGURES** | **v** |
| | **LIST OF TABLES** | **vii** |
| | **LIST OF ABBREVATIONS** | **viii** |
| **1** | **INTRODUCTION** | **1** |
| | 1.1SYNOPSIS | 1 |
| | 1.2 NEED FOR PROPOSED SYSTEM | 8 |
| | 1.2.1 CURRENT EXISTING SYSTEM | 8 |
| | 1.2.2 DESCRIPTION OF PROPOSED SYSTEM | 9 |
| | 1.2.3 BENEFITS OF PROPOSED SYSTEM | 10 |
| **2** | **LITERATURE SURVEY** | **12** |
| **3** | **SYSTEM CONFIGURATION** | **25** |
| | 3.1 HARDWARE SPECIFICATION | 25 |

| | | |
|---|---|---|
| | 3.2 SOFTWARE SPECIFICATION | 25 |
| **4** | **SYSTEM ANALYSIS AND DESIGN** | **26** |
| | 4.1 ARCHITECTURAL DESIGN | 27 |
| | 4.2 DATA FLOW DIAGRAM | 29 |
| | 4.3 SEQUENCE DIAGRAM | 33 |
| | 4.4 IMPLEMENTATION | 36 |
| | 4.4.1 PRIMARY COMPONENTS | 36 |
| | 4.4.2 IMAGE CAPTURE AND CORRECTION | 53 |
| | 4.4.3 IMAGE HAND DETECTION | 56 |
| | 4.4.4 EVENT TRIGGERING | 58 |
| **5** | **TESTING** | **60** |
| | 5.1 TYPES OF TESTING | 61 |
| | 5.1.1 BLACK BOX TESTING | 61 |
| | 5.1.2 WHITE BOX TESTING | 63 |
| | 5.2 OTHER TYPES OF TESTING | 64 |
| | 5.2.3 REGRESSION TESTING | 64 |
| | 5.3 TESTING IN PARTICULAR | 65 |
| | 5.3.1 UNIT TESTING - Test Case Design | 65 |
| **6** | **EXPERIMENT RESULTS AND DISCUSSION** | **73** |

| | | |
|---|---|---|
| | 6.1 BRIGHTNESS AND CONTRAST CONTROL | 73 |
| | 6.2 DETECTION OF OBJECTS | 74 |
| | 6.3 CAMSHIFT TRACKING | 76 |
| **7** | **SCREENSHOTS** | **79** |
| **8** | **CONCLUSION AND FUTURE ENHANCEMENT** | **85** |
| | 8.1 CONCLUSION | 85 |
| | 8.2 FUTURE ENHANCEMENT | 85 |
| | APPENDIX (CODING) | 86 |
| **9** | **BIBLIOGRAPHY** | **130** |

# LIST OF FIGURES

| FIGURE NO. | FIGURE NAME | PAGE NO. |
|---|---|---|
| 1.1 | American Sign Language | 7 |
| 4.1 | Hand Gesture System Overall Architecture | 27 |
| 4.2 | Hand Gesture System 0-Level Data Flow Diagram | 31 |
| 4.3 | Hand Gesture System 1-Level Data Flow Diagram | 32 |
| 4.4 | Hand Gesture System Sequence Diagram | 34 |
| 4.5 | Execution of threads for producer and consumer | 48 |
| 4.6 | HAAR Detection for hand detection | 52 |
| 4.7 | Flow chart for the Correction Algorithm | 55 |
| 4.8 | Flow chart for the Hand Detection Algorithm | 57 |
| 4.9 | Flow chart for Event Trigger Algorithm | 59 |
| 5.1 | Black Box Testing Diagram | 61 |
| 6.1 | Detection for hand based on Skin tone | 74 |
| 6.2 | Object Detection Techniques and Accuracy versus Time Complexity | 76 |
| 6.3 | CAMSHIFT Tracking based on input image | 78 |

| | | |
|---|---|---|
| 7.1 | Developer console | 79 |
| 7.2 | HSV Skin Detection | 80 |
| 7.3 | Edge Detection | 81 |
| 7.4 | Face Detection | 82 |
| 7.5 | Fist Detection | 83 |
| 7.6 | Fist Motion Detection | 83 |
| 7.7 | Palm Detection | 84 |

# LIST OF TABLES

| TABLE NO. | TABLE NAME | PAGE NO. |
|---|---|---|
| 5.1 | Testing of the image enhancement of the system | 68 |
| 5.2 | Testing of the Image Analysis & processing of the system | 70 |
| 5.3 | Testing of the Event triggering Module of the system | 72 |
| 6.1 | Comparison Results of Skin Detection with and without enhancement | 73 |
| 6.2 | Object Detection Techniques its Accuracy and Time Complexity | 75 |
| 6.3 | CAMSHIFT Tracking based on input images | 77 |

# LIST OF SYMBOLS AND ABBREVATIONS

| | |
|---|---|
| $\alpha$ | Alpha value / Contrasts Value |
| $\beta$ | Beta value / Brightness Value |

| | |
|---|---|
| API | Application Protocol Interface |
| ARTMAP | Adaptive Resonance Theory Mapping |
| CAMSHIFT | Continuously Adaptive Mean Shift Algorithm |
| DLL | Dynamic Link library |
| HCI | Human Computer Interaction |
| HMM | Hidden Markov Model |
| HRCNN | Hyper Rectangular Composite Neural Networks |
| TDNN | Time Delay Neural Network |

# 1. INTRODUCTION

## 1.1 SYNOPSIS

Since the genesis of computer, we have found better ways to interact with the system. We have broken many invisible barriers between man and machine. We have, in the process, moved from the most basic punch-card system, to hardwired devices like mouse, keyboard and, subsequently, to the current generation touch-screen technology. The system has become easier to interact with and use. In an attempt to render interaction with the computer even more easy we have introduced a system which takes human gestures into consideration and performs the corresponding operation. Computers are used by people either to aid them in their work or to entertain them in times of leisure. Special input and output devices have been designed over the years with the purpose of easing communication between computers and humans, the two most widely used of which are the keyboard and mouse.

Every new device can be seen as an endeavor to make the computer more effective, more intelligent and enabling humans able to perform more complicated communication with the computer. This has been possible due to the result oriented efforts made by computer professionals for creating successful human computer interfaces. The complexities of human needs have increased manifold and continues to grow and, hence, the need for complex programming ability and intuitiveness which are critical attributes required by computer programmers to facilitate their survival in a fiercely competitive environment.

Computer programmers have been incredibly successful in easing the communication between computers and human. With the emergence of every new product in the market; the ease with which we are able to complete progressively complex jobs has only increased. For instance, new products have helped in facilitating tele-operating, robotic use, better human control over complex work systems like cars, aeroplanes and monitoring systems. Earlier, computer programmers would avoid such complex programs as the focus was more on speed than on other modifiable features. However, a shift towards a user friendly environment has driven them to revisit the focus area.

The idea is to make computers understand human language and develop a user friendly human computer interfaces (HCI). Making a computer understand speech, facial expressions and human gestures are some steps towards it. Gestures are the non-verbally exchanged information. A person can perform innumerable gestures at a time. Since human gestures are perceived through vision, it is a subject of great interest for computer vision researchers. This project aims to determine human gestures by creating an HCI Coding of these gestures into machine language which, in turn, demands a complex programming algorithm. An overview of gesture recognition system is provided to gain basic insight.

## GESTURE RECOGNITION

It is hard to settle on a specific useful definition of gestures due to its wide variety of applications and a statement can only specify a particular domain of gestures. Many researchers had tried to define gestures but their actual meaning is still arbitrary. For a successful communication, a sender and a receiver must have the same set of information for a particular gesture. In accordance with the context of this project, gesture is defined as an expressive movement of body parts which has a particular message, to be communicated precisely between a sender and a receiver. A gesture is scientifically categorized into two distinctive categories: dynamic and static.

A dynamic gesture is intended to change over a period of time whereas a static gesture is observed at the spurt of time. A waving hand symbolizing ‘ goodbye ‘ is an example of dynamic gesture and the stop sign is an example of static gesture. To understand a full message, it is necessary to interpret all the static and dynamic gestures over a period of time. This complex process is called gesture recognition. Gesture recognition is the process of recognizing and interpreting a stream of continuous sequential gestures from the given set of input data.

A gesture recognition system could be used in any of the following areas:

- *Man-machine interface*: Using hand gestures to control the computer mouse and/or keyboard functions. An example of this, which has been implemented in this project, controls various keyboard and mouse functions using gestures alone.

- *Visualization*: Just as objects can be visually examined by rotating them with the hand, so it would be advantageous if virtual 3D objects (displayed on the computer screen) could be manipulated by rotating the hand in space [Bretzner & Lindeberg, 1998].

- *Computer games*: Using the hand to interact with computer games would be more natural for many applications. 'Kinect' is one example of this technique of gesture recognition.

- *Control of mechanical systems (such as robotics)*: Using the hand to remotely control a manipulator. This can be used in places where human strength may be inadequate and human dynamics in hand movement can be used.

## GESTURE BASED APPLICATIONS

Gesture based applications are broadly classified into two groups on the basis of their purpose: multi directional control and a symbolic language.

*3D Design:* CAD (Computer Aided Design) is an HCI which provides a platform for interpretation and manipulation of 3-Dimensional inputs which can be the gestures. Manipulating 3D inputs with a mouse is a time consuming task as it involves a complicated process of decomposing a six degree freedom task into at least three sequential two degree tasks. Massachusetts Institute of Technology, Boston (MIT) has come up with the 3DRAW technology that uses a pen embedded in a Polhemus device to track the pen position and orientation in 3D. A 3Space Sensor is embedded in a flat palette, representing the plane in which the objects rest. The CAD model is moved synchronously with the user's gesture movements and objects can thus be rotated and translated in order to view them from all sides as they are being created and altered.

*Tele presence:* There may arise a need for manual operations in some cases such as system failure, emergency hostile conditions or inaccessible remote areas. Often it is impossible for human operators to be physically present near the machines. Tele presence is that area of technical intelligence which aims to provide physical operation support that maps the operator's forearm to the robotic arm to carry out the necessary task. For instance the Real Time ROBOGEST system constructed at University of California, San Diego presents a natural way of controlling an outdoor autonomous vehicle by use of a language of hand gestures.  The prospective areas of tele-presence includes space, underwater missions, medicine manufacturing and in maintenance of nuclear power reactors.

*Virtual Reality:* Virtual reality is applied to computer-simulated environments that can simulate physical presence in places in the real world, as well as in imaginary worlds. Most current virtual reality environments are primarily visual experiences, displayed either on a computer screen or through special stereoscopic displays. There are also some simulations which include additional sensory information, such as sound through speakers or headphones. Some advanced, haptic systems now include tactile information, generally known as force feedback, in medical and gaming applications.

*Sign Language:* Sign languages are the most raw and natural form of languages and may be traced back to as early as the advent of human civilization, when the first reports of sign language appeared in history. It started even before the emergence of spoken languages. Since then sign language has evolved and has been adopted as an integral part of our day to day communication process. Now, sign language is being used extensively in international sign use of deaf and dumb, in the world of sports, for religious practices and also at work places. Gestures are one of the first forms of communication when a child learns to express its need for food, warmth and comfort. It enhances the emphasis of spoken language and helps in expressing thoughts and feelings effectively.

A simple gesture with one hand has the same meaning all over the world and means either ’hi’ or ‘goodbye’. Many people travel to foreign countries without knowing the official language of the visited country and still manage to communicate by using gestures and sign language. These examples show that gestures can be considered international and used almost all over the world. In a number of jobs around the world gestures are means of communication. In airports, a predefined set of gestures makes people on the ground able to communicate with the pilots and thereby give directions to the pilots of how to get off and on the run-way and the referee in almost any sport uses gestures to communicate his decisions. In the world of sports gestures are common. The pitcher in baseball receives a series of gestures from the coach to help him in deciding the type of throw he is about to give. Hearing impaired people have over the years developed a gestural language where all defined gestures have an assigned meaning. The language allows them to communicate with each other and the world they live in.

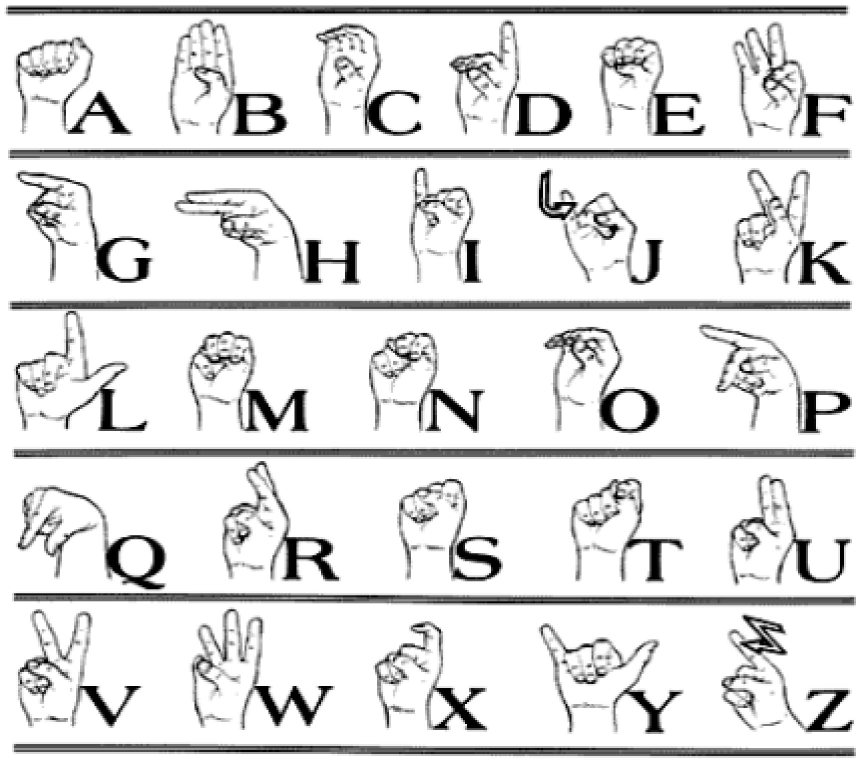


**Figure 1.1 American Sign Language**

The ‘Kinect’ which was introduced by Microsoft is the first example of gesture recognition technology. The system is basically designed for entertainment, to be more specific ‘for gaming based application’. The ‘Kinect’ system provides additional hardware to enable better image detection, image coordination and image control which in-turn provide the fundamentals for gesture recognizes process. The proposed system uses a basic web camera to get the gesture inputs from the user and then process them for various gestures. This project aims to design and build a man-machine interface using a video camera to interpret the gestures like basic click operation.

In other areas where 3D information is required, such as computer games, robotics and design, other mechanical devices such as roller-balls, joysticks and data-gloves are used. Humans communicate mainly by vision and sound and, therefore, a man-machine interface would be more intuitive if it made greater use of vision and audio recognition. Another advantage is that the user not only can communicate from a distance and does not require any physical contact with the computer. However, unlike audio commands, a visual system would be preferable in noisy environments or in situations where sound would cause a disturbance.

The visual system chosen was the recognition of hand gestures. The amount of computation required to process hand gestures is much greater than that of the mechanical devices. However, standard desktop computers are available now are quick enough to make this project — hand gesture recognition using computer vision — a viable proposition.

## 1.2 NEED FOR PROPOSED SYSTEM

### 1.2.1 CURRENT EXISTING SYSTEM

The current system consists of hardware devices like a simple keyboard or a simple mouse, which requires additional space to dock the various components. Even if the current system proves useful in certain scenarios, a more creative system which allows useful gesture inputs from the user. The touchscreen system provides much more creative input gestures but with the additional cost, it may not be feasible for more users.

Hence we have described a new system which basically uses only the web-camera.

## 1.2.2 DESCRIPTION OF THE PROPOSED SYSTEM

The system which we have proposed consists of a simple hand gesture system. The software obtains the inputs from the camera to the image enhancement subsystem component, which improves the quality of the image that is captured from the web-camera. The system must, subsequently, detect the hand from all other objects in the user's environment where he/she uses the given computer system. The system uses the HAAR based algorithm to get the metadata which is used to differentiate the human hand from all other objects within the environment. Once the system identifies the hand in the environment, the tracker must use an algorithm which is less time consuming than HAAR based detection which is a very CPU intensive process. Hence, the requirement is to find a less complex and less CPU intensive process to keep track of the detected hand. So, we use CAMSHIFT to track the hand after the system detects the hand via HAAR detection. The CAMSHIFT uses an image which is a combination of the image detected via HSV skin detection and edge detection using canny edge detection. This improves the tracking rate of the CAMSHIFT algorithm.

Once we manage to track the hand from the environment the system must use this information to find the relative displacement of the hand from the original position of the hand in the environment. Using this displacement value we move the mouse pointer on the system. To perform the click operation and other mouse operations we must use the convex-hull algorithm on the hand. The convex-hull detection algorithm and the convex-hull defects detection algorithm together are used to calculate the various ratios in the hand which can be used by the system to determine the operation which it must perform. For the current system, we will implement only a simple click operation which is triggered by the display of the user's palm.

### 1.2.3 BENEFITS OF PROPOSED SYSTEM

The Advantages of our proposed system are the following:

- All the system needs is a good camera and a good processor to perform image processing for the inputs and no additional space is required for the system for the mouse track pad.

- The system does not need the user to use cumbersome devices like hand gloves or any sensors connected to a system.

- The system can be used more securely than mouse and keyboards since there is no actual use of the system buffer of the machine like in the case of keyboards, hence reducing the chance of eaves-dropping the device.

- An advanced version of the same system can be used for complex gestures like different types of clicks, scrolling, swiping, pinch zoom etc.

- This system is comparatively cheaper than other image based systems since the cost of development of the system is low, and also since there is no other device or additional sensors which are required for the functioning.

- Since the current application uses relative position of the hand, the motion of the hand can be tracked better and movement of the mouse is much smoother and faster.

- The image correction incorporated reduces the noise in the environment and performs minor image corrections to improve the overall quality and rate of detection.

- The system has HAAR like feature detection which is widely used in face detection techniques. We have also added HAAR based palm and fist detection by using the same principle as HAAR face detection to improve accuracy.

# 2. LITERATURE SURVEY

Ever since the computer has come into being, we have found better ways to interact with systems, thereby reducing the invisible barriers between man and machine. We have progressed from the most basic punch-card system to hardwired devices like mouse, keyboard and, subsequently, to the current generation touch-screen technology. The system has become easier to interact with and use. Hence, we have introduced a system which takes human gestures into consideration and performs the corresponding operation. The 'Kinect' which was introduced by Microsoft is the first example of such technology. The system is basically designed for entertainment and to be more specific, 'for gaming based application'. The system provides additional hardware to support better image detection, image coordination and image control which in turn provides the fundamentals for gesture recognizes process.

One major issue with the 'Kinect' system is that it is expensive and may not prove to be useful and cheap for everyday computing. To circumvent this problem we must create new software which uses the basic camera (say web-camera) to identify the hand gesture of the user. In our system, we have endeavored to implement a simple hand gesture system to perform basic and simple mouse operations on the currently used system.

Frederick C. Harris et al., propose a Vision-based Human Computer Interaction [5] hand-gesture analysis which includes a variety of gestures as input and a virtual hand as output to display the results. In Human Computer Interaction (HCI) applications, traditional Devices, such as keyboard and mouse, can end up cumbersome and unsuitable for use. Although the hand is a complex biological organ, we can also think of it as a mechanical machine and apply mechanical principles to study it. The various movements of the hand's different parts are analyzed and categorized as biomechanical motions. This provided us with the guidance for a computer model. We can look at hand motions as complex combinations of the movements (rotations around different axes) of different bones at various joints. Our current hand model cannot represent different hand sizes (for example, thin or thick) and, hence, we plan to calibrate it to different representative hand shapes.

The problem of different hand sizes in Vision-based Human Computer Interaction is overcome by Continuously Adaptive Mean Shift algorithm. CAMSHIFT [11] is a simple, computationally efficient face and colored object tracker designed to be a highly efficient face tracking algorithm rather than a more complex higher MIPs usage algorithm. CAMSHIFT is then used as a computer interface for controlling commercial computer games and for exploring immersive 3D graphic worlds. The mean shift algorithm operates on probability distributions. Tracker should be able to serve as part of a user interface that is in turn part of the computational tasks that a computer might routinely be expected to carry out. This tracker also needs to run on inexpensive consumer cameras and should not require calibrated lenses. CAMSHIFT is usable as a visual interface now, yet designed to be part of a more robust, larger tracking system in the future. In order, therefore, to find a fast, simple algorithm for basic tracking, we have focused on color-based tracking.

An algorithm based on skin color hand segmentation and tracking for gesture recognition from extracted hand morphological features [7] is employed to overcome the defects described in CAMSHIFT. This algorithm is based on three main steps: hand segmentation, hand tracking and gesture recognition from hand features. For the hand segmentation step we use the color cue due to the characteristic color values of human skin, its invariant properties and its computational simplicity. To prevent errors from hand segmentation we add the hand tracking as a second step. Tracking is performed assuming a constant velocity model and using a pixel labeling approach. From the tracking process we extract several hand features that are fed into a finite state classifier which identifies the hand configuration. The hand can be classified into one of four gesture classes or one of four different movement directions. Finally, the system's performance is evaluated by showing the usability of the algorithm in a videogame environment. The main problem addressed here is the overcoming of the restriction of real-time response and the use of unconstrained environments.

The accurate classification of hand gestures is crucial in the development of novel hand gesture based systems and the problem of morphological features is eliminated by using Morphological filtering, Gesture segmentation and Contour representation [17]. There are two factors that need to be considered in the development of gesture based classification systems for HCI and HAAC applications. The first is to demonstrate that the gestures can be classified accurately by the system. The next is to demonstrate that the gestures can be classified in real-time. It is estimated that if the system is developed using a high-speed PC with high-speed digital signal processing chips, images are read directly from a buffer, and the code for the processing steps is optimized, gestures can be classified fast enough to make real-time applications possible. A complete vision-based system consisting of hand gesture acquisition, segmentation, filtering, representation, and classification is developed to robustly classify hand gestures.

Additionally, through the further reduction in classification time, the VHGC could be extended to classify dynamic gestures represented by sequences of static gestures.

