

## Learning Resource Center

BOOK NUM.:

ACCESSION NO.: SP08036/SP0812055

This book was issued is overdue due on the date stamped below. If the book is kept over due, a fine will be charged as per the library rules.

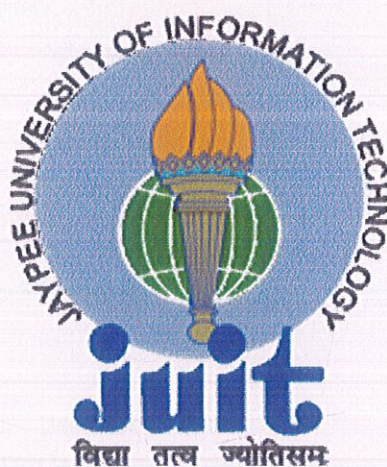
[illegible]



# GESTURE RECOGNITION SYSTEM

Name of the Students - Sandarsh Srivastava (081231)  
Mayank Goyal (081232)  
Pulkita Jain (081233)

Name of the supervisor - Mr. Suman Saha



May – 2012



**Submitted in partial fulfillment of the Degree of  
Bachelor of Technology**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING  
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,  
WAKNAGHAT**



## TABLE OF CONTENTS

Chapter No.	Topics	Page No.
	Certificate from the Supervisor	II
	Acknowledgement	III
	Summary	IV
	List of Symbols and acronyms	V
<b>1.</b>	<b>Introduction</b>	<b>1-12</b>
	1.1. Project Scope and Objectives	
	1.2. Project Summary	
	1.3. Literature Review	
	1.4. Existing Systems	
<b>2.</b>	<b>System Requirements</b>	<b>13-19</b>
	2.1. Required system behaviour	
	2.2. System Functionalities	
	2.3. System interfaces, inputs, and outputs	
	2.4. Concept of Moments	
	2.5. System models:	
	2.5.1. Data-Flow Model	
	2.5.2. Behavioural Model – Use Case Diagram	
	2.6. Failure modes and action on failure	
<b>3.</b>	<b>Task Analysis and Schedule of Activities</