The real time implementation of skin color hand segmentation is done using a Kalman filter and hand blobs [2] analysis to obtain motion descriptors and hand regions. It is fairly robust to background cluster and uses skin color for hand gesture tracking and recognition. The system consists of three modules: real time hand tracking, training gesture and gesture recognition using pseudo two dimension hidden Markov. The basic idea lies in the real-time generation of gesture model for hand gesture recognition in the content analysis of video sequence from camera. Since hand images are two-dimensional, it is natural to believe that the 2-DHMM, an extension to the standard HMM, will be helpful and would offer great potential for analyzing and recognizing gesture patterns. The system is fully automatic and it works in real-time. The advantage of the system lies in the ease of its use. The users do not need to wear a glove and neither is there need for a uniform background.

William T. Freeman devised a method to recognize hand gestures, based on a pattern recognition technique developed by McConnell employing histograms of local orientation [15]. We use the orientation histogram as a feature vector for gesture classification and interpolation. For static hand gestures, we use the histogram of local orientations as a feature vector for recognition. The methods are image based, simple, and fast. The trackball, the joystick, and the mouse are extremely successful devices for hand-based computer input. Yet all require that the user hold some hardware, which can be awkward. Devices such as the Data glove can be worn which sense hand and other positions. We seek a visually based method which will be free of gloves and wires.

Orientation Histogram algorithm describes a vision-based approach to hand gesture recognition which deals with real-time tracking of the hand with simultaneous recognition of the gesture being made. Accurate tracking of the hand is done using HAAR like features [16] while gesture recognition is done by computing orientation histograms of video frames. The system is trained using a well-defined vocabulary of hand-gestures, and the algorithm generates the current position of the hand and the gesture being made. The AdaBoost algorithm is used to boost the process of HAAR training. The performance of the algorithm is at its best when the hand is positioned at the centre of the camera viewing space. There is a gradual decrease in performance when the hand is moved towards the boundaries. In order to achieve robustness to varying conditions of operation such as illumination, posture, zooming conditions, skin color and size of the hand, it is crucial to incorporate all possible variations during the training phase of the system.

The Orientation histogram method applied using HAAR like features have some problems like similar gestures having different orientation histograms while different gestures could have similar orientation histograms. Hence, a new method is proposed to achieve good results for any objects that dominate the image even if it is not the hand gesture. Neural Network classifier has been applied for gestures. The methodology includes three steps namely Extraction Method and image pre-processing, Features Extraction [14] and Gestures Classification. For dynamic gestures HMM tools are perfect and have demonstrated their efficiency especially for robot control NNs.

Trajectory recording [20] and Trajectory segmentation is based on strokes in order to allow the use of a low complexity gesture recognition method. The gesture recognition process is trivial, being reduced to a syntactic analysis of the feature vector avoiding the necessity of complex classification methods based on curve matching. Kernel based probability density estimation is used in estimating the probability density gradient. Despite the restrictions imposed on account of the stroke based definition of gestures, the low computational complexity of the algorithm allows its implementation on low-cost processing systems. Some further developments are also possible in the tracking algorithm. The use of a more generally lighting invariant color function and the use of some semantic information about the tracked object (human hand) can lead to the development of an automatic tracker initialization method and can also improve the tracking results under severe conditions like important lighting variance.

A fuzzy rule-based approach to spatio-temporal hand gesture recognition employs a powerful method based on Hyper Rectangular Composite Neural Networks (HRCNNs) [24] for selecting templates. Templates for each hand shape are represented in the form of crisp IF-THEN rules that are extracted from the values of synaptic weights of the corresponding trained HRCNNs. Based on the method we were able to implement a small-sized dynamic hand gesture recognition system. By training a HRCNN, we efficiently generated templates for each basic hand shape. Then an unknown sign word was classified to the corresponding sign word in the vocabulary by computing cumulative similarities. In this manner, we obviated substantial computation for time alignment.

A new approach to gesture recognition based on incorporating the idea of fuzzy ARTMAP [3] in the feature recognition neural network is introduced. Firstly, gray level for each pixel in a monochrome image will be used to determine membership function value. Hence, the input layer of neural network contains fuzzy entries corresponding to each pixel. Secondly, we divide an initial physical image G to regions for obtaining a logical image representation P. It is very important that the size of logical images is invariant for various physical images.  It is clear that neural network technique is more effective for logical images than for physical images. Thirdly, an initial physical image is also divided into regions. For each region some characteristic features will be extracted using neural network. Training patterns are allowed to be merged based on the measure of similarity among features. A subnet is allowed to be shared by similar patterns. A minimal number of subnets is learned automatically to meet accuracy criteria. Therefore, the network size can be reduced and training patterns better.  A big network is obtained when the number training patterns is large. The use of vigilance parameters and matching degrees permits the combination of similar training patterns automatically in the same subnet. The complexity of the operations is reduced, due to the patterns have been permitted to be merged based on the measure of similarity among features.

A novel method for a hand-pose estimation that can be used for vision-based human interfaces uses two models namely voxel model [25] and hand model. The aim of this method is to estimate all joint angles. The hand regions are extracted from multiple images obtained by a multi viewpoint camera system. Estimation of joint angles by the silhouette images and hand pose estimation are performed. At present, there are two major problems in our system. One is an accuracy problem, which is caused by modeling errors in the hand model. A technique to model hands more accurately is required. The other problem is processing speed. To build natural and practical human interfaces, the system should run at least in 10 Hz, which is three times faster than our current system.

A vision-based hand tracking system [9] is employed that does not require any special markers or gloves and can operate in real-time on a commodity PC with low-cost cameras. Stereo vision hand tracking system is used for interaction purposes. The system uses two low-cost web cameras mounted above the work area and facing downward that can track the 3D position and 2D orientation of the hand without the use of special markers or gloves. The problem is still far from being solved since the hand exhibits significant amounts of articulation and self-occlusion that cause difficulties with existing algorithms.

3-D visual modelling and real-time tracking [18] is used for presenting a non intrusive system based computer vision for human-computer interaction in three–dimensional (3-D) environments controlled by hand pointing gestures. Once captured and tracked in real-time using stereo vision, hand pointing gestures are remapped onto the current point of interest, thus reproducing them in an advanced interaction scenario of the “Drag and Click” behavior of traditional mice. The system is given a careful modeling of both user and optical subsystem, and visual algorithms for self-calibration and adaptation to both user peculiarities and environmental changes. The system works in real time on a low-cost hardware platform, is fairly accurate and independent of user characteristics and position, camera layout, and environmental changes

The problem of vision-based hand tracking system is rectified by introducing a 3D reconstruction technique [10] which is capable to determine the

3D model of a scene without any external (human) intervention. 3D model reconstruction uses as input only digital images taken from different camera positions. The technique combines recent results of epipolar geometry, intelligent methods of image processing and different fuzzy techniques. One of the most important difficulties in autonomous 3D reconstruction is the (automatic) selection of the ‘significant’ points which carry information about the shape of the 3D bodies.

The selection of significant points can be solved using fuzzy based noise elimination and feature detection algorithms [12]. Here a new fuzzy based method for the matching of corresponding feature points in images is introduced. It consists of mainly three parts namely pre-processing of the Images, Noise Elimination and detection of the Feature Points and matching the Corresponding Feature Points in Stereo Images. The most significant problem in stereo vision is how to find the corresponding points in two different fields. In the field of computer vision several applications require to match feature points of images taken from different camera positions.

A vision interface immersive projection system, CAVE [8], is introduced in virtual reality research field so that hand gesture recognition with computer vision techniques is made possible. Photographs of real environment can be applied to this device to make more immersive contents. The idea of our technique came from real image-based rendering in computer graphics. We add an affected interface to this system. A Stand-alone system like wireless network connected notebook, tablet, PC is preferred for interface processing because of the seamless feature of CAVE. A background image was subtracted from current image frame of webcam and we converted the color space of the image into HSV space. We presented the framework for hand gesture recognition and it can be applied for other environments, too. It uses built-in webcam of notebook PC and, hence, does not require expensive devices.

The problem with CAVE is that as a result of extracting hands based on color image segmentation or background subtraction it often fails when the scene has a complicated background and dynamic lighting. Hence, fingertip tracking [6] is being employed to recognize a particular gesture. Once we locate a user's arm regions including hands, in an input image, we search for fingertips within those regions. It improves our tracking method's reliability by incorporating additional sensors. Although using an infrared camera has some advantages, it did not work well on cold hands. The problem is solved by using a color camera in addition to the infrared camera.

Fingertip based detection [19] combined with fuzzy logic proves to be an intelligent approach in the area of hand gesture recognition system. The approaches include soft computing based methods like artificial neural network, fuzzy logic, genetic algorithms etc. Soft computing provides a way to define things which are not certain, but with an approximation. This can be made certain by using learning models and training data and, hence, it is effective in obtaining results where the exact positions of hands/fingers are not available. The challenge is to work in the area of individual finger position bending detection and movements, as work done in this area is limited.

A technique is introduced which overcomes background complexity and distance from camera which is up to 2.5meters. A new technique is proposed which begins by detecting the hand and determining its canter [13], tracking the hands trajectory and analyzing the variations in the hand locations, and finally recognizing the gesture. The main problem addressed here is that Glove-based gesture interfaces require the user to wear a cumbersome device, and generally carry a load of cables that connect the device to a computer. Hence, this method makes sure that user does not need to wear any gloves to interact with the system.

The need for perceptual information about world state which further differentiates intelligent environments from traditional computing is studied. TheEasyLiving [4] project is building architecture for intelligent environment. TheEasyLiving project is concerned with development of architecture and technologies for intelligent environments which allow the dynamic aggregation of diverse I/O devices into a single coherent user experience. The design and implementation of this architecture is an ongoing effort. While some progress has been made, there are still a number of major issues to address.

The problem which arises through TheEasyLiving project is overcome by Intelligent sensors [1]. The sensors are distributed in a space and they provide functions based on position information. The proposed architecture satisfies not only scalability but also re-configurability, modularity, easy maintenance and affinity problems. Intelligent space is still at an early stage of development and is far from being a practical system. For it to be practical and useful, many functions and agents remain to be developed.

We address the issue of re-configurability [1] of a hand-gesture recognition system. The calibration or setup of the operational parameters of such a system is a time-consuming effort, usually performed by trial and error, and often causing system performance to suffer because of designer impatience. In this work, we suggest a methodology using a Neighborhood-search Algorithm [23] for tuning system parameters. Thus the design of hand-gesture recognition system is transformed into an optimization problem. Simultaneous calibration of the parameters of the image and supervision of FCM algorithm are used. Our near-optimal parameter-search procedure is easily extended to systems with larger parameters and more complex hand-gesture recognition systems. Ongoing research is aimed to improve the NS algorithm.

A standard face detection algorithm [21] to recognize gestures is used. While this use of face detection assumes a frontal face, i.e. the user is facing the camera; one could also develop a system using tracking in addition to face detection. This could allow the recognition of gestures with larger torso movements. Given an input video, we derive a set of motion features based on smoothed optical flow estimates. A user-centric representation of these features is obtained using face detection and an efficient classifier is learned to discriminate between gestures. A real-time system using GPU programming for implementing the classifier is developed. One could also use the GPU for the face detection and optical flow computation, the other major processing needs of our system.

An algorithm is used for extracting and classifying two-dimensional motion in an image sequence based on motion trajectories using time-delay neural network [22]. Pixels matches over consecutive image pairs are concatenated to obtain pixel-level motion trajectories across the image sequence. Motion segmentation is performed to generate regions with uniform motion, and moving regions are then extracted using color and geometric cues. Motion patterns are learned from the extracted trajectories using a time-delay neural network. We have emphasized a generic method to extract motion trajectories of hand gestures and to utilize TDNN for recognition. Given that the method is able to extract gestural motion trajectories, we believe that hand gestures can also be recognized by other methods based on Hidden Markov Model (HMM) or principal curves.

# 3. SYSTEM CONFIGURATION

## 3.1 HARDWARE CONFIGURATION

**Processor**: Any Intel or AMD Processor
(We recommend using an Intel Core-series or Intel Core I-series, since the OpenCV libraries are optimized using Intel Threading Building blocks libraries and, further, it gives better parallel processing which is required).

**Hard disk:** Space of 50MB or more (The development software and resources will require around 6-7 GB, but for a non-developer machine, addition resources will not prove useful).

**RAM capacity:** 1 GB or more (Generally the system will remove all unnecessary resource during normal execution and, hence, 1GB will do).
Web Camera
(Use a camera with good quality color detection sensors and image resolution to improve accuracy)

## 3.2 SOFTWARE CONFIGURATION

**Framework**: Visual C++ 2010, OpenCV2.4, Intel Threading Building Blocks, Boost Threading Libraries and CMake.

**Platform**: Windows 7 and above.

**Additional Software:** Device driver for web camera in use.

# 4. SYSTEM ANALYSIS AND DESIGN

System design is basically a bridge between analysis and implementation phases. It illustrates how to achieve the solution domain from the problem domain. The main objective of the design is to transform the high level analysis concepts used to describe problem domain into an implementation form. System design is the evaluation of alternative solutions and specification of a detailed computer based solution.

**UML** (Unified Modeling Language) is a language for specifying, visualizing and constructing the artifacts of software system as well for business needs. Grady Booch, Ivar Jaccobson and James Rambaugh introduced it. The UML notation is useful for graphically depicting object oriented analysis and object oriented design (OOA and OOD) modules.

**Use Case diagram:** Describes the functionality provided by a system in terms of goals represented as use cases actors and any dependencies among those use cases.

**Sequence Diagram:** Shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespan of objects relative to those messages.

**UML State Chart Diagram:** Describes the states and state transitions of the system.

## 4.1 ARCHITECTURAL DESIGN

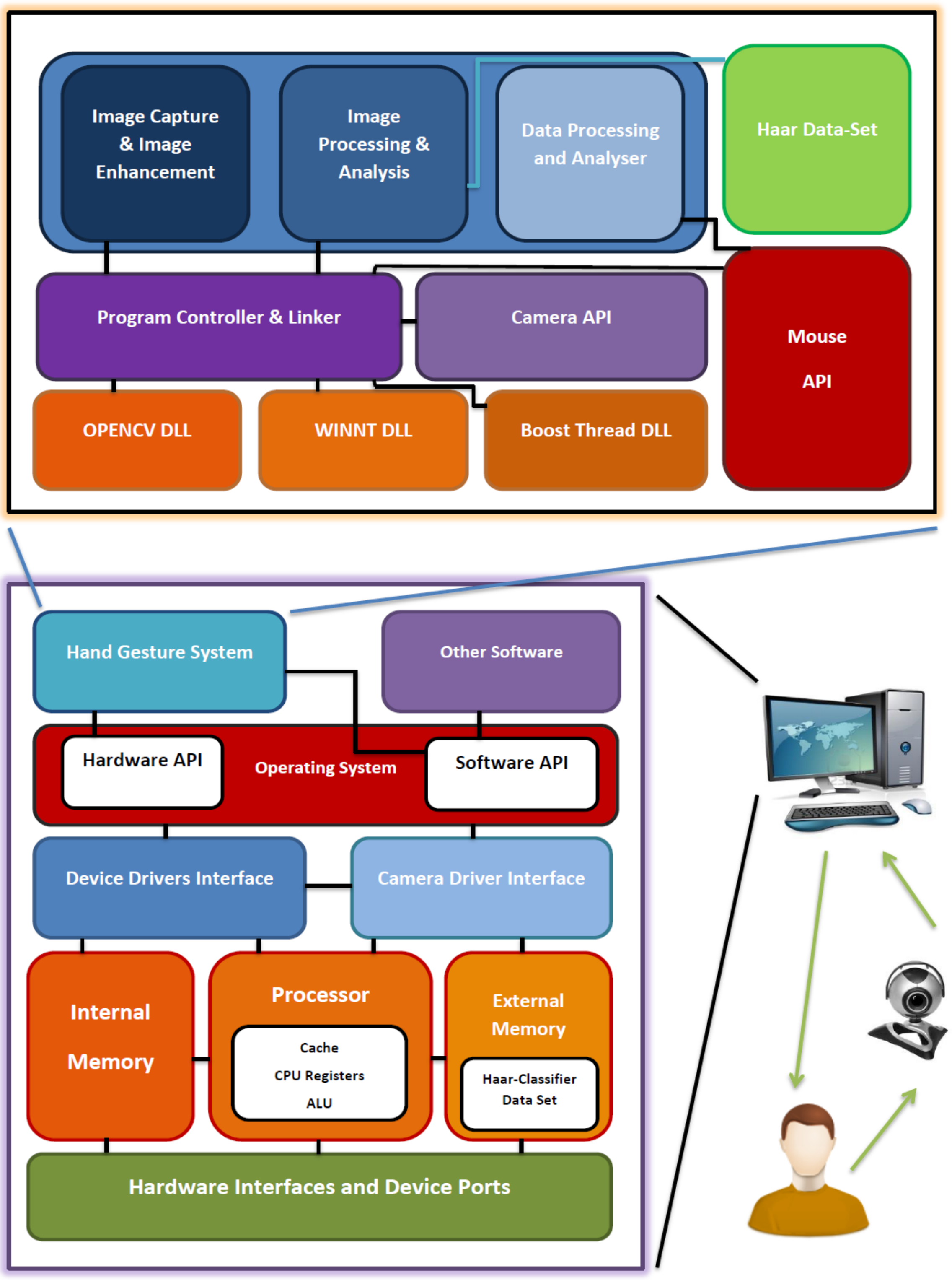


Figure 4.1 Hand Gesture System Overall Architecture

While the system architecture gives the complete overview of how each and every component of the system functions with both the environmental variables (e.g.: Human hand), the diagram gives a rough overview on how each and every component of the system functions with one another.

The system will need the core hardware component of a webcam or any digital camera which takes the image of the environment in order to get the image feed which will be required by the program to ensure the proper connectivity between the webcam and the system. The Operating System will require the primitive low level instruction set which will be provided by the webcam device drivers. The Operating System using the device driver will provide the captured image frame, the program then fetches the frames for next stage of processing.

The system converts the RGB image to gray scale HSV. Using the two sets of images the system then tries to isolate the human being from the environment. The image processing component also removes small scaled noise from the image, increases the brightness and contrast of the image to improve the overall detection rate. The modified frame is now stored on the main memory for more processing.

Once the basic processed images frames are available, we can use skin detection technique to remove wide amount of noise in the environment as well as some bit of the background, hence, to a certain extent isolate the human being from the environment. The isolation is also aided using HAAR like detection.

The application now using the isolated hand, the system calculates the ratio between the various fingers from the fixed point. By using the ratio values we determine the change and thereby determine the operations which have to be performed by the system.

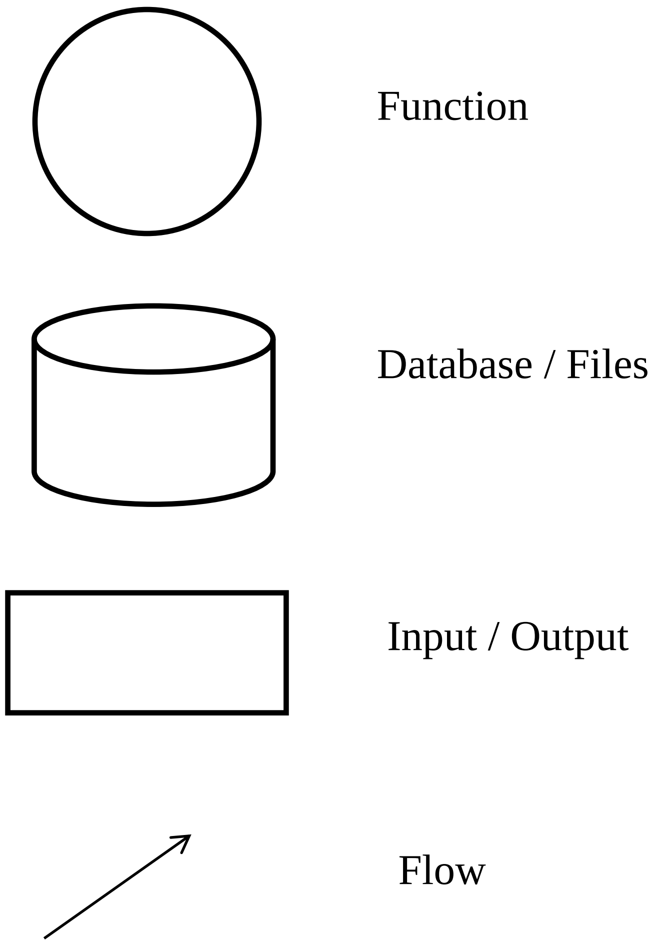
The application then uses the knowledge of the operation and determines the system Application interface to trigger the event onto the corresponding Device. It loads the event onto the device buffer; the Operating System in turn reads the buffer and triggers the device event as if it was the device itself.

The system uses the open Computer Vision (openCV) libraries for the functioning of the various system components; the library provides real time camera support as well as predefined image processing support. Additionally, the libraries can be applied to both the x86 and x64 architecture. Also, since the libraries can be imported to C++, we are provided with fast and optimal program without the bottlenecks due to post compilation of the program. The downside is that since the programming is done in C++, the coding must be well coded with little or no room for error, since there is no just-in-time debugger and the playing field for the program is open to the system memory space.

For the Runtime environment the application will use the precompiled dynamic link libraries, the already available services of the operating system and its subcomponents.

## 4.2 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).



### 4.2.1 LEVEL-0 DFD

The level-0 DFD gives the idea of the general outline of the various primary components which will be required by the system for its development. This diagram gives the team members a general idea on what is need by the team to develop the project. The level-0 DFD is a primitive layout of the system.

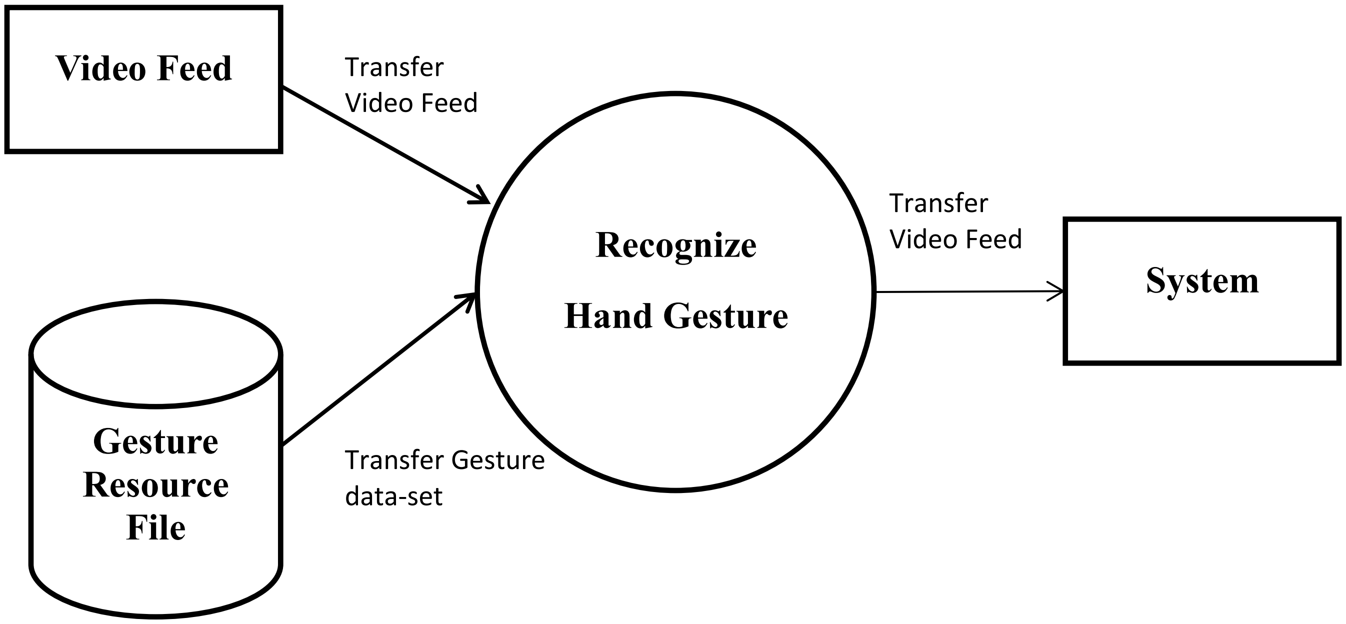


**Figure 4.2 Hand Gesture System 0-Level Data Flow Diagram**

The 0-Level Data flow diagram of hand gesture user interface system shows the basic overview of the data flow in the hand gesture system. The Video Feed (webcam) provides the main source of input for the image processing. The system transfers the whole video feed into the system where it obtains all necessary details and necessary image processing operations are performed. The system then gets gesture details which are used by the system to trigger the events on the system.

### 4.2.2 LEVEL-1 DFD

The 1-Level DFD for the hand gesture system consists of 2 major components which correspond to its functioning. The first is the producer which gets the information from the camera and captures the region of interest. The second is the consumer which provides the analysis of the region of interest and gets the input

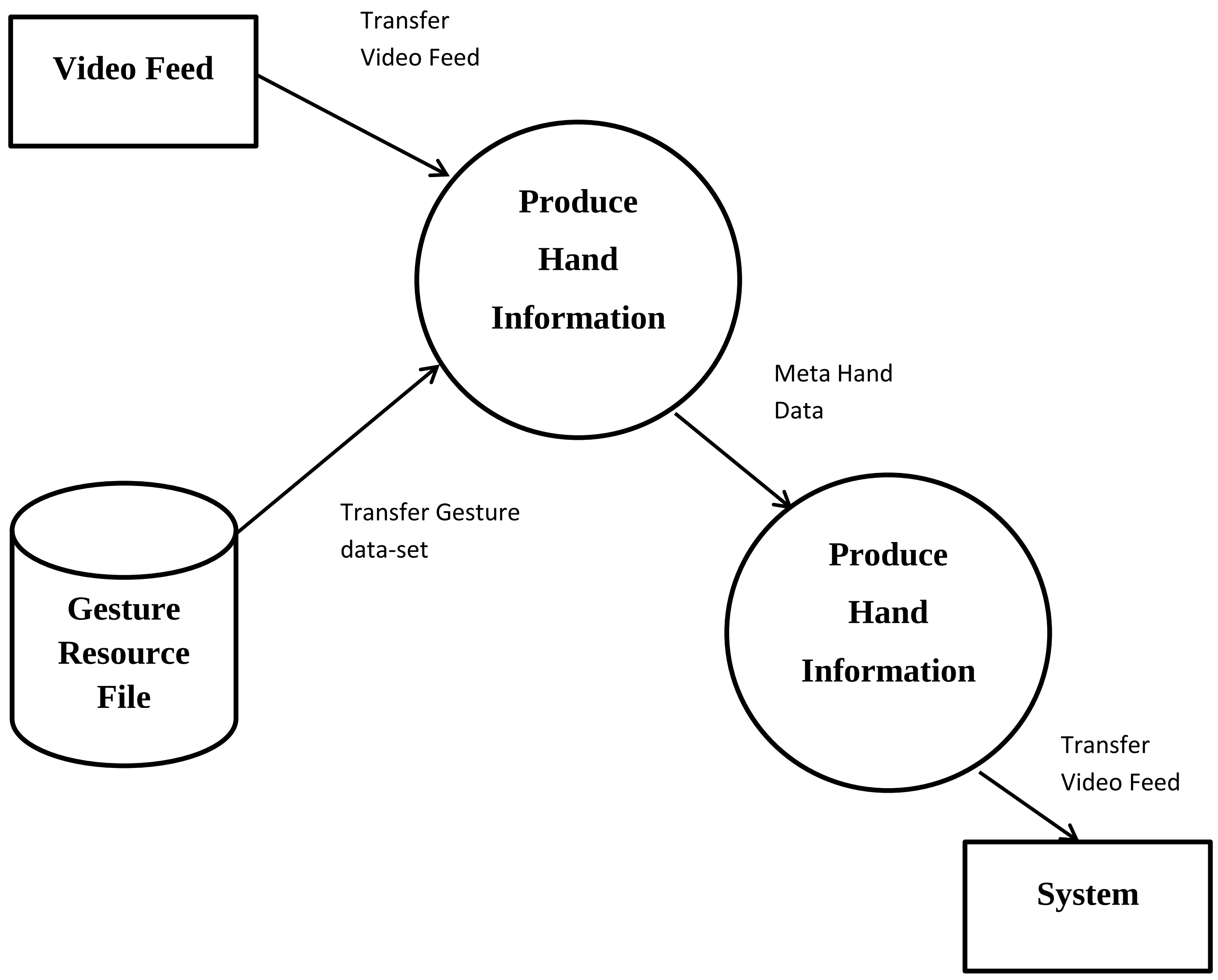


**Figure 4.3 Hand Gesture System 1-Level Data Flow Diagram**

The 1-Level Data Flow diagram of the hand gesture user interface system shows the sublevel or subcomponents which are present in the inner working Data flow models. The video feed is obtained through a web camera and it is given as one of the inputs to the system. The other input, gesture resource file which contains the possible ratios for a particular hand gesture, is facilitated on the system so as to accommodate future changes and enhancements for the system. The file also contains the dataset for the hand detection module using HAAR like detection method. The HAAR like detection is used to improve the overall hand detection rate and enhance accuracy.

The produce Hand information process uses contour detection and contour defect detection to get the approximate points which will be used for the calculation of the various ratios. The system uses ratios instead of fixed values to make the system more robust and improve gesture analysis and detection. Another reason is that we cannot easily determine the distance of the user from the camera and, hence, determining the gesture becomes difficult with fixed values.

Once the ratios are calculated the system must determine whether the ratio calculated by the producer is correct and a valid gesture should be analyzed by the detection and analysis system. Once the classification of the gesture is done, the system triggers the device event on the system. For our application we will trigger simple mouse operation on the system. The system sends the mouse event data to the mouse device driver buffer which in turn is read by the Operating System as a default mouse event.

## 4.3 SEQUENCE DIAGRAM

A sequence diagram in the context of **UML** represents object collaboration and is used to define events sequences between objects for a certain outcome. A sequence diagram is an essential component used in process related to analysis, design and documentation.

The sequence diagram gives the overview of the flow of control of the various sequence of execution of the various classes of the given program, the class may include various hardware control, the actor and the various system components both software and hardware.

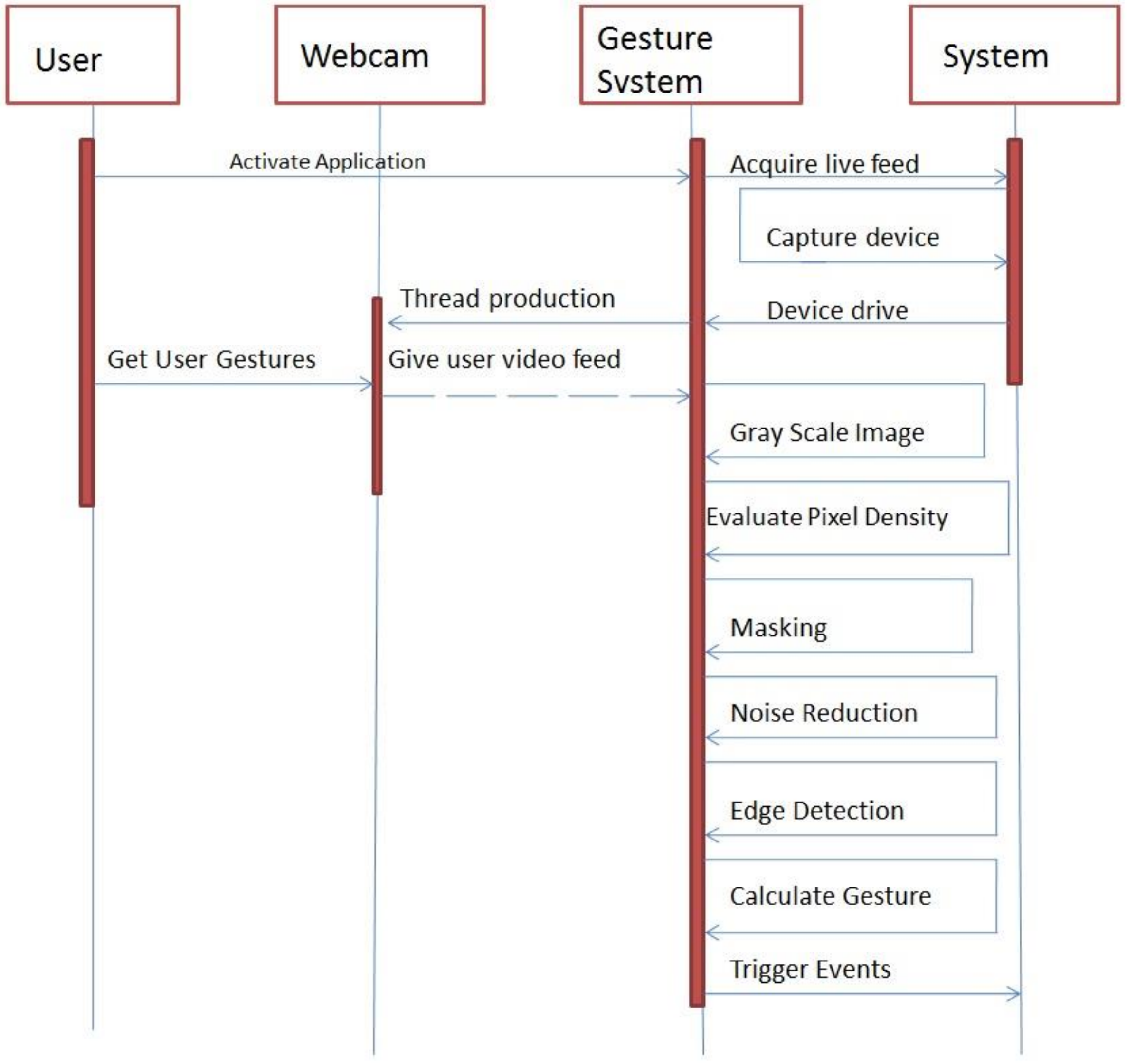


**Figure 4.4 Hand Gesture System Sequence Diagram**

The Sequence Diagram in Figure 4.4 denotes the flow of control from the various modules, objects and classes within the system. The user first activates the main application which in turn activates a thread to produce the video feed from the webcam. Once the thread is generated by the main application, the thread continues to produce the video feed to the application. This is similar to the working of producer and consumer.

The gesture system captures the device and transfer control to the producer thread which produces the frames which are required by the system. The image received from the system is checked first for pixel density which is used by the system to generate the change in contrast and brightness. This helps the system to improve edge detection, contour detection and skin detection which will be used to produce the final results.

Using skin detection and HAAR like detection the image is masked to obtain an image which is easier to process and analyze. The noise reduction is done on the masked image to remove the various noises which are generated due to dynamic nature of the background, inaccurate nature of the camera and inaccuracies present in the algorithm. This will also remove inaccuracies which are inherited. Edge detection is done so as to get a better contour map of the hand.

The final image is analyzed by the system to determine the gesture using contours hull and the contour defects. Finally the event is triggered on the system.

## 4.4 IMPLEMENTATION

Implementation is the stage of the project when the logical design is turned out into a prototype. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user confidence that the new system will work and be effective. The implementation stage involves careful planning, investigation of the existing system and its constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

The implementation stage also takes into account the overall system structure which includes the various primary components of the system that are needed for the development of the overall system which has to be constructed.

We will discuss the overall system dividing the system into two primary groups.

- Primary components
- Proposed system components

### 4.4.1 PRIMARY COMPONENTS

The primary technology or components which are used for the development of the system consists of three main components. The system consists of primitive image processing components which are used to implement the basic primitive operation of the system. It also consists of components which optimize Parallel processing, thereby improving CPU utilization in the case of image processing. The final component which is required for the system consists of a threading library which implements the producer and consumer scenarios which are needed for the hand gesture system. Here we use

- OpenCV libraries (Open Source Computer Vision Library)
- Intel® Threading Building Block
- Boost C++ Libraries

**OpenCV**

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 and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. 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.

Officially launched in 1999, the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. Some of the basic components used are

## *Mat*

Mat is basically a class with two data parts: the matrix header (containing information such as the size of the matrix, the method used for storing, the address at which the matrix is stored, and so on) and a pointer to the matrix containing the pixel values (taking any dimensionality depending on the method chosen for storing). The matrix header size is constant. However, the size of the matrix itself may vary from image to image and usually is larger by orders of magnitude. OpenCV is an image processing library. It contains a large collection of image processing functions. To solve a computational challenge, most of the time you will end up using multiple functions of the library. Because of this, passing images to functions is a common practice. We should not forget that we are talking about image processing algorithms, which tend to be computationally heavy. The last thing we want to do is to further decrease the speed of the program by making unnecessary copies of potentially large images. To tackle this issue OpenCV uses a reference counting system. The idea is that each Mat object has its own header; however the matrix may be shared between two instances of them by having their matrix pointers point to the same address. Moreover, the copy operators will only copy the headers and the pointer to the large matrix, not the data itself.

## *Mask operations on matrices*

Mask operations on matrices are quite simple. The idea is that we recalculate each pixels value in an image according to a mask matrix (also known as kernel). This mask holds values that will adjust how much influence neighboring pixels (and the current pixel) have on the new pixel value. From a mathematical point of view we make a weighted average, with our specified values.

### *Smoothing Images*

Smoothing, also called blurring, is a simple and frequently used image processing operation. There are many reasons for the use of smoothing algorithms; one of the important uses is for noise reduction. Some of the algorithms available are

- Blur
- Gaussian Blur
- Median Blur
- Bilateral Filter

### *Dilating and Eroding*

#### *Dilation*

This operation consists of convoluting an image 'A' with some kernel (B), which can have any shape or size, usually a square or circle. The kernel 'B' has a defined anchor point, usually being the center of the kernel. As the kernel 'B' is scanned over the image, we compute the maximal pixel value overlapped by 'B' and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name dilation).

### *Erosion*

This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel. As the kernel ‘B’ is scanned over the image, we compute the minimal pixel value overlapped by ‘B’ and replace the image pixel under the anchor point with that minimal value.

### *Basic Thresholding Operations*

This is the simplest segmentation method. This separates out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels. To differentiate the pixels we are interested in from the rest (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a threshold (determined according to the problem to solve). Once we have separated properly the important pixels, we can set them with a determined value to identify them (i.e. we can assign them a value of 0(black), 255(white) or any value that suits your needs). OpenCV offers the function threshold to perform thresholding operations such as Threshold Binary, Inverted Threshold Binary, Truncate, Threshold to Zero and Inverted Threshold to Zero.

## Find Contours

The find contours is a method which is found in OpenCV, the method consists of an algorithm which finds a contour of a bound object and using deviation of the object detail from the environment. The finds contour uses a binary image.

- ***findContours (InputOutputArray image, OutputArrayOfArrays contours, int mode, int method, Point offset=Point())***

**Draw Contours**

The draw contours is a method which is found in OpenCV, the method consists of an algorithm which draws the contours of a bound object using the points derived from the find contour method. The above method can be applied to any image which is supplied as input image. The contour is determined by the array which is given as input to the function, the method also gives the users the privilege to specify the color and the type of fill which is required for the output image.

- ***drawContours (InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar& color, int thickness=1, int lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point offset=Point() )***

**Convex Hull**

The Convex hull is a method which is found in OpenCV, the method consists of an algorithm which draws the convex hull (polygon) for the given bound object using the points derived from the find contour method.

The hull uses the positive deviation of the points which form the polygon hull which enclosed the given contour which is detected.

- ***convexHull(InputArray points, OutputArray hull, bool clockwise=false, bool returnPoints=true )***

**Convexity Defects**

The Convexity defects is a method which is found in OpenCV, the method consists of an algorithm which finds the convex hull (polygon) for the given bound object using the points derived from the find contour method. The hull uses the positive deviation of the points which form the polygon hull which enclosed the given contour which is detected. This method is useful in finding the tip of the finger during hand detection.

- ***convexityDefects(InputArray contour, InputArray convexhull, OutputArray convexityDefects)***

**Moments**

The moment method is used to calculate all of the moments up to the third order of a polygon or rasterized shape which can be determined by the contour which has been detected using find contour method.

- ***moments(InputArray array, bool binaryImage=false )***

## HAAR Feature-based Cascade Classifier for Object Detection

The object detector described below has been initially proposed by Paul Viola [Viola01] and improved by Rainer Lienhart [Lienhart02]. First, a classifier (namely a cascade of boosted classifiers working with Haar-like features) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a “1” if the region is likely to show the object (i.e., face/car), and “0” if otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily “resized” in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

## Named Window

It is used to create a new window which is used to display the output of the program, which can be either an image or another data which must be displayed on the window.

- ***namedWindow(const string& winname, int flags)***

**Image show**

It is used to displays an image in the specified window. It gets the image in matrix (mat object) format after which the image is displayed to the user.

- ***imshow(const string& winname, InputArray mat)***

The above few function are frequently used by the proposed system to function hence if becomes importance to discuss its functionality.

**Introducing Intel® Threading Building Blocks**

Intel® Threading Building Blocks (Intel® TBB) is a library that supports scalable parallel programming using standard ISO C++ code. It does not require special languages or compilers. It is designed to promote scalable data parallel programming. Additionally, it fully supports nested parallelism, so you can build larger parallel components from smaller parallel components. To use the library, you specify tasks, not threads, and let the library map tasks onto threads in an efficient manner.

Many of the library interfaces employ generic programming, in which interfaces are defined by requirements on types and not specific types. The C++ Standard Template Library (STL) is an example of generic programming. Generic programming enables Intel® Threading Building Blocks to be flexible yet efficient. The generic interfaces enable you to customize components to your specific needs.

**Intel® Threading Building Blocks** enables you to specify logical parallelism instead of threads. Most threading packages require you to specify threads. Programming directly in terms of threads can be tedious and lead to inefficient programs, because threads are low-level, heavy constructs that are close to the hardware. Direct programming with threads forces you to efficiently map logical tasks onto threads. In contrast, the Intel® Threading Building Blocks run-time library automatically maps logical parallelism onto threads in a way that makes efficient use of processor resources.

**Intel® Threading Building Blocks targets threading for performance.** Most general-purpose threading packages support many different kinds of threading, such as threading for asynchronous events in graphical user interfaces. As a result, general-purpose packages tend to be low-level tools that provide a foundation, not a solution. Instead, Intel® Threading Building Blocks focuses on the particular goal of parallelizing computationally intensive work, delivering higher-level, simpler solutions.

**Intel® Threading Building Blocks is compatible with other threading packages**. Because the library is not designed to address all threading problems, it can coexist seamlessly with other threading packages.

**Intel® Threading Building Blocks emphasizes scalable, data parallel programming**. Breaking a program up into separate functional blocks, and assigning a separate thread to each block is a solution that typically does not scale well since typically the number of functional blocks is fixed. In contrast, Intel® Threading Building Blocks emphasizes data-parallel programming, enabling multiple threads to work on different parts of a collection. Data-parallel programming scales well to larger numbers of processors by dividing the collection into smaller pieces. With data-parallel programming, program performance increases as you add processors.

**Intel® Threading Building Blocks relies on generic programming.** Traditional libraries specify interfaces in terms of specific types or base classes. Instead, Intel® Threading Building Blocks uses generic programming. The essence of generic programming is writing the best possible algorithms with the fewest constraints. The C++ Standard Template Library (STL) is a good example of generic programming in which the interfaces are specified by requirements on types. For example, C++ STL has a template function sort that sorts a sequence abstractly defined in terms of iterators on the sequence.

## Boost Threading Library

Boost provides free peer-reviewed portable C++ source libraries. We emphasize libraries that work well with the C++ Standard Library. Boost libraries are intended to be widely useful and usable across a broad spectrum of applications. The Boost license encourages both commercial and non-commercial use. We aim to establish "existing practice" and provide reference implementations so that Boost libraries are suitable for eventual standardization. Ten Boost libraries are included in the C++ Standards Committee's Library Technical Report (TR1) and in the new C++11 Standard. C++11 also includes several more Boost libraries in addition to those from TR1.

## Getting Started

Boost works on almost any modern operating system, including UNIX and Windows variants. Follow the Getting Started Guide to download and install Boost. Popular Linux and Unix distributions such as Fedora, Debian, and NetBSD include pre-built Boost packages. Boost may also already be available on your organization's internal web server.

A boost::thread object represents a single thread of execution, as you would normally create and manage using your operating system specific interfaces. For example: on POSIX systems, a Boost thread uses the Pthreads API, and on Win32 it uses the native CreateThread and related calls. Because Boost abstracts away all the platform-specific code, you can easily write sophisticated and portable code that runs across all major platforms. A thread object can be set to a special state of not-a-thread, in which case it is inactive (or hasn't been given a thread function to run yet).

A boost::thread object is normally constructed by passing the threading function or method it is to run. There are actually a number of different ways to do so. I cover the main thread creation approaches below. The main() function, we then wait for the worker thread to complete using the join() method. This will cause the main thread to sleep until the worker thread completes (successfully or otherwise).

## A Thread Function

The simplest threading scenario is where you have a simple (C-style) function that you want to run as a separate thread. You just pass the function to the boost::thread constructor, and it will start running. You can then wait for the thread to complete by calling join()

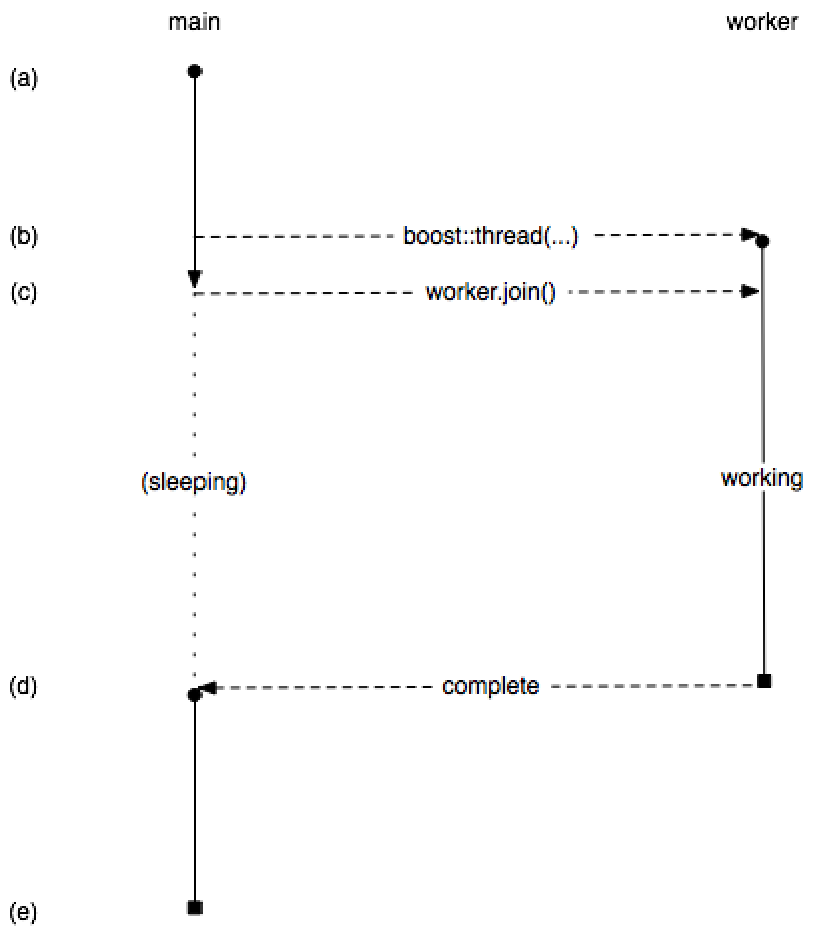


**Figure 4.5 Execution of Threads for Producer and Consumer**

## Installation in Windows

The description here was tested on Windows 7 SP1. Nevertheless, it should also work on any other relatively modern version of Windows OS.

## Installation by Using the Pre-built Libraries

- Launch a web browser of choice and go to our page on Sourceforge.
- Choose a build you want to use and download it.
- Make sure you have admin rights. Unpack the self-extracting archive.
- Set the installation path for the files in the extract.

### Building the library

- Make sure you have a working IDE with a valid compiler. In case of the Microsoft Visual Studio just install it and make sure it starts up.

- Install CMake. Simply follow the wizard, no need to add it to the path. The default install options are OK.

- Now start CMake and find the folder to the source of the project. Also specify the build source output.

- Configure to the environment to which the CMake should build the final project to in our scenario 'Visual Studio X'

- Since we are using ‘Intel Threading Building Block’ for improving the quality of parallel processing in the program.

- Build the project using OpenCV.

- After this build the required Libraries, binaries and dll using visual studio.

- After the build is completed, link the libraries, binaries and dll which are required program. *Note:* For the final build we just need to link the binaries and dll required by the application.

- Now we must download and install the boost setup files, after the download is completed we can select the required files and libraries which are needed for our given program. This may include release, debug tools etc.

- Link the libraries, binaries and dll required to the visual studio compiler for our project. After this process is done we will have the required environment to develop the proposed system.

**Proposed System Component**

Here we will describe the various components needed for the proposed system along with information on other components in the given system.

## Input Devices

The trackball, the joystick and the mouse are extremely successful devices for hand-based computer input. Current Implementation of Glove-based gesture interfaces require the user to wear a cumbersome device, and generally carry a load of cables that connect the device to a computer. Devices such as controllers for games, security systems and television remotes require the user to either push a button or touch a screen for their functioning. Over the centuries, these devices have always been improved and enhanced by advancements in technology, making them easier to operate from the user's point of view.

In the past decade, a lot of devices that were previously button-operated have been replaced successfully with touch-technology, in which the user generates a command by simply touching a screen. Today, we recognize a definite possibility of reducing these human-machine interactions to a simple hand gesture. Hence, we use our hand gestures to perform operations based on our desire. This kind of human-machine interface would allow a user to control a wide variety of devices through hand gestures.

## HAAR - Like Features

HAAR-like features are digital image features used in object detection algorithms. The main purpose of HAAR- like features is to meet the real-time requirements, while providing speed, accuracy and robustness against different backgrounds and lighting conditions. The idea behind HAAR-like features is to recognize objects in an image based on the value of simple features, instead of using raw pixel values directly. The HAAR-like features can efficiently reduce/increase the in-class/out-of class variability, thus making classification easier. A HAAR-like feature in simple terms is a template of multiple connected black and white rectangles as shown in figure below

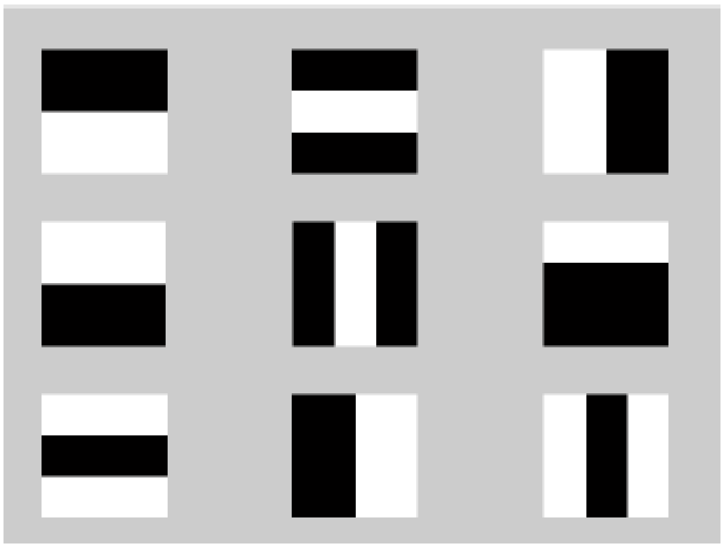


**Figure 4.6 HAAR Detection for hand detection**

The value of a HAAR-like feature is the difference between the sums of the pixels values within the black and white rectangular regions. The concept of integral image is used to compute a rich set of HAAR-like features. The simplest HAAR-like feature is a two-rectangle feature. The value of this is calculated as the difference between the sums of the pixels within the two rectangles. This will indicate characteristics like edges or borders between light and dark regions. Three-rectangle features indicate for instance a dark line or a dark thin area lying between light regions, depending on the size of the middle rectangle. Using the integral image for computing the sum will allow a two-rectangle feature to be calculated with six references to the integral image, a three-rectangle feature with eight references, a four-rectangle feature with nine references etc.

Our architecture will utilize above proposed techniques and methods to satisfy the requirements designed. The Implementation is modularized following sections:

- Image Capture and Correction
- Image Hand Detection
- Event Triggering

We will describe the various modules in the detail in the following chapter of the project documentation.

### 4.4.2 IMAGE CAPTURE AND CORRECTION

Human gestures constitute a space of motion expressed by the body, face, or hands. Among a variety of gestures, hand gesture is the most expressive and the most frequently used. Gestures have been used as an alternative form to communicate with computers in an easy way. Hence the gestures are obtained for performing various operations.

The web camera is used to capture the gestures or movements made by the user for performing the specific operation. A number of producer threads are formed and the system gets activated. The gestures are obtained as live feed from the user and it is given as an input to the gesture system. The obtained video is first converted to an image. Since the image obtained as it is from the system is not suitable for performing operations, it is converted into the required format.

The image obtained is flipped first so that the position of the image is in correct way. Some people may show their gestures in an upside down way and it must be rotated to the clockwise and anticlockwise direction respectively. The native screen resolution is being calculated with reference to the flipped image. The image with good resolution is useful in performing the operations accurately. Then the conversion ratio is obtained from the camera which captures the video.

The pixel density needs to be calculated in order to increase or decrease the brightness. Hence the image needs to be converted into grayscale image. It can be calculated using various methods. Thus the pixel density is calculated using grayscale image.

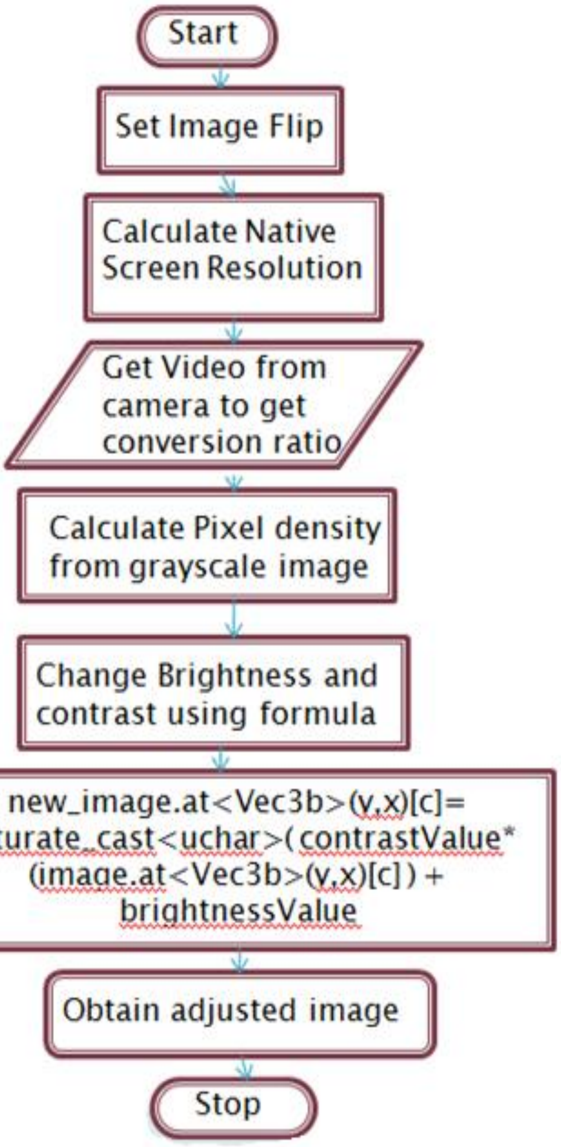
The brightness and contrast of the image needs to be adjusted with the nearby environment in which it is operated. So the image processed lies in accordance with the requirements of the operation. The brightness and contrast can be adjusted using the below relation

**newImage.at<Vec3b>(y,x)[c]=saturateCast($\alpha$*(image.at<Vec3b>(y,x)[c]) + $\beta$);**

Where Contrast Value and Brightness Value Determines the image contrast and brightness. This is dynamically produced by the system based on the current density of light and dark pixels in the given image.

Thus the adjusted image is obtained after a series of operations and the image is transformed to the next module so that it undergoes further operations. The reason for image enhancement is improve the rate of detection of the various objects and their attributes

The flow of the process can be depicted using the flowchart as follows



Start
Set Image Flip
Calculate Native Screen Resolution
Get Video from camera to get conversion ratio
Calculate Pixel density from grayscale image
Change Brightness and contrast using formula
new_image.at<Vec3b>(y,x)[c]= saturate_cast<uchar>( contrastValue* (image.at<Vec3b>(y,x)[c] ) + brightnessValue
Obtain adjusted image
Stop

**Figure 4.7 Flow chart for the Correction Algorithm**

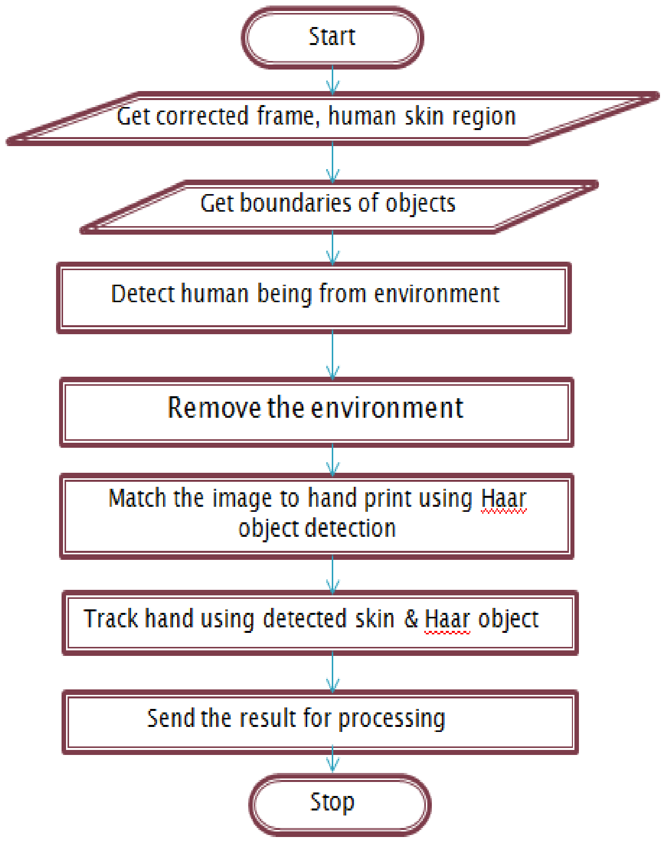
### 4.4.3 IMAGE HAND DETECTION

The corrected image is obtained from the previous module and it serves as input for detecting the skin region of hand. The entire hand is considered for the operation and its movements are detected only with the help of skin region. Firstly, the hand needs to be located exactly by the web camera.

After locating the hand, its boundaries are being calculated using Edge detection. Each one’s hand size varies with one another and hence a particular area cannot be considered for detection. Hence boundary serves as a better idea for detecting one’s hand completely. The skin and edge detection are being used in the image to get the actual human being in the environment. The camera may focus on all the objects that lie in its view and it is necessary to separate the human part from other objects in order to identify the gesture made by them. The Masking process is being done with these detection methods and the background environment is subtracted from the human. This process eliminates all the surroundings that is present nearby the user and the focus is mainly on the human beings with the help of skin detection.

The hand must be separately focused from other parts of the body since it is the one that produces gesture. Hence using Haar detection, the hand gets detected when the user is present before the web camera. This detection helps the system to identify the hand part alone so that its operations can be preceded.

The detection is possible with the help of positive and negative stereotype sets which are used in the Haar based training. The hand groove gets identified with the help of both positive and negative values and hence the proper hand like structure gets detected by the system. The values may change from person to person and hence a type of learning needs to be given to the system in order to identify them correctly. Both the skin detection and Haar object classification is used to improve the accuracy of gesture recognition so that no errors are being made during the process of identification of gesture. The human can make any number of gestures and hence the system must be in the position to identify those gestures accurately.



Start
Get corrected frame, human skin region
Get boundaries of objects
Detect human being from environment
Remove the environment
Match the image to hand print using Haar object detection
Track hand using detected skin & Haar object
Send the result for processing
Stop

**Figure 4.8 Flow chart for the Hand Detection Algorithm**

### 4.4.4 EVENT TRIGGERING

The hand gets detected from the previous module using a mix of skin detection, Edge Detection and Haar object identification. It is one of the primary modules which are needed by the event triggers module. The user is made to be present before the web camera and is allowed to make movements. The system identifies the human from other objects based on the previous module. The position of the hand is also being taken care of because it is necessary for the user to perform gesture within the frame. If the actions are being made outside the frame, then it’s not possible to identify the gesture. The virtual desktop is also included to trigger the event. Thus with the help of absolute position and virtual desktop, the event gets triggered using default mouse device driver. Now the hand is used as a mouse and wherever our hand moves, the pointer also moves accordingly. The producer threads perform the operations that are being triggered by the gesture and the operations are reflected in the desktop. The flow of the process is explained using the flowchart.

There are two parts to the event trigger module one is the analysis of the user and the other is the event trigger. The analysis provide the additional calculation which is needed like contour detection and defect in contour for complex gesture which may be incorporated in the system. The second part of the system triggers the corresponding event onto the mouse system or touch-driver based on the gesture. For the current model of the system we will use the system for simple mouse driver.

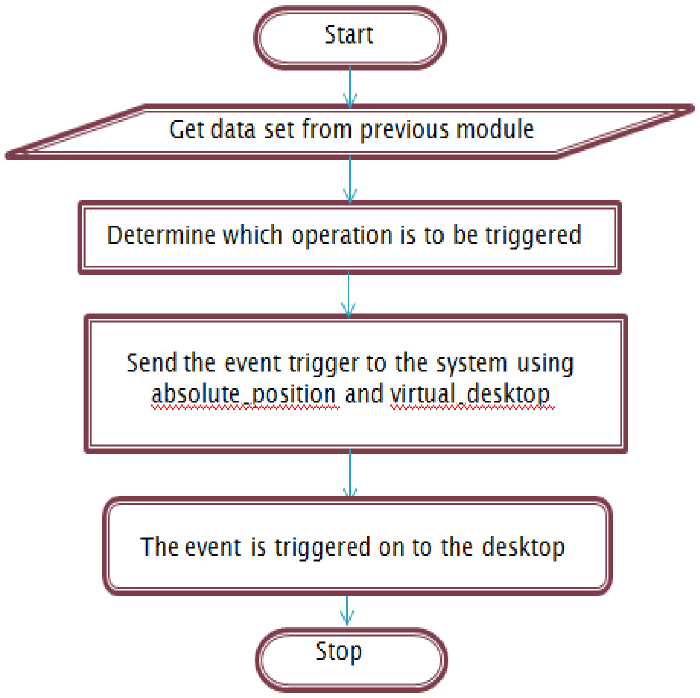

Start
Get data set from previous module
Determine which operation is to be triggered
Send the event trigger to the system using absolute_position and virtual_desktop
The event is triggered on to the desktop
Stop

**Figure 4.9 Flow chart for Event Trigger Algorithm**

The basic operations like click, double click, select can be made possible with the help of this system at the moment. To perform advanced operations, the system must be made to learn complex gestures, for the current system this is not possible because of the time constrain and complexity of the job, hence for the current working model of the project only one mouse operation is mimicked by this system.

# 5. TESTING

As the pervasiveness of software has increased over the decades, testing has become a business-critical part of the software lifecycle. The primary goal of software testing is not to eliminate all possible errors, but to reduce the residual risk, after testing the software, to an acceptable level. Testing is primarily an exercise in “risk mitigation” than an exercise to assure software quality. Software, if released to the market or to customers with defects, affects organizational productivity due to cost of rework post release. This also affects the brand image of the organization, which has a direct impact on its revenues. Testing involves verification and validation techniques, use of formal methodologies and automation tools. Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted. Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

## 5.1 TYPES OF TESTING

### 5.1.1 BLACK BOX TESTING

Black-box testing is a method of software testing those tests the functionality of an application as opposed to its internal structures or 38 workings. Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is only aware of what the software is supposed to do, but not how i.e. when he enters a certain input, he gets a certain output; without being aware of how the output was produced in the first place. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid inputs and determines the correct output. There is no knowledge of the test object's internal structure.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

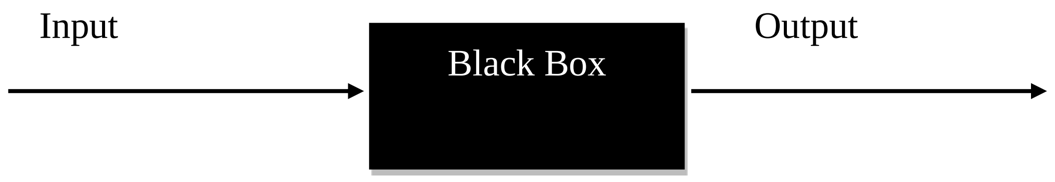


**Figure 5.1 Black Box Testing Diagram**

In penetration testing, black-box testing refers to a methodology where an ethical hacker has no knowledge of the system being attacked. The goal of a black-box penetration test is to simulate an external hacking or cyber warfare attack.

## 5.1.1.1 REQUIREMENTS BASED TESTING

Requirements-Based Testing (RBT) is a rigorous technique for improving the quality of requirements, while deriving the minimum number of test cases to cover 100% of those requirements. RBT is comprised of two techniques: Ambiguity Reviews and Cause-Effect Graphing. An Ambiguity Review is used in the requirements phase of software development to identify ambiguities in functional requirements.

The intent of an Ambiguity Review is to identify anything that is unclear, not concise or ambiguous in the requirements. The elimination of these ambiguities improves the quality of those requirements. Cause-Effect Graphing is a test case design technique that is performed once requirements have been reviewed for ambiguity, followed by a review for content. Requirements are reviewed for content to insure that they are correct and complete. The Cause-Effect Graphing technique derives the minimum number of 40 test cases to cover 100% of the functional requirements to improve the quality of test coverage.

## 5.1.2 WHITE BOX TESTING

White box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black box testing). In white box testing, an internal perspective of the system as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. In Circuit Testing (ICT).

While white box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

White box test design techniques include:

- Data flow testing

In penetration testing, white box testing refers to a methodology where a white hat hacker has full knowledge of the system being attacked. The goal of a white box penetration test is to simulate a malicious insider who has some knowledge and possibly basic credentials to the target system.

### 5.1.2.1 DATA FLOW TESTING

Data flow testing looks at the life cycle of a particular piece of data (i.e. a variable) in an application. By looking for patterns of data usage, risky areas of code can be found and more test cases can be applied. There are four ways data can be used: defined, used in a predicate, used in a calculation and killed. Certain patterns, using a piece of data in a calculation after it has been killed, show an anomaly in the code, and therefore the possibility of a bug.

## 5.2 OTHER TYPES OF TESTING

### 5.2.1 REGRESSION TESTING

Regression testing is any type of software testing that seeks to uncover new software bugs, or regressions, in existing functional and non-functional areas of a system after changes, such as enhancements, patches or configuration changes, have been made to them.

The intent of regression testing is to ensure that a change, such as a bug fix, did not introduce new faults. One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software. Common methods of regression testing include rerunning previously run tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. Regression testing can be used to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.

## 5.3 TESTING IN PARTICULAR

### 5.3.1 UNIT TESTING

In computer programming, unit testing is a procedure used to validate that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method which may belong to a base/super class, abstract class or derived/child class. Ideally, each test case is independent from the others. Mock objects and test harnesses can be used to assist testing a module in isolation. Unit testing is typically done by developers and not by Software testers or end-users.

The six Rules of Unit Testing are

- Write the test first
- Never write a test that succeeds the first time
- Start with the null case, or something that doesn't work
- Do not be afraid of doing something trivial to make the test work
- Loose coupling and testability go hand in hand
- Use mock objects

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. A unit test provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits. Unit testing allows the programmer to refractor code at a later date, and make sure the module still works correctly (i.e. regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed. Readily-available unit tests make it easy for the programmer to check whether a piece of code is still working properly. Good unit test design produces test cases that cover all paths through the unit with attention paid to loop conditions. In continuous unit testing environments, through the inherent practice of sustained maintenance, unit tests will continue to accurately reflect the intended use of the executable and code in the face of any change. Depending upon established development practices and unit test coverage, up-to-the-second accuracy can be maintained.

***Test Case Design – Contrast and brightness enhancement***

Input Image with low, medium and over-lighted environment.
Invalid test is when the environment is too dark or too bright. Say if we remove the entire light to the room then the system will try to adjust the image till the highest feasible value and stop. Alternatively, if the image has too much light, say with 90% of the object overexposed, then the system will try to adjust the image to the lower region till the maximum fix limit.

***Test case T1***:  The webcam is inside an ideal environment (50% brightness). ***Output T1***: The image remains the same and no correction is attempted by the system.

***Test case T2***:  The webcam is inside a room with low light (25% brightness).

***Output T2***: The image is corrected by a small amount and repair is attempted by the system. The image brightness is enhanced and contrast adjusted to improve object exposure.

***Test case T3***: The webcam is inside a room with low light (15% brightness).
***Output T3***: The image is corrected by a large amount and repair is attempted by the system. The image brightness is enhanced and contrast adjusted to improve object exposure.

***Test case T4***: The webcam is inside a room with low light (10% brightness).
***Output T4***: The image is corrected by a very large amount and repair is attempted by the system. The image brightness is enhanced and contrast adjusted to improve object exposure.

***Test case T5***: The webcam is inside a room with high lighting (75% brightness).
***Output T5***: The image is corrected by a small amount to the negative axis and repair is attempted by the system. The image brightness is enhanced and contrast adjusted to reduce object exposure.

***Test case T6***: The webcam is inside a room with high lighting (85% brightness).
***Output T6***: The image is corrected by a large amount to the negative axis and repair is attempted by the system. The image brightness is enhanced and contrast adjusted to reduce object exposure.

***Test case T7***: The webcam is inside a room with high lighting (90% brightness).
***Output T7***: The image is corrected by a very large amount to the negative axis and repair is attempted by the system. The image brightness is enhanced and contrast

adjusted to reduce object exposure. But, the objects are not correctly detected by the system.

**Test report ID:** 001

**Product:** Black Box

**Module:** Image Enhancement

**Table 5.1 Testing of the image enhancement of the system**

| **Test Case ID** | **Input Video Feed** | **Expected output Video Feed** | **Obtained output** | **Pass/Fail** |
|---|---|---|---|---|
| T1 | 50% Brightness | 50%-57% Brightness | 53% Brightness | Pass |
| T2 | 25% Brightness | 50%-57% Brightness | 51% Brightness | Pass |
| T3 | 15% Brightness | 50-57% Brightness | 40% Brightness | Fail |
| T4 | 10% Brightness | <50% Brightness | 28% Brightness | Pass |
| T5 | 75% Brightness | 50%-57% Brightness | 55% Brightness | Pass |
| T6 | 85% Brightness | 50%-57% Brightness | 62% Brightness | Fail |
| T7 | 90% Brightness | >50% Brightness | 72% Brightness | Pass |

***Test Case Design – Image Hand Detection***

Input Image with/without fist and palm in the given webcam view frame.
Here we will test the rate of detection of both fist and palm using the HAAR cascade object detection algorithm, the system will be tested in low, normal and high light conditions and a check of the number of true and false positives of the object detection in the given frame is carried out. If the object is not detected in the stipulated time then it is considered as a failure also.

***Test case T1***: The webcam is inside an ideal environment (50% brightness) and fist is present in the given frame.
***Output T1***: The algorithm will detect the fist within 10 seconds.

***Test case T2***: The webcam is inside an ideal environment (50% brightness) and palm is present in the given frame.
***Output T2***: The algorithm will detect the palm within 10 seconds.

***Test case T3***: The webcam is inside an ideal environment (50% brightness) and no fist is present in the given frame.
***Output T3***: The algorithm should not detect the fist for at-least 5 minutes.

***Test case T4***: The webcam is inside an ideal environment (50% brightness) and no palm is present in the given frame.
***Output T4***: The algorithm should not detect the palm for at-least 5 minutes.

***Test case T5***: The webcam in a non-ideal room (change in lighting is possible) and fist or palm is not present

***Output T5***: The algorithm will not detect any object which maybe assumed as fist for a minimum of 10 minutes.

**Test report ID:** 002

**Product:** Black Box

**Module:** Image Analysis & processing

**Table 5.2 Testing of the Image Analysis & processing of the system**

| Test Case ID | Input Video Feed | Expected output | Obtained output | Pass/Fail |
|---|---|---|---|---|
| T1 | 50% Brightness with fist | Detection of the fist in <10 seconds | 6 seconds | Pass |
| T2 | 50% Brightness with palm | Detection of the fist in <10 seconds | 8 seconds | Pass |
| T3 | 50% Brightness without fist | Should not detect any fist for <5 minutes | 6 minutes | Pass |
| T4 | 50% Brightness without palm | Should not detect any palm for <5 minutes | 6 minutes | Pass |
| T5 | Non-ideal environment without fist or palm in frame | Should not detect any palm for <10 minutes | False detection of object within 4 minute. | Fail |

***Test Case Design – Event triggering***

For Input we trigger the function which performs the corresponding event on the operating system.

Invalid test is when an operation performs another event when the event trigger function is called. Say when we call the 'double left click' operation and the 'single left click' is performed.

***Test case T1***: Call the function which triggers the 'Single Left Click' operation ***Output T1***: The Object is triggered by the 'Single Left Click' event.

***Test case T2***: Call the function which triggers the 'Double Left Click' operation ***Output T2***: The Object is triggered by the 'Double Left Click' event.

***Test case T3***: Call the function which triggers the 'Single Right Click' operation ***Output T3***: The Object is triggered by the 'Double Left Click' event.

***Test case T4***: Call the function which triggers the 'Double Right Click' operation
***Output T4***: The Object is triggered by the 'Double Right Click' event.

**Test report ID:** 003

**Product:** Black Box

**Module:** Event triggering Module

**Table 5.3 Testing of the Event triggering Module of the system**

| Test Case ID | Input | Expected Output | Obtained Output | Pass/Fail |
|---|---|---|---|---|
| T1 | Single Left Click | Single Left Click | Single Left Click | Pass |
| T2 | Double Left Click | Double Left Click | Double Left Click | Pass |
| T3 | Single Right Click | Double Left Click | Single Right Click | Fail |
| T4 | Double Right Click | Double Right Click | Double Right Click | Pass |

# 6. EXPERIMENT RESULTS AND DISCUSSION

The system was implemented using the OpenCV and parallel process optimizes to improve performance.

## 6.1 Brightness and Contrast Control

The brightness and contrast system is used to adjust the image details to improve the quality of tracking and quality of detection of the objects. The tabular column below shows the variation of detection of the object based on the skin color as a factor. As we see dark skinned people have better detection rate than without the image enhancement. Similarly people with mild skin tone have also been improved by a small margin. But due to over-exposure in certain scenarios the detection of skin of fair people reduces.

In Table 6.1, shows a HSV based skin detection rate which calculates if it is actually human skin based on the various races ‘or’ rather the skin tone.

**Table 6.1 Comparison Results of Skin Detection with and without enhancement**

| **Operation** | **System Tracking Rate** | |
|---|---|---|
| | **Original System** | **Proposed System** |
| Dark Skin Human | 33.2% | 56.3% |
| Mild Tone Skin Human | 55.3% | 60.5% |
| Fair Skin Human | 50.1% | 47.5% |

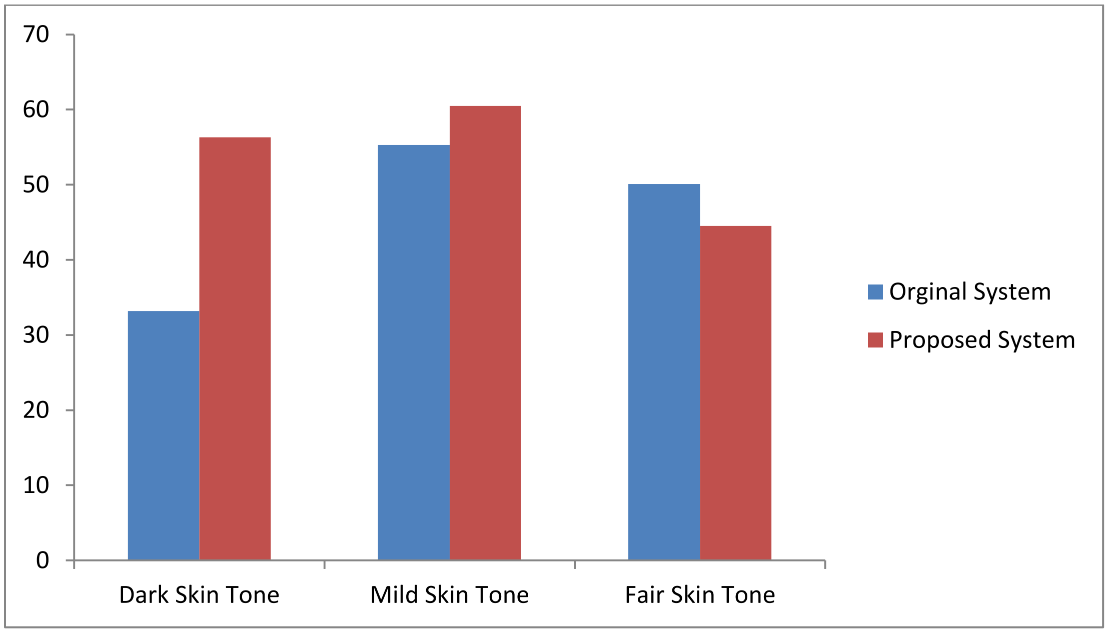


**Figure 6.1 Detection for hand based on Skin tone.**

Figure 6.1 shows that the new system has better skin detection of the human hand which has to be detected properly in order for the tracker to keep accurate lock on the given human hand. The above graph shows that compared to normal detection of dark skin tone people, the new system has substantial increased from the conventional skin detection, whereas for mild skin tone human beings detection rate have been increased by a small amount and for fair skin tones the rate of detection falls by a small margin.

## 6.2 Detection of Object

The tabulation below will give a rough idea about object detection and processing with respect to the various techniques available for the system. The criteria for measurement are both accuracy and time. The accuracy of the system determines the rate at which the correct object is tracked by the system, whereas time refers to the time which the system requires to ensuring the correct object is tracked.

**Table 6.2 Object Detection Techniques its Accuracy and Time Complexity**

| **Operation** | **Time (at 2.4GHz)** | **Accuracy (at 2.4GHz)** | **Comments** |
|---|---|---|---|
| Background Subtraction | Less than 7 seconds (Primary delay) Less than 1 second for secondary detection. | 34% | The background detection is not adequate since the environment's dynamic nature is not accounted. |
| Skin Detection | Less than 1 seconds | 30% | The skin detection technique using HSV color model is less accurate is some environmental lighting |
| Haar Based Detection | 4 - 20 seconds based on environment | 64.2% | The Haar object tracker is the best as far as detection goes; the only drawback is that the rate of tracking is directly proportional to the environmental factors. |

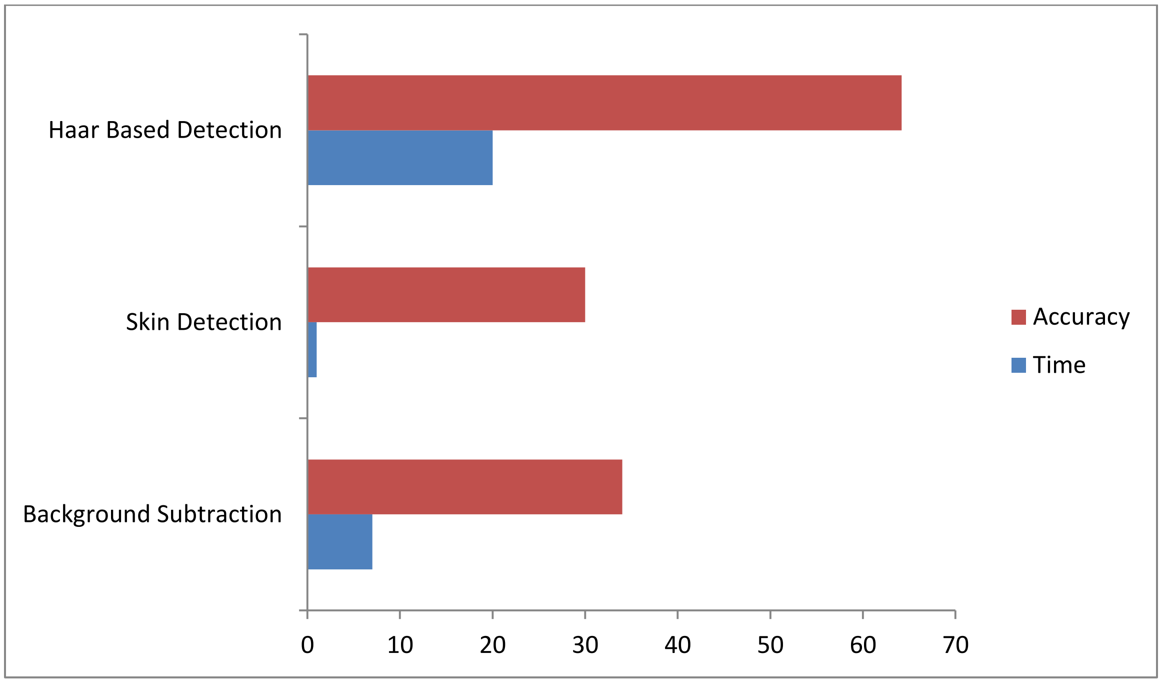


**Figure 6.2 Object Detection Techniques its Accuracy and Time Complexity**

Figure 6.2 explains how Accurate tracking of the hand is done using HAAR-like features, while gesture recognition is done by computing orientation histograms of video frames. The HAAR based system uses positive and negative data sets (images) to make the XML file which consists of the threshold and detection values. Since the time complexity for HAAR based systems is really high, we use CAMSHIFT after the initial detection of the object. This method tracks the object using mean-shift and histogram. Though CAMSHIFT improves the tracking rate of the object, accuracy is reduced in certain scenarios.

## 6.3 CAMSHIFT Tracking

The camshaft tracking method also varies based on the image which is used to perform the tracking operation on the right object. We can use CAMSHIFT on colored images, grayscale images, skin detected images, edge detected images and a combination of the above specified.

**Table 6.3 CAMSHIFT Tracking based on input image**

| **Operation** | **Color** | **Grayscale** | **Skin detected** | **Edge detected** | **Skin & edge detected** |
|---|---|---|---|---|---|
| Tracking (static environment) | 50.2% | 40.3% | 54.6% | 42.2% | 63.2% |
| Tracking (dynamic environment) | 24.5% | 20.2% | 45.1% | 35.2% | 60.2% |

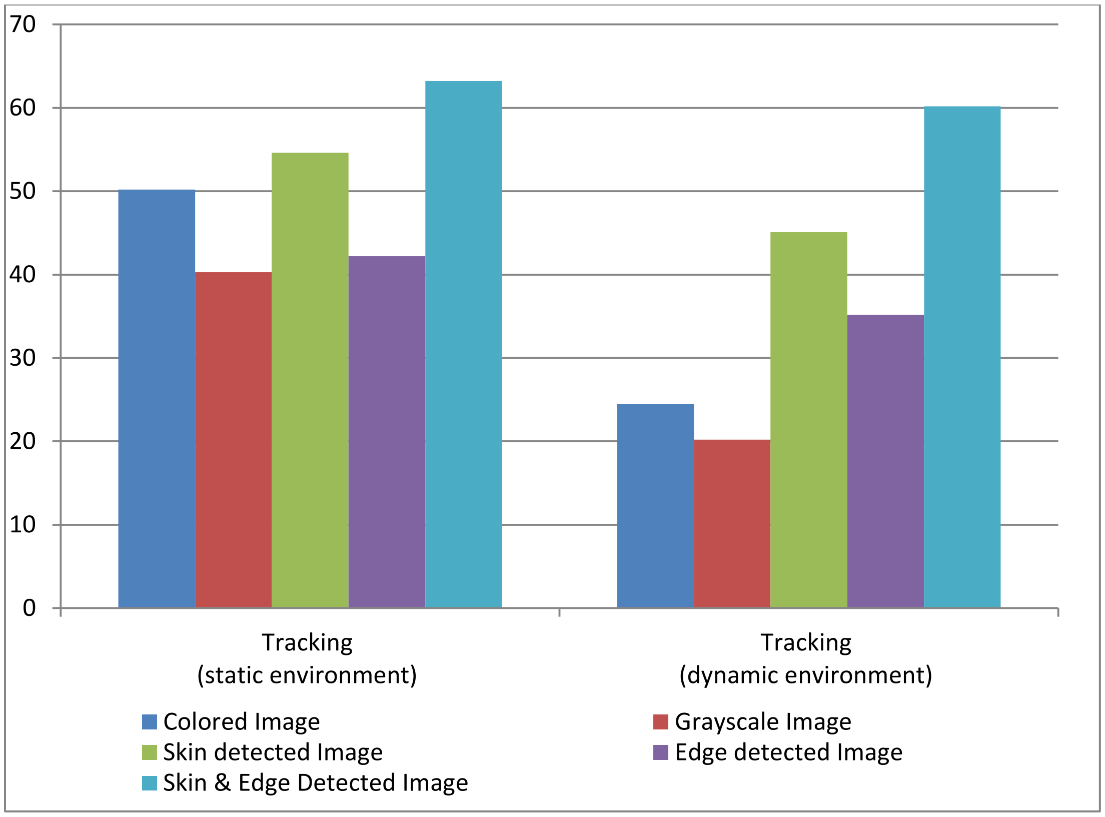


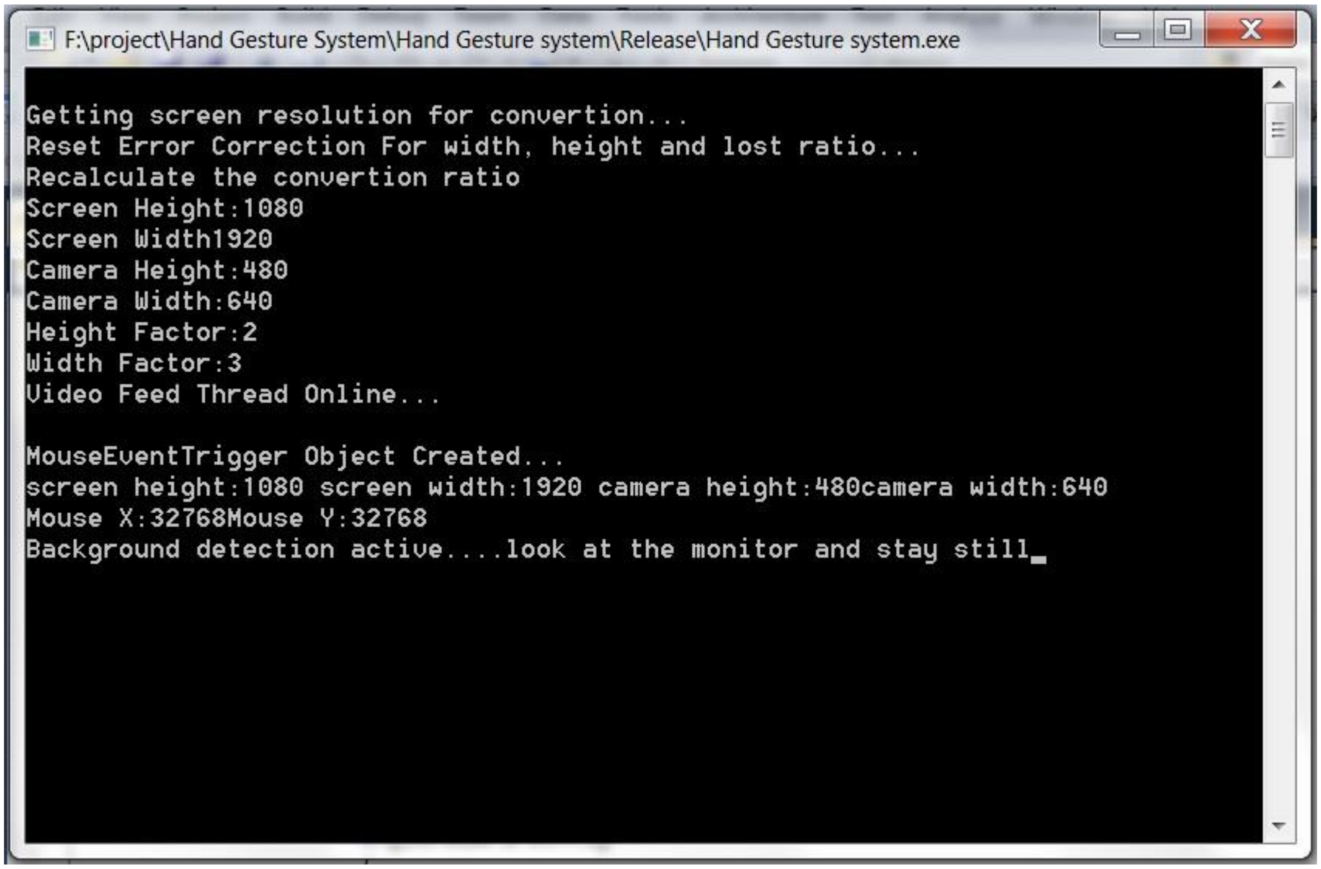
**Figure 6.3 CAMSHIFT Tracking based on input image**

Figure 6.3 show how the information specifies that the best input for CAMSHIFT is an image which is the combination of skin & edge detected image. For color images and grayscale images the algorithm becomes less effective since we detect the background also during tracking of the image and, hence, a drop in accuracy of tracking is observed. In case of skin detection we notice that the tracker loses accuracy of the center of the Region of interest and, hence, drop in accuracy. Edge detection fails to provide the needed accuracy because the system cannot determine which edge is the hand and which is the background edge.

# 7. SCREENSHOTS

The developer's console which is used to test the given hand gesture system and check if the system is properly functioning. This will prove useful for the developer to track error, check if the system is properly functioning.

**Developer Console**



F:\project\Hand Gesture System\Hand Gesture system\Release\Hand Gesture system.exe

```
Getting screen resolution for convertion...
Reset Error Correction For width, height and lost ratio...
Recalculate the convertion ratio
Screen Height:1080
Screen Width1920
Camera Height:480
Camera Width:640
Height Factor:2
Width Factor:3
Video Feed Thread Online...

MouseEventTrigger Object Created...
screen height:1080 screen width:1920 camera height:480camera width:640
Mouse X:32768Mouse Y:32768
Background detection active....look at the monitor and stay still_
```

**Figure 7.1 Developer Console**

The skin detection using HSV color model, here we detect all parts of the image which falls into the color profile.

**HSV Skin detection**

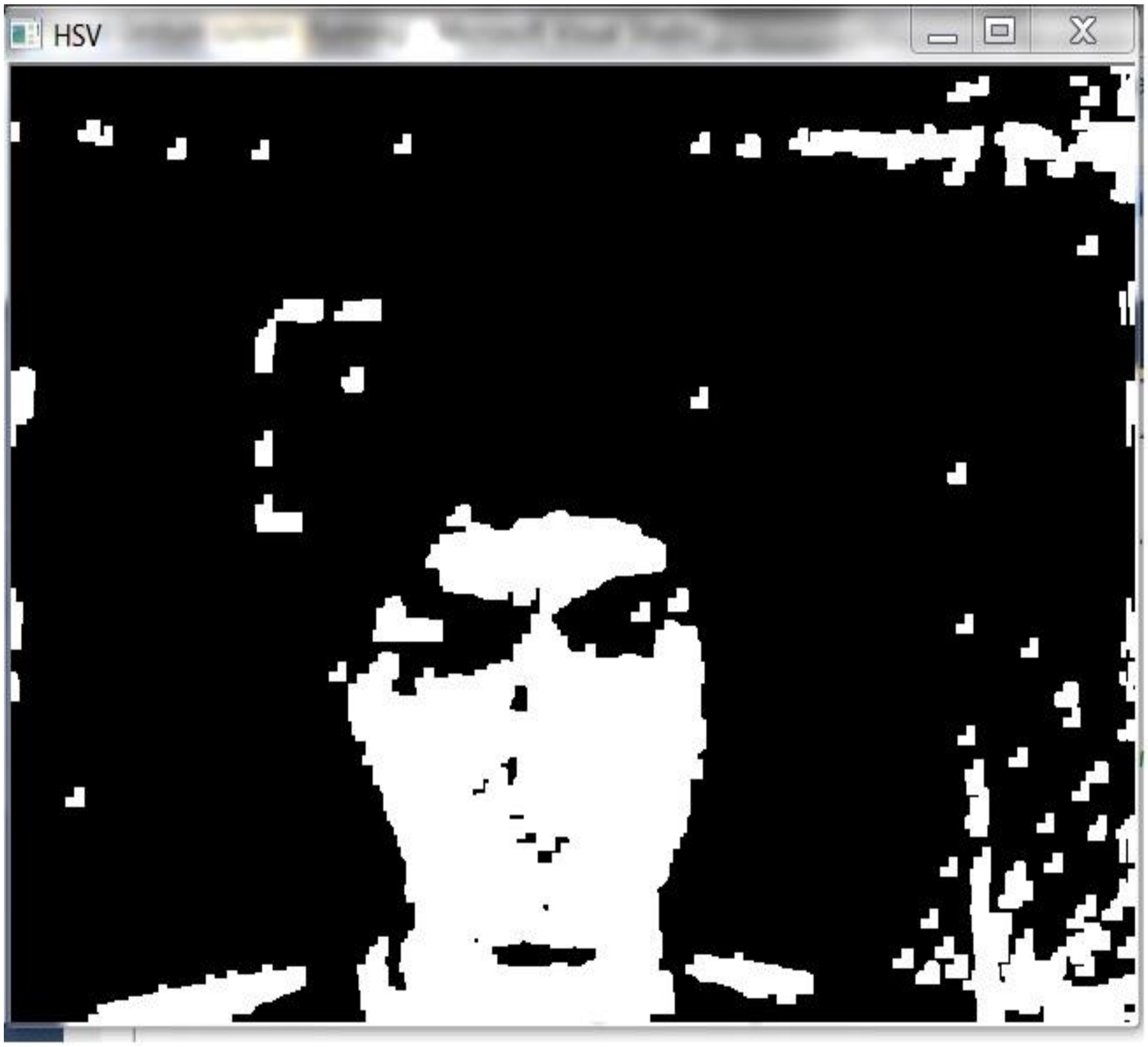


**Figure 7.2 HSV Skin Detection**

Here we detect the edge with the skin detected image which together provides the necessary image which is used for the tracking of the hand and face.

## Edge Detection

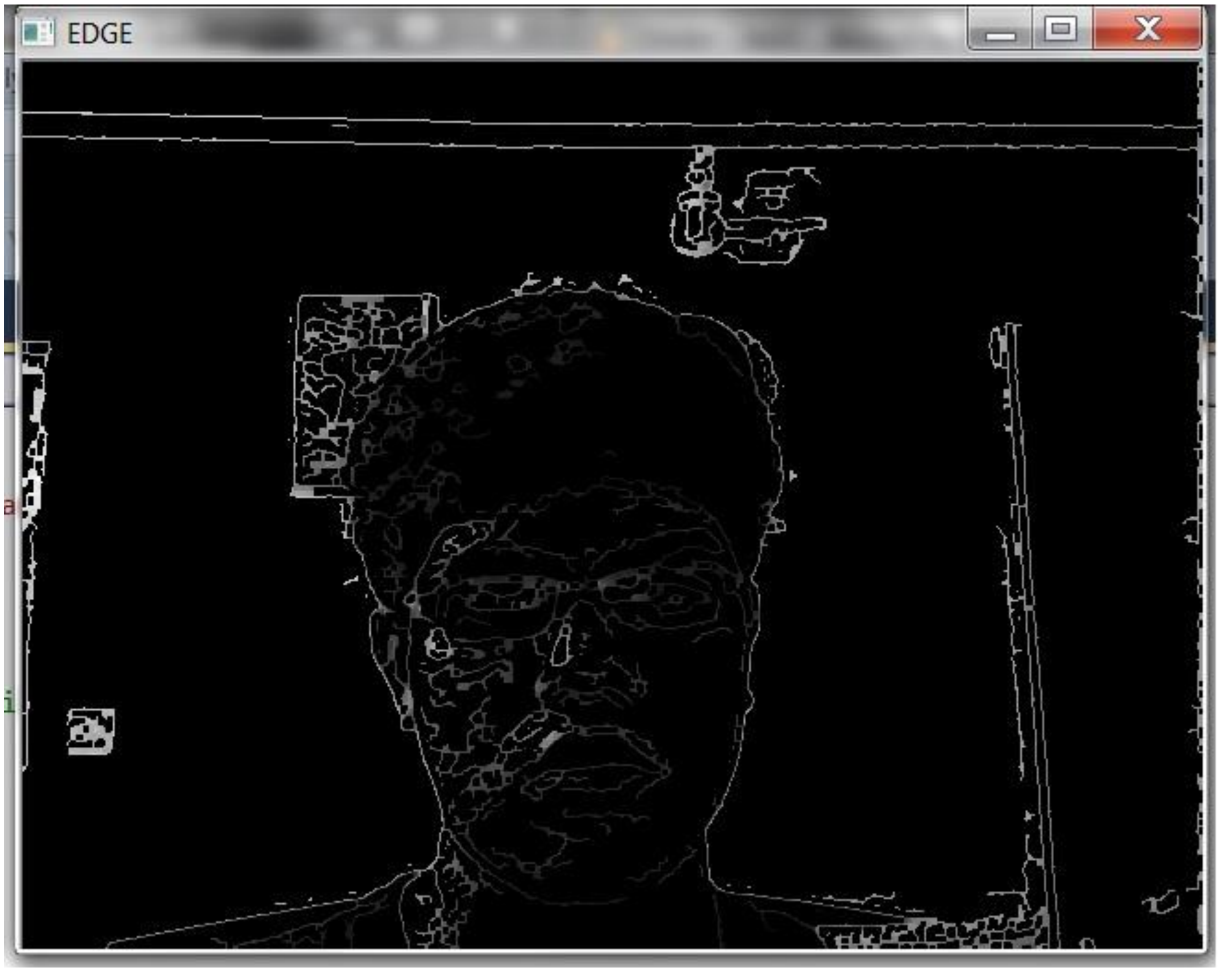


**Figure 7.3 Edge Detection**

The below is the face detection function of the system. The reason for face detection test is too remove any false detection of fist because of the primitive nature of the HAAR-set for both Fist and Palm Detection.

## Face Detection

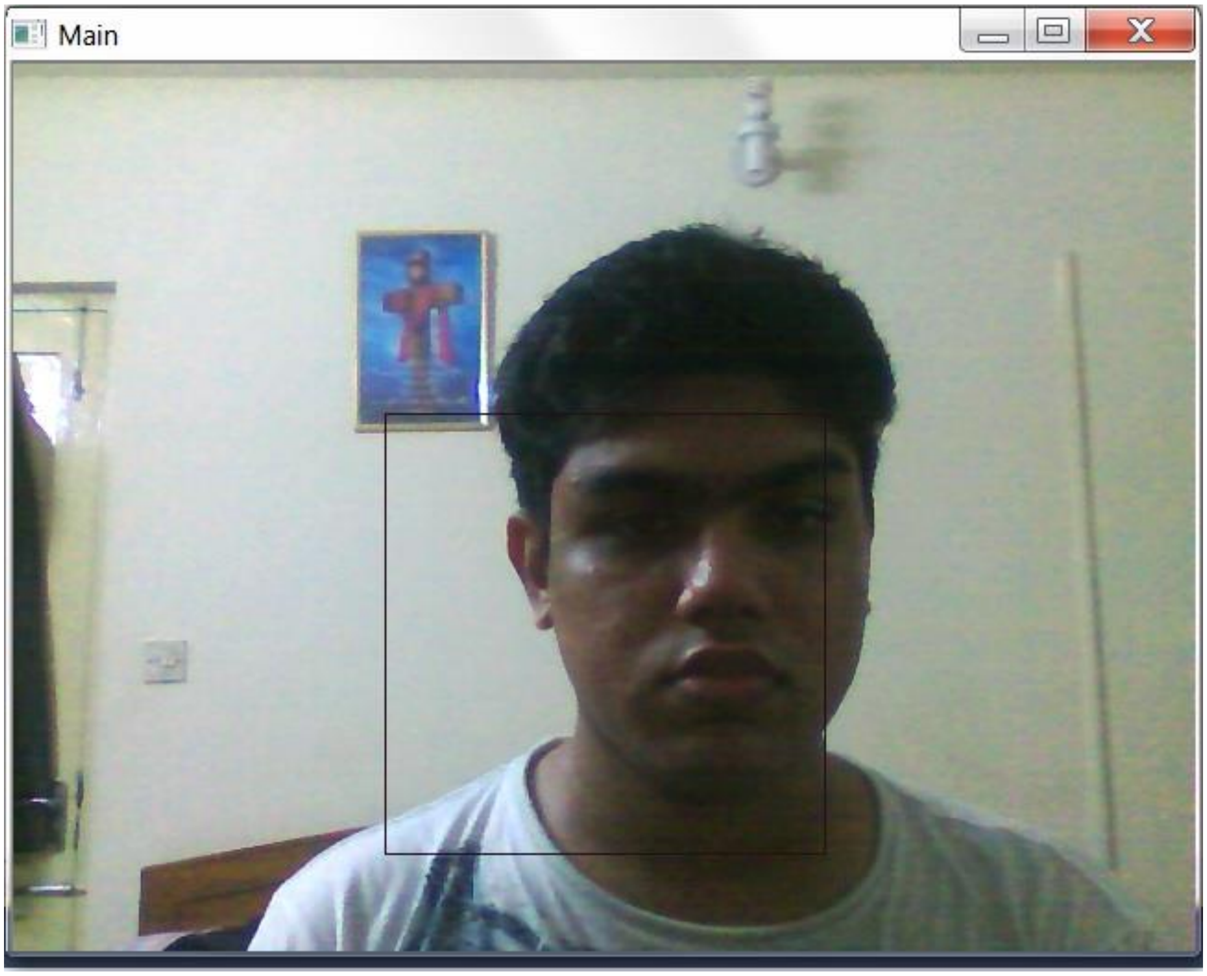
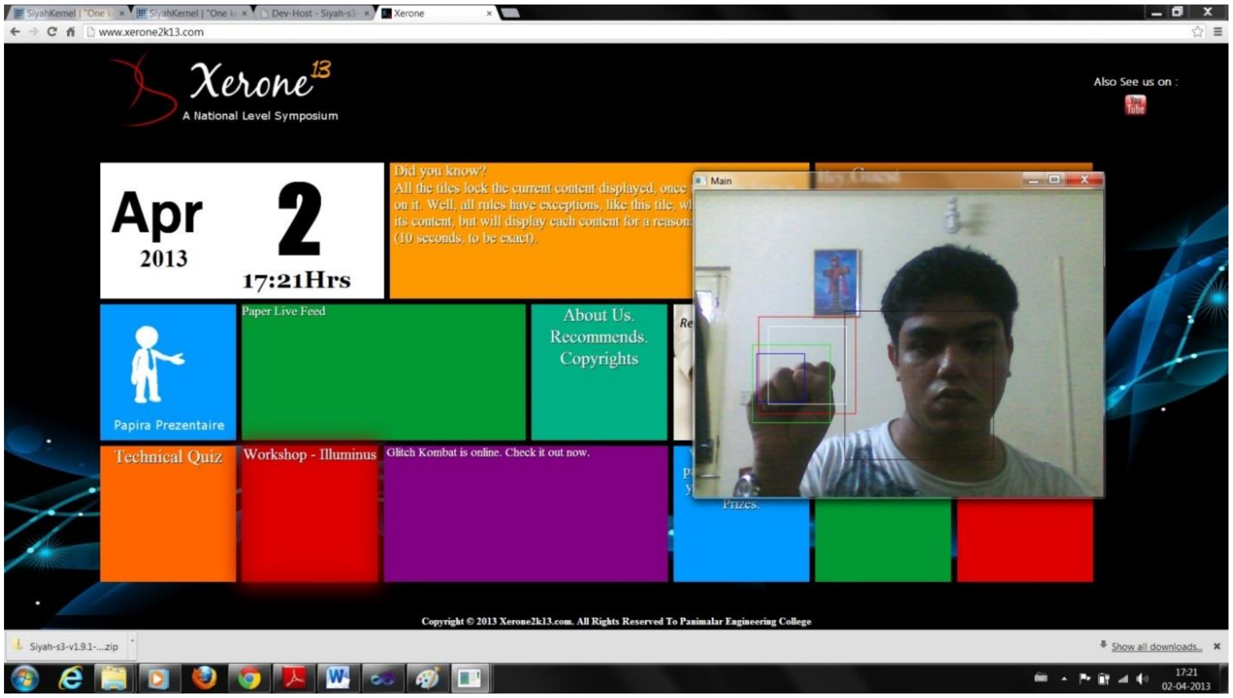


**Figure 7.4 Face Detection**

If the fist or palm is found in the face region the system will throw it as false detection and hence reduce the chances of errors during execution.
The fist is detected using HAAR Based Detection of the fist.

**Fist Detection**

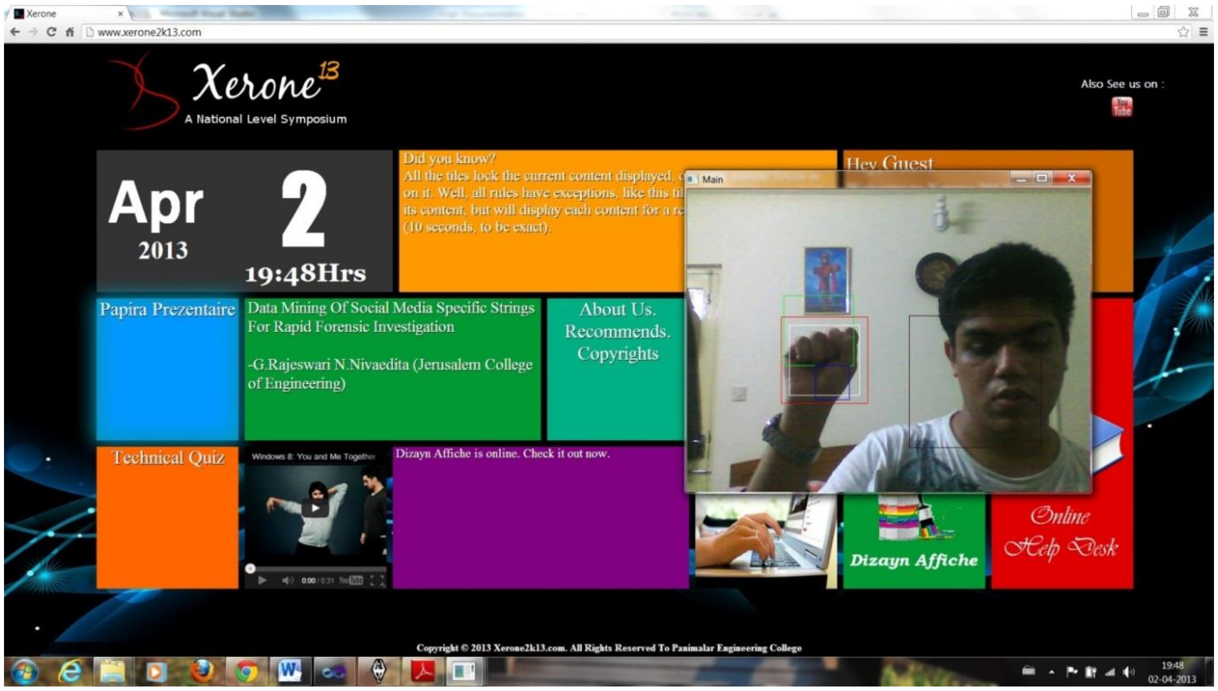


Did you know?
All the tiles lock the current content displayed, once on it. Well, all rules have exceptions, like this tile, wi its content, but will display each content for a reason (10 seconds, to be exact).

Glitch Kombat is online. Check it out now.

Figure 7.5 Fist Detection

**Fist Motion Tracking**


Xerone
www.xerone2k13.com
Xerone 13
A National Level Symposium
Also See us on :
Apr 2013
2
19:48Hrs
Hey Guest
Main
Papira Prezentaire
About Us. Recommends. Copyrights
Technical Quiz
Windows 8: You and Me Together
Dizayn Affiche
Online Help Desk
Copyright © 2013 Xerone2k13.com. All Rights Reserved To Panimalar Engineering College

Did you know?
All the tiles lock the current content displayed, on it. Well, all rules have exceptions, like this ti its content, but will display each content for a re (10 seconds, to be exact).

Data Mining Of Social Media Specific Strings For Rapid Forensic Investigation

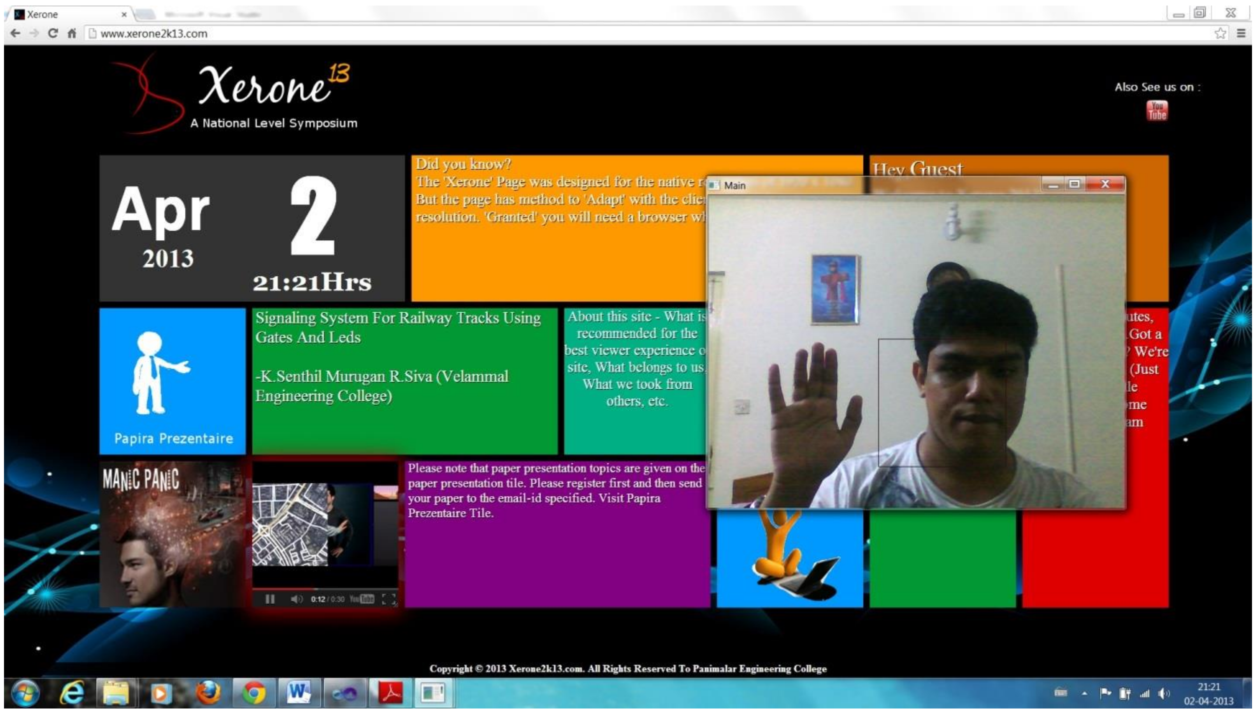
-G.Rajeswari N.Nivaedita (Jerusalem College of Engineering)

Dizayn Affiche is online. Check it out now.

Figure 7.6 Fist Motion Detection

Once the palm is detected the system counts number of detects and if it is above the threshold number then ‘left click’ is triggered.

## Palm Detection – Single Left Click Operation



Did you know?
The 'Xerone' Page was designed for the native r
But the page has method to 'Adapt' with the clie
resolution. 'Granted' you will need a browser wi

Signaling System For Railway Tracks Using Gates And Leds

-K.Senthil Murugan R.Siva (Velammal Engineering College)

About this site - What is recommended for the best viewer experience o site, What belongs to us What we took from others, etc.

Please note that paper presentation topics are given on the paper presentation tile. Please register first and then send your paper to the email-id specified. Visit Papira Prezentaire Tile.

Figure 7.7 Palm Detection

# 8. CONCLUSION AND FUTURE ENHANCEMENT

## 8.1 CONCLUSION

In this project, we have found an innovative way to interact with the computer using a web-camera to record the gesture and perform the hand gesture analysis and processing to mimic the basic operation of the mouse. The system is capable of tracking the fist and palm of a human in most scenarios. The proposed system does not require any additional expensive hardware and, hence, the reproduction of the system is simpler and much cheaper. The current system is capable of tracking and performing single left click operation on any object which is available on the system UI. The proposed system is very primitive and performs a simple left click gesture.

The HAAR dataset which is used for the detection is very primitive and, hence, the detection rate is slow and false positives are still a possibility. The system still is not a good replacement for the current system as far as speed is concern.

## 8.2 FUTURE ENHANCEMENT

So the future system will have a better dataset for the HAAR detection of fist and palm. The current system has incorporated methods to detect and analyze more complex gestures like variation types of click operation, pinch zoom etc. The algorithm which is used for detection and tracking still needs improvement in accuracy and speed. Hence, optimization is required.

# APPENDIX

As we discussed in the implementation section, we do have three main modules which are as follows

- Image Enhancement
- Image Analysis and Processing
- Event Triggering

Since the program cannot be easily divided because of the inter-dependency, hence we will have to give the entire program. The comments on the program code will give information on the various modules involved.

```
// Hand Gesture system.cpp : Defines the entry point for the console application.
// Hand Gesture system v1.0.0.4025 beta

//Define for all necessary headers
#include "stdafx.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
/* _beginthread, _endthread */
#include <process.h>
#include <stddef.h>
//timer for counter
#include <time.h>
```

```
//opencv headers
#include <cv.h>
#include <highgui.h>
#include <cxcore.h>
#include <opencv2/opencv.hpp>
#include <opencv2/core/types_c.h>
#include <opencv2/objdetect/objdetect.hpp>

// The below header contains the various contours detection
#include <myContourDetection.c>
#include <myBackgroundDetection.c>
#include <myHaarHandDetection.c>
#include <myHaarPalmDetection.c>
#include <myHaarFistDetection.c>
#include <myHaarFaceDetection.c>
#include <myPointCalculaterAlgorithms.c>
// hide the windows console which is displayed in the background of the
#include <windows.h>
#include <WinUser.h>
#include <iostream>
#define _WIN32_WINNT 0x0500
//boost
#include <boost\thread.hpp>

//using namespaced
using namespace std;
using namespace cv;



//Hide the console... No one should see it, unless required.
void windowHideConsole()
{
        HWND hWnd = GetConsoleWindow();
        ShowWindow( hWnd, SW_HIDE);
}

//hand structure model for detail gesture the is form the touch UI
//which will be mimiced by the system
struct handRatio
{
        //all distance from the center of the palm
        double legth1, defect1;
        double legth2, defect2;
        double legth3, defect3;
        double legth4, defect4;
        double legth5, defect5;

        double legth6, defect6;
        double legth7, defect7;
        double legth8, defect8;
        double legth9, defect9;
        double legth10, defect10;
};
```

```
/*
        The Class eventTrigger is used to trigger the mouse event on the system.
        This is (3rd)Three Module which is involved with the system <<<<Event
        Triggering
*/

//Event trigger for the mouse events
class eventTrigger
{
        private:

        //mouse varaibles
        int xMouse,yMouse;

        //adapter setting
        int screenHeight,screenWidth;
        int cameraHeight,cameraWidth;
        int totalPixel;

        double virtualScreenRatioHeight;
        double virtualScreenRatioWidth;

        double xRatio,yRatio;

        public:
```

```
	eventTrigger(int ScreenHeight,int ScreenHWidth,int CameraHeight,int 
CameraWidth)
	{
		screenHeight=ScreenHeight;
		screenWidth=ScreenHWidth;
		cameraHeight=CameraHeight;
		cameraWidth=CameraWidth;

		totalPixel = screenWidth * screenHeight;

		//calculate ratio
		yRatio = (double)((double)screenHeight/(double)cameraHeight);
		xRatio = (double)((double)screenWidth/(double)cameraWidth);

		//convertion ratio for the virtual screen
		virtualScreenRatioHeight = (double)(65536/(double)screenHeight);
		virtualScreenRatioWidth = (double)(65536/(double)screenWidth);

		//reset the mouse to the center of the screen
		yMouse = virtualScreenRatioHeight * ((double)screenHeight/2);
		xMouse = virtualScreenRatioWidth * ((double)screenWidth/2);
		setmousePosition(0,0);

		cout<<"\n\nMouseEventTrigger Object Created...\nscreen 
height:"<<screenHeight<<" screen width:"<<screenWidth<<" camera 
height:"<<cameraHeight<<"camera width:"<<cameraWidth;
		cout<<"\nMouse X:"<<xMouse<<"Mouse Y:"<<yMouse;
	}

	void setmousePosition(int dx,int dy)
	{
		dx = (double)virtualScreenRatioWidth  * ((double)dx * xRatio);
		dy = (double)virtualScreenRatioHeight * ((double)dy * yRatio);

		if(((long)(xMouse+dx)<=65535) && ((long)(xMouse+dx)>0))
		{
			xMouse=xMouse+dx;
		}


		if((long)(yMouse+dy)<=65535 && ((long)(yMouse+dy)>0))
		{
			yMouse=yMouse+dy;
		}

		if(xMouse>=0 && xMouse<=65535 && yMouse>=0 &&
yMouse<=65535)
		{
			INPUT input;
			input.type=INPUT_MOUSE;
			//set coordinates
			input.mi.dx=xMouse;
			input.mi.dy=yMouse;
```

```
                    //inital Flag of the mouse

        input.mi.dwFlags=(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_VIR
TUALDESK|MOUSEEVENTF_MOVE);
                    input.mi.mouseData=0;
                    input.mi.dwExtraInfo=NULL;
                    input.mi.time=0;
                    SendInput(1,&input,sizeof(INPUT));
                }
                //cout<<"\nx:"<<xMouse<<" y:"<<yMouse;
        }

        //Right mouse hold
        void setMouseRightHold()
        {
                if(xMouse>=0 && xMouse<=65535 && yMouse>=0 &&
yMouse<=65535)
                {
                    INPUT input;
                    input.type=INPUT_MOUSE;

                    //Point of the mouse
                    input.mi.dx=xMouse;
                    input.mi.dy=yMouse;

                    //SET inital state of the mouse -> Flag varialbe
        input.mi.dwFlags=(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_VIR
TUALDESK|MOUSEEVENTF_MOVE|MOUSEEVENTF_RIGHTDOWN);
                        input.mi.mouseData=0;
                        input.mi.dwExtraInfo=NULL;
                        input.mi.time=0;
                        SendInput(1,&input,sizeof(INPUT));
                }
                //cout<<"\nx:"<<x<<" y:"<<y<<" Hold Right";
        }

        //Right mouse leave
        void setMouseRightLeave()
        {
                if(xMouse>=0 && xMouse<=65535 && yMouse>=0 &&
yMouse<=65535)
                {
                        INPUT input;
                        input.type=INPUT_MOUSE;
                        //Point of the mouse
                        input.mi.dx=xMouse;
                        input.mi.dy=yMouse;

                        //SET inital state of the mouse -> Flag varialbe

        input.mi.dwFlags=(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_VIR
TUALDESK|MOUSEEVENTF_MOVE|MOUSEEVENTF_RIGHTUP);
```

```
//|MOUSEEVENTF_MOVE)|MOUSEEVENTF_RIGHTDOWN|MOUSEEVENT
F_RIGHTUP
                        input.mi.mouseData=0;
                        input.mi.dwExtraInfo=NULL;
                        input.mi.time=0;
                        SendInput(1,&input,sizeof(INPUT));
                }
                //cout<<"\nx:"<<x<<" y:"<<y<<" Leave Right";
        }

        //Left mouse Hold
        void setMouseLeftHold()
        {
                if(xMouse>=0 && xMouse<=65535 && yMouse>=0 &&
yMouse<=65535)
                {
                        INPUT input;
                        input.type=INPUT_MOUSE;

                        //Point of the mouse
                        input.mi.dx=xMouse;
                        input.mi.dy=yMouse;

                        //SET inital state of the mouse -> Flag varialbe

        input.mi.dwFlags=(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_VIR
TUALDESK|MOUSEEVENTF_MOVE|MOUSEEVENTF_LEFTDOWN);
```

```
//|MOUSEEVENTF_MOVE)|MOUSEEVENTF_RIGHTDOWN|MOUSEEVENT
F_RIGHTUP
                        input.mi.mouseData=0;
                        input.mi.dwExtraInfo=NULL;
                        input.mi.time=0;
                        SendInput(1,&input,sizeof(INPUT));
                }
                //cout<<"\nx:"<<x<<" y:"<<y<<" Hold Left";
        }

        //Left mouse leave
        void setMouseLeftLeave()
        {
                if(xMouse>=0 && xMouse<=65535 && yMouse>=0 &&
yMouse<=65535)
                {
                        INPUT input;
                        input.type=INPUT_MOUSE;

                        //Point of the mouse
                        input.mi.dx=xMouse;
                        input.mi.dy=yMouse;

                        //SET inital state of the mouse -> Flag varialbe

        input.mi.dwFlags=(MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_VIR
TUALDESK|MOUSEEVENTF_MOVE|MOUSEEVENTF_LEFTUP);
```

```
//|MOUSEEVENTF_MOVE)|MOUSEEVENTF_RIGHTDOWN|MOUSEEVENT
F_RIGHTUP
                input.mi.mouseData=0;
                input.mi.dwExtraInfo=NULL;
                input.mi.time=0;
                SendInput(1,&input,sizeof(INPUT));
            }
            //cout<<"\nx:"<<x<<" y:"<<y<<" Leave Left";
        }

        //Single CLick of the Left Button
        void setMouseLeftClick()
        {
            setMouseLeftHold();
            setMouseLeftLeave();
            //cout<<"\n>>x:"<<xMouse<<" y:"<<yMouse<<" Single Left Click";
        }

        //Single CLick of the Right Button
        void setMouseRightClick()
        {
            setMouseRightHold();
            setMouseRightLeave();
            //cout<<"\n>>x:"<<xMouse<<" y:"<<yMouse<<" Single Right
Click";
        }
```

```
	//Double CLick of the Left Button
	void setMouseLeftDoubleClick()
	{
		setMouseLeftClick();
		setMouseLeftClick();
		//cout<<"\nx:"<<x<<" y:"<<y<<" Double Left Click";
	}

	//Double CLick of the Right Button
	void setMouseRightDoubleClick()
	{
		setMouseRightClick();
		setMouseRightClick();
		//cout<<"\nx:"<<x<<" y:"<<y<<" Double Right Click";
	}

}*mousePosition;

/*
	The Class cameraSoftware is used to analysis and detection the various
objects and also the image enhancement
	This is (1st and 2nd))First & Second Module which is involved with the
system <<<<Event Triggering
*/
class cameraSoftware
{
	private:
```

```
//sort point array
struct pointWrapper
{
	Point pointValue;
	int sortValues;
};

/*Camera setting*/
bool isFlipImage;

//adapter setting
int screenHeight,screenWidth;
int cameraHeight,cameraWidth;
int correctionHeight,correctionWidth;
int totalPixel;

double xRatio,yRatio;
double xCorrection,yCorrection;

/*The camera variables and the feed which can be used in the other process
of the system*/
VideoCapture *capture; //camera capture variable
bool runCamFeed; //A variable to run the loop of the camera
Mat imgSrcMat; //The Source matrixof the Video Feed

Mat imgDestMat; //The matrix for the Video Feed
Mat imgDestMatHSV; //HSV version of the image
Mat imgDestMatGrayScale; // the matrix for the video feed as gray scale
Mat imgDestMatEdge; //Edge Detection matrix

Mat background; //background matrix
Mat foreground; //foreground matrix

//get screen resolution for the given os
void calculateScreenResolution()
{
	cout<<"\nGetting screen resolution for convertion...";
	RECT desktop;
	// Get a handle to the desktop window
	const HWND hDesktop = GetDesktopWindow();

	// Get the size of screen to the variable desktop
	GetWindowRect(hDesktop, &desktop);
	/*
		The top left corner will have coordinates (0,0)
		and the bottom right corner will have coordinates
		(horizontal, vertical)
	*/
	screenWidth = desktop.right;
	screenHeight = desktop.bottom;

	totalPixel = screenWidth * screenHeight;
```

```
		cout<<"\nReset Error Correction For width, height and lost ratio...";
		correctionHeight=0;
		correctionWidth=0;
		xCorrection=1;
		yCorrection=1;
	}

	void correctionResolution(Point point1,Point point2)
	{
		static int area;
		int width= point2.x - point1.x,height= point2.y - point1.y;

		if(area < (width*height))
		{
			area = width * height;
			correctionWidth = (int)((double)width/(double)2);
			correctionHeight = (int)((double)height/(double)2);

			xCorrection = ((double)cameraWidth/(double)(cameraWidth-
width));
			yCorrection = ((double)cameraHeight/(double)(cameraHeight-
height));

			cout<<"\nCorrection Activated";
			cout<<"\nwidth:"<<width<<"height:"<<height;
		}
	}
```

```
        //Shift of matrix MODULE NEED TO BE REPAIRED
        Mat shiftMatrixPositive(Mat sourceMat,int shiftValueX,int shiftValueY)
        {
                Mat dstMat = Mat::zeros(sourceMat.size(), CV_8UC1);

                //Matrix shift
                for( int i = shiftValueX; i < sourceMat.rows; i++ )
                {
                        for( int j = shiftValueY; j < sourceMat.cols; j++ )
                        {
                                dstMat.at<unsigned char>(i,j) = sourceMat.at<unsigned
    char>(((i+shiftValueX)%sourceMat.rows),((j+shiftValueY)%sourceMat.cols));
                                //cout<<dstMat.at<unsigned char>(i,j)<<" ";
                        }
                }
                return dstMat;
        }

        //Shift of matrix MODULE NEED TO BE REPAIRED
        Mat shiftMatrixNegative(Mat sourceMat,int shiftValueX,int shiftValueY)
        {
                Mat dstMat;
                sourceMat.copyTo(dstMat);

                //Matrix shift
                for( int i = sourceMat.rows-1; i >=shiftValueX; i-- )
                {
                        for( int j = sourceMat.cols-1; j >=shiftValueY ;j-- )
                        {
                                dstMat.at<unsigned char>(i,j) = sourceMat.at<unsigned
    char>(((i+shiftValueX)%sourceMat.rows),((j+shiftValueY)%sourceMat.cols));
                                //cout<<dstMat.at<unsigned char>(i,j)<<" ";
                        }
                }
                return dstMat;
        }

        //edge detection Canny apparently a good algorithm for edge detection
        Mat cannyEdgeDetection(Mat srcGrayScale, int lowerThreshold=9)
        {
                //variables required for detection
                int edgeThresh = 1;
                int lowThreshold=lowerThreshold;
                int const max_lowThreshold = 100;
                int ratio = 3;
                int kernel_size = 3;

                Mat detectedEdges,dst;

                /// Reduce noise with a kernel 3x3
                //detectedEdges = srcGrayScale;
                blur(srcGrayScale, detectedEdges, Size(3,3));
```

```
                /// Canny detector
                Canny(detectedEdges,detectedEdges,lowThreshold,
lowThreshold*ratio,kernel_size);

                 /// Using Canny's output as a mask, we display our result
                dst = Scalar::all(0);

                srcGrayScale.copyTo(dst,detectedEdges);

                return dst;
        }



        //contrast Adjustment and brightness adjustment <my fuzzy logic> -- not the
best but does a bit of the job
        void getHistogramInfo(Mat imgSourceMat, double &contrastValue,int
&brightnessValue)
        {
                double minValue,maxValue,density;
                double highLevelPixel;

                cv::Point minBin,maxBin;

                Mat result;
                int histSize = 20; //from 0 to 20 >> value 5

                /// Set the ranges ( for B,G,R) )
                float range[] = { 0, 256 } ; //the upper boundary is exclusive
                const float* histRange = { range };
                bool uniform = true; bool accumulate = false;



        calcHist(&imgSourceMat,1,0,Mat(),result,1,&histSize,&histRange,uniform,
accumulate);
                minMaxLoc(result,&minValue,&maxValue,&minBin,&maxBin);

                //mid Value for the system
                highLevelPixel = (result.at<float>(0,0) +
result.at<float>(1,0)+result.at<float>(2,0)+result.at<float>(3,0)+result.at<float>(4,
0)+result.at<float>(19,0) + result.at<float>(18,0)+result.at<float>(17,0) +
result.at<float>(16,0)+ result.at<float>(15,0));
                density = (double)(highLevelPixel/totalPixel);

                density = (( 0.1 - (density)) + 1 );
                //cout<<density<<"\n";

                //17-MAXBIN.... my fuzzy logic for brightness adjustment
                brightnessValue = (int)((15-maxBin.y)*2);
                contrastValue = density;
        }

        // Black and White Skin Detection
        Mat blackWhiteImageOfSkin(Mat skinMatrix)
        {
```

```
		Mat blackWhiteImage;

		blur(skinMatrix,skinMatrix,Size(3,3));

		inRange(skinMatrix, Scalar(0, 38,50), Scalar(25, 173, 255),
blackWhiteImage);

		return (blackWhiteImage);
	}

	//changes the contrast and brightness
	Mat contrastBrightnessAdjusted(Mat imgSourceMat,double contrastValue,
int brightnessValue)
	{
		/// Read image given by user
		Mat image = imgSourceMat;
		Mat new_image = Mat::zeros( image.size(), image.type() );

		/// Do the operation new_image(i,j) = alpha*image(i,j) + beta
		for( int y = 0; y < image.rows; y++ )
		{
			for( int x = 0; x < image.cols; x++ )
			{
				for( int c = 0; c < 3; c++ )
				{
```

```
                                        new_image.at<Vec3b>(y,x)[c]
=saturate_cast<uchar>( contrastValue*( image.at<Vec3b>(y,x)[c] ) +
brightnessValue );
                                }
                        }
                }
                return new_image;
        }

        //Relative mouse tracker
        Point mouseMotionDetection(Point referencePoints1,Point
referencePoints2,Point referenceMidPoints, Point currentPoints, double sensitivity
= 8)
        {
                //If the refernce mid point is with the bound then movement is not
required
                if(currentPoints.x > referencePoints1.x && currentPoints.y >
referencePoints1.y && currentPoints.x < referencePoints2.x && currentPoints.y <
referencePoints2.y)
                {
                        return Point(0,0);
                }
                else
                {
                        int dx=0,dy=0;
                        Point disDir = referenceMidPoints - currentPoints;
                        if(disDir.x <0)
                        {
                                dx = currentPoints.x - referencePoints1.x;
                        }
                        else
                        {
                                dx = currentPoints.x -  referencePoints2.x;
                        }

                        if(disDir.y <0)
                        {
                                dy = currentPoints.y - referencePoints1.y;
                        }
                        else
                        {
                                dy = currentPoints.y - referencePoints2.y;
                        }

                        dx= (int)((double)dx/sensitivity);
                        dy= (int)((double)dy/sensitivity);

                        return Point(dx,dy);

                }
        }

        //Comman HSV & GrayScale
```

```
        void convertImageSystem(Mat &imgDestMat,Mat
&imgDestMatGrayScale,Mat &imgDestMatHSV)
        {
                //gray scal convertion......
                cvtColor(imgDestMat,imgDestMatGrayScale,CV_RGB2GRAY);
                //SKIN Detection beta version => bINARY iMAGE
                cvtColor(imgDestMat,imgDestMatHSV,CV_BGR2HSV);


        }


        //Contrast and brightness system
        Mat convertImageContrastBrightness(Mat imgDestMat)
        {
                        double contrastValue=1.0;
                        int brightnessValue=0;

                        //Apply Histogram Equalization

        getHistogramInfo(imgDestMatGrayScale,contrastValue,brightnessValue);

                        //contrast and brightness adjustment for cols
                        imgDestMat =
contrastBrightnessAdjusted(imgDestMat,contrastValue,brightnessValue);

                        return imgDestMat;
        }
        //Get Skin detection with error correction
        Mat getImageSkinDetection(Mat imgDestMatHSV)
        {
                imgDestMatHSV = blackWhiteImageOfSkin(imgDestMatHSV);

                dilate(imgDestMatHSV,imgDestMatHSV,Mat(),Point(-1,-1),5);
                erode(imgDestMatHSV,imgDestMatHSV,Mat(),Point(-1,-1),5);
                dilate(imgDestMatHSV,imgDestMatHSV,Mat(),Point(-1,-1),2);

                imgDestMatHSV = imgDestMatHSV |
shiftMatrixPositive(imgDestMatHSV,5,0) |
shiftMatrixPositive(imgDestMatHSV,0,5) |
shiftMatrixNegative(imgDestMatHSV,5,0) |
shiftMatrixNegative(imgDestMatHSV,0,5);

                return imgDestMatHSV;
        }

        //Get Edge of object with enhance Edge Detection
        Mat getImageEdgeEnhance(Mat imgDestMatGrayScale)
        {
                imgDestMatEdge =  cannyEdgeDetection(imgDestMatGrayScale,9);
                dilate(imgDestMatEdge,imgDestMatEdge,Mat(),Point(-1,-1));
                erode(imgDestMatEdge,imgDestMatEdge,Mat(),Point(-1,-1));

                return imgDestMatEdge;
        }
```

```
	//main get video feed function
	int getVideoFeed(int escapeKey=27)
	{
		mousePosition = new
eventTrigger(screenHeight,screenWidth,cameraHeight,cameraWidth);

		//background subtractor protocal
		BackgroundSubtractorMOG2 bg(30,6,false);

		//face detection so that i can eliminate that better caculation
		Point midPointFace(0,0),lowPointFace(0,0),highPointFace(0,0);
		Point refFacePt1(0,0),refFacePt2(0,0);

		int isFaceFound=0;
		Size detectedFace(0,0),camshiftDetectedFace(0,0);
		bool camShiftPrimaryDetectionFace = false;

		////found variables
		int isFoundFist=0,isFoundPalm=0;
		int isFistFirst=1,isPalmFirst=1;
		int fistNotDetectedCount=0;

		////ideal hand count is to check the ideal nature of the hand
		bool camShiftPrimaryDetectionFist = false;

		//fist find and palm
```

```
                    Point pt1,pt2;//specifies the points of the rectangle
                    Point midPointPalm,midPointHand,midPointFist; // the midpoint
values for the various object detected
                    Point minPointPalm,maxPointPalm; //low and upper bound of fist
detection
                    Point minHandPt1,minHandPt2; //low and upper bound for hand

                    Size detectedFist(0,0),camshiftDetectedFist(0,0);

                    //Reference Pointer for the mouse
                    Point mousePointerReference(0,0);
                    Point referencePalmPt1(0,0);
                    Point referencePalmPt2(0,0);

                    //time controls for events
                    time_t currentTime = time(0);

                    //duration for hand detection and fist detection
                    time_t handTime = time(0);
                    time_t handClickTime = time(0);

                    double handElapseTime = 0;

                    Point recordedHandPoint1,recordedHandPoint2;

                    //mat for mask hadn
                    Mat handMask = Mat::zeros(imgDestMat.size(),CV_8UC1);
```

```
		cout<<"\nBackground detection active....look at the monitor and stay
still";
		initialBackground(bg);
		Mat foregroundMask = Mat::zeros(imgDestMat.size(),CV_8UC1);

		// Show the image captured from the camera in the window and repeat
		while (runCamFeed)
		{
			// Get one frame
			capture->retrieve(imgSrcMat);

			if(isFlipImage)
				flip(imgSrcMat,imgSrcMat,1);

			imgDestMat=imgSrcMat;

			//Get Orginal HSV and Org. GrayScale for the image

	convertImageSystem(imgDestMat,imgDestMatGrayScale,imgDestMatHSV)
;

			//Adjust brightness and contrast based on the histogram
			imgDestMat = convertImageContrastBrightness(imgDestMat);

			//foreground & background extraction
			foregroundMask = currentBackground(bg,imgSrcMat);
```

```
                        imgDestMatGrayScale.copyTo(imgSrcMat,foregroundMask);

                        namedWindow("s");
                        imshow("s",imgSrcMat);

                        //the new edge detected
                        imgDestMatEdge =
getImageEdgeEnhance(imgDestMatGrayScale);

                        ////skin detection ....the function need a hsv based image
                        imgDestMatHSV = getImageSkinDetection(imgDestMatHSV);



                        Mat imgDestMatBoundary;

        imgDestMatEdge.copyTo(imgDestMatBoundary,imgDestMatHSV);

                        //Face detection
                        if(!isFaceFound)
                        {
                                midPointFace =
detectFaceMask(imgDestMatGrayScale,lowPointFace,highPointFace,isFaceFound
);

        rectangle(imgDestMat,lowPointFace,highPointFace,Scalar(0,255,255),1,8,0)
;
```

```
                camShiftPrimaryDetectionFace = false;
            }
            else
            {
                static Point refFacePt1,refFacePt2;
                int detectedHeight,detectedWidth;
                static Point distanceFacePts(0,0);

                try
                {
                    if(camShiftPrimaryDetectionFist==false)
                    {
                        refFacePt1 = lowPointFace;
                        refFacePt2 = highPointFace;
                        camShiftPrimaryDetectionFace=true;

                        detectedHeight = highPointFace.y -
lowPointFace.y;
                        detectedWidth  = highPointFace.x -
lowPointFace.x;
                    }
                    else
                    {
                        detectedHeight = highPointFace.y -
lowPointFace.y;
```

```
                                        detectedWidth = highPointFace.x -
lowPointFace.x;
                                }

                                Rect faceMaskRect(refFacePt1,refFacePt2);

                                TermCriteria theTerm = TermCriteria(
CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10,1);
                                RotatedRect trackBox =
CamShift(imgDestMatBoundary,faceMaskRect,theTerm);

                                midPointFace = trackBox.center;
                                camshiftDetectedFace = trackBox.size;

                                refFacePt1 = Point(midPointFace.x -
detectedWidth/2 ,midPointFace.y - detectedHeight/2);
                                refFacePt2 = Point(refFacePt1.x +
detectedWidth,refFacePt1.y + detectedHeight);



        rectangle(imgDestMat,refFacePt1,refFacePt2,Scalar(23,12,40),1,8,0);//flood
fill the face with black

                                if((((double)detectedFace.area() * 0.90 >
(double)camshiftDetectedFace.area()) && ((double)detectedFace.area() * 1.10 <
(double)camshiftDetectedFace.area())))
                                {
                                        isFoundFist=0;
                                        camShiftPrimaryDetectionFist=false;
                                        cout<<"\nSize inadequate fist";
                                }
                        }
                        catch(...)
                        {
                                isFaceFound=0;
                        }
                }

                midPointPalm =
detectPalmMask(imgDestMat,minPointPalm,maxPointPalm,isFoundPalm);

                //fist finder for mouse tracker>>
                if(isFoundFist==0 && isFoundPalm==0)
                {
                        //Find the midpoint of the fist
                        midPointFist =
detectFistMask(imgDestMatGrayScale,pt1,pt2,isFoundFist);

                        detectedFist.height = pt2.y-pt1.y;
                        detectedFist.width = pt2.x-pt1.x;

                        if(midPointFist.x > lowPointFace.x && midPointFist.y >
lowPointFace.y && midPointFist.x < highPointFace.x && midPointFist.y <
highPointFace.y )
```

```
        {
            isFoundFist=0;
        }

        camShiftPrimaryDetectionFist = false;
    }
    else
    {
        static Point refPt1,refPt2;
        int detectedHeight,detectedWidth;
        static Point distancePts(0,0);

        try
        {
            if(camShiftPrimaryDetectionFist==false)
            {
                refPt1 = pt1;
                refPt2 = pt2;
                camShiftPrimaryDetectionFist=true;

                distancePts = pt2 - pt1;
                distancePts =
Point(distancePts.x/8,distancePts.y/8);

                detectedHeight = pt2.y - pt1.y;
                detectedWidth = pt2.x - pt1.x;
            }
```

```
                                        else
                                        {
                                                detectedHeight = pt2.y - pt1.y;
                                                detectedWidth = pt2.x - pt1.x;
                                        }

                                        Rect palmMaskRect(refPt1,refPt2);
                                        Rect regionOfInterest(pt1 - distancePts,pt2 +
distancePts);

                                        TermCriteria theTerm = TermCriteria(
CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10,1);
                                        RotatedRect trackBox =
CamShift(imgDestMatBoundary,palmMaskRect,theTerm);

                                        midPointFist = trackBox.center;
                                        camshiftDetectedFist = trackBox.size;

                                        refPt1 = Point(midPointFist.x -
detectedWidth/2,midPointFist.y - detectedHeight/2);
                                        refPt2 = Point(refPt1.x + detectedWidth,refPt1.y +
detectedHeight);

                                        if((((double)detectedFist.area() * 0.90 >
(double)camshiftDetectedFist.area()) && ((double)detectedFist.area() * 1.10 <
(double)camshiftDetectedFist.area())))
                                        {
```

```
						isFoundFist=0;
						camShiftPrimaryDetectionFist=false;
						cout<<"\nSize inadequate fist";
					}

					//fist is not in fist area
					if(!(((pt1.x - 
distancePts.x)<=midPointFist.x)&&((pt1.y - distancePts.y)<=midPointFist.y)&&
((pt2.x + distancePts.x)>=midPointFist.x)&& ((pt2.y + 
distancePts.y)>=midPointFist.y)))
					{
						isFoundFist=0;
						camShiftPrimaryDetectionFist=false;
						cout<<"\nDistance inadequate fist";
					}

					//if the tracker gets a value outside the range
					if(midPointFist.x > lowPointFace.x && 
midPointFist.y > lowPointFace.y && midPointFist.x < highPointFace.x && 
midPointFist.y < highPointFace.y )
					{
						isFoundFist=0;
					}



	rectangle(imgDestMat,pt1,pt2,Scalar(255,0,0),1,8,0);
```

```
          rectangle(imgDestMat,refPt1,refPt2,Scalar(0,255,0),1,8,0);
                                        rectangle(imgDestMat,pt1-
distancePts,pt2+distancePts,Scalar(0,0,255),1,8,0);

                              }
                              catch(...)
                              {
                                        isFoundFist=0;
                                        camShiftPrimaryDetectionFist=false;
                              }
                    }

                    //hand and fist for click operation and complex operation
                    midPointHand =
detectHandMask(imgDestMatGrayScale,pt1,pt2,minHandPt1,minHandPt2);

                    //get currentTime
                    currentTime = time(0);

                    //get mask for advance gesture like finger click
                    if((minHandPt1 != recordedHandPoint1) && (minHandPt2 !=
recordedHandPoint2))
                    {
                              recordedHandPoint1 = minHandPt1;
                              recordedHandPoint2 = minHandPt2;
```

```
				handMask = Mat::zeros(imgDestMat.size(),CV_8UC1);

	rectangle(handMask,recordedHandPoint1,recordedHandPoint2,Scalar(255),-
1);

				handTime = time(0);
			}

			if(difftime(handTime,currentTime)==-8)
			{
				handMask = Mat::zeros(imgDestMat.size(),CV_8UC1);

			}

			//calculates the difference in
			if(isFoundFist)
			{
				if(isFistFirst)
				{
					mousePointerReference = midPointFist;
					Point dist = pt2 - pt1;
					referencePalmPt1 = pt1 + (Point(dist.x/4,dist.y/4));
					referencePalmPt2 = pt2 - (Point(dist.x/4,dist.y/4));

					isFistFirst=0;
					cout<<"\nMouse Movement online....";
				}
```

```
                        else
                        {

            rectangle(imgDestMat,pt1,pt2,Scalar(255,255,255),1,8,0);

            rectangle(imgDestMat,referencePalmPt1,referencePalmPt2,Scalar(255,0,0),
1,8,0);

                                fistNotDetectedCount=0;

                                Point referenceDisplacement =
mouseMotionDetection(referencePalmPt1,referencePalmPt2,mousePointerReferen
ce,midPointFist,18);

                                if( referenceDisplacement.x +
referenceDisplacement.y !=0)
                                {
                                        mousePosition-
>setmousePosition(referenceDisplacement.x,referenceDisplacement.y);
                                }

                        }
                        isPalmFirst=1;
                }
                else
                {
                        fistNotDetectedCount++;
```

```
		if(fistNotDetectedCount>=6)
		{
			isFistFirst=1;
		}
	}

	//Palm found for click basic operation
	if(isFoundPalm && isPalmFirst)
	{
		if(isPalmFirst>=1)
		{
			isPalmFirst++;
			isFoundFist=0;

			//reduces the chance of fail positive <<
			if(isPalmFirst==3)
			{
				mousePosition->setMouseLeftClick();
				cout<<"\nMouse Single left click....";
				isPalmFirst=0;


			}
		}
	}

	//FINAL USER WINDOW FOR IDENTIFICATION
```

```
                namedWindow("Main",CV_WINDOW_AUTOSIZE);
                imshow("Main",imgDestMat);



                //imgDestMat.copyTo(imgDestMatEdge,fistMask);
                imgDestMatHSV.copyTo(imgDestMat,handMask);

                imgDestMat = contourAdjustment(imgDestMat);

                Mat imgDestMatCon = convexHullDetection(imgDestMat);

                if (!capture->isOpened())
                {
                        fprintf( stderr, "\nERROR: Can't get a valid frame to
process...\n" );
                        getchar();
                        return 0;
                }
                //remove higher bits using AND operator
                if ((cvWaitKey(10) & 255) == escapeKey)
                {
                        break;
                }
        }

        return 1;
    }
```

```
	//to map the hand to the screen
	void calculateConvertionRatio()
	{
		cout<<"\nRecalculate the convertion ratio";
		capture->retrieve(imgSrcMat);

		cameraHeight= imgSrcMat.size().height;
		cameraWidth = imgSrcMat.size().width;

		yRatio = (double)(screenHeight/cameraHeight);
		xRatio = (double)(screenWidth/cameraWidth);

		cout<<"\nScreen Height:"<<screenHeight<<"\nScreen
Width"<<screenWidth<<"\nCamera Height:"<<cameraHeight<<"\nCamera
Width:"<<cameraWidth<<"\nHeight Factor:"<<yRatio<<"\nWidth
Factor:"<<xRatio;
	}

	//device capture mod
	bool captureVideoFeed()
	{
		capture = new VideoCapture(CV_CAP_ANY);

		if (!capture->isOpened() )
		{
```

```
                    fprintf( stderr, "\nERROR: Can't Get a camera video feed.
Please Check the camera and the required software \n" );
                    getchar();
                    runCamFeed = false;
                    return false;
                }

                calculateConvertionRatio();

                runCamFeed=true;
                return true;
        }

public :

        cameraSoftware()
        {
                isFlipImage=true;

                calculateScreenResolution();
        }

        bool camerasSetting(bool isFlip)
        {
                isFlipImage=isFlip;
                calculateScreenResolution();
        }
```

```
	/*Capture camera feedback from any available camera*/
	bool captureVideoFeedThread()
	{
		return captureVideoFeed();
	}

	/* the producer for the images*/
	void getVideoFeedThread()
	{
		cout<<"\nVideo Feed Thread Online...";
		getVideoFeed();
	}

	/*Remove all captured hardware resources*/
	void destroyTakenResource()
	{
		cout<<"\nThis is Galactus, Destroyer of World!!!...\nTerminating the
captured hardware resources.";
		// Release the capture device housekeeping
		delete(capture);

		return ;
	}

};
```

```
void handGestureSystem()
{

        //windowHideConsole();

        /* Calling GetDC with argument 0 retrieves the desktop's DC */
   HDC hDC_Desktop = GetDC(0);

        cameraSoftware myCamera;

        myCamera.captureVideoFeedThread();

        boost::thread
myProducer(&cameraSoftware::getVideoFeedThread,&myCamera);

        myProducer.join();

        //kills the complete object
        myCamera.destroyTakenResource();
        myProducer.detach();

        return;
}

int _tmain(int argc, _TCHAR* argv[])
{
        handGestureSystem();
    return 0;
}
```

# 9. BIBLOGRAPHY

[1] Joo-Ho Lee, Kazuyuki Morioka, Noriaki Ando, Hideki Hashimoto (2004) ‘Cooperation of Distributed Intelligent Sensors in Intelligent Environment’, Proceedings of IEEE/ASME Transactions on Mechatronics, Vol. 9, No. 3, pp 535 – 543.

[2] Nguyen Dang Binh, Enokida Shuichi, Toshiaki Ejima (2005) ‘Real-Time Hand Tracking and Gesture Recognition System’, Proceedings of GVIP Conference held at Cairo, Egypt.

[3] Nguyen Dang Binh, Toshiaki Ejima (2005) ‘Hand Gesture Recognition Using Fuzzy Neural Network’, Proceedings of GVIP Conference held at Cairo, Egypt.

[4] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, Steven Shafer (2000) ‘EasyLiving: Technologies for Intelligent Environments’, Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing at Bristol, pp. 12-29.

[5] Beifang Yi, Frederick C. Harris Jr., Ling Wang, Yusong Yan (2005) ‘Real-Time Natural Hand Gestures’, Proceedings of IEEE CS and the AIP, pp. 92-97.

[6] Kenji Oka, Yoichi Sato, Hideki Koike (2002) ‘Real-Time Fingertip Tracking and Gesture Recognition’, Proceeding of IEEE Computer Graphics and Applications, pp. 64-71.

[7] Cristina Manresa, Javier Varona, Ramon Mas, Francisco J. Perales (2005) ‘Hand Tracking and Gesture Recognition for Human-Computer Interaction’, Journal ofElectronic Letters on Computer Vision and Image Analysis, pp. 96-104.

[8] Hasup Lee, Yoshisuke Tateyama, Tetsuro Ogi (2012) ‘Hand Gesture Recognition using blob Detection for immersive projection display system’, International Journal of Computer and Communication Engineering, pp. 115- 118.

[9] Shahzad Malik (2003) ‘Real-time Hand Tracking and Finger Tracking for Interaction’, project report submitted at Toronto university.

[10] Annamária R. Várkonyi-Kóczy (2008) ‘Intelligent Autonomous Primary 3D Feature Extraction in Vehicle System Dynamics Analysis: Theory and Application’, Proceeding at Acta Polytechnica Hungarica, Vol. 5, No. 1, pp. 5 – 29.

[11] Gary R. Bradski, Santa Clara (1998) ‘Computer Vision Face Tracking For Use in a Perceptual User Interface’, Published in Journal of Intel Technology, pp. 1-15.

[12] Annamária R. Várkonyi-Kóczy, AndrásRövid, GracaRuano (2003) ‘Fuzzy Logic Supported Point Correspondence Matching for 3D Reconstruction’, sponsored by Hungarian Fund for Scientific Research.

[13] Mohamed Alsheakhali, Ahmed Skaik, Mohammed Aldahdouh, Mahmoud Alhelou (2011) ‘Hand Gesture Recognition System’, submitted at The Islamic University of Gaza.

[14] Rafiqul Zaman Khan and Noor Adnan Ibraheem (2012) ‘HAND GESTURE RECOGNITION: A LITERATURE REVIEW’, Proceeding of International Journal of Artificial Intelligence & Applications (IJAIA), Vol.3, No.4, pp. 161-174.

[15] William T. Freeman, Michal Roth (1994) ‘Orientation Histograms for Hand Gesture Recognition’, Proceeding of IEEE International workshop on Automatic Face and Gesture Recognition.

[16] Pavithra Kannan, Rohan Pradip Shah, Thulasi Srinivasan, Uzma Parveen S (2009) ‘Hand Gesture Recognition Using Computer Vision’, submitted at Visvesaraya Technological University.

[17] Lalit Gupta and Suwei Ma (2001) ‘Gesture-Based Interaction and Communication: Automated Classification of Hand Gesture Contours’, Proceeding of IEEE Transactions on systems, man, and cybernatics—part C: Applications and reviews, Vol. 31, No. 1, pp. 114-120.

[18] Carlo Colombo, Alberto Del Bimbo, and Alessandro Valli (2003) ‘Visual Capture and Understanding of Hand Pointing Actions in a 3-D Environment’, Proceeding of IEEE Transactions on systems, man, and cybernatics —part B, cybernatics, Vol. 33, No. 4, pp 677 – 686.

[19] Ankit Chaudhary, J. L. Raheja, Karen Das, Sonia Raheja (2011) ‘Intelligent Approaches to interact with Machines using Hand Gesture Recognition in Natural way: A Survey’, Published at International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1, pp. 122-133.

[20] Daniel Popa, Georgiana Simion, VasileGui, Marius Otesteanu (2008) ‘Real Time Trajectory Based Hand Gesture Recognition’, Proceeding at WSEAS Transactions on Information Science & Applications, Issue 4, Volume 5, pp. 532-546.

[21] Mark Bayazit, Alex Couture-Beil, Greg Mori (2005) ‘Real-time Motion-based Gesture Recognition using the GPU’ , Published at Simon Fraser University Burnaby, BC, Canada.

[22] Ming-Hsuan Yang, NarendraAhuja, Mark Tabb (2002) ‘Extraction of 2D Motion Trajectories and Its Application to Hand Gesture Recognition’, Proceeding of IEEE Transactions on pattern analysis and machine intelligence, Vol. 24, No. 8, pp 1061-1074.

[23] Juan P. Wachs, Helman Stern, Yael Edan (2005) ‘Cluster Labeling and Parameter Estimation for the Automated Setup of a Hand-Gesture Recognition System’, Proceeding of IEEE Transactions on systems, man, and cybernatics - Part A: Systems and Humans, Vol. 35, No. 6, pp. 932- 944.

[24] Mu-Chun Su (2000) ‘A Fuzzy Rule-Based Approach to Spatio-Temporal Hand Gesture Recognition’, Proceeding of IEEE Transactions on systems, man, and cybernatics—part C: Applications and reviews, Vol. 30, No. 2, pp. 276- 281.

[25] Etsuko Ueda, Yoshio Matsumoto, Masakazu Imai, Tsukasa Ogasawara *(2003)* ‘A Hand-Pose Estimation for Vision-Based Human Interfaces’, Proceeding at IEEE Transactions on Industrial Electronics, Vol. 50, No. 4, pp. 676-684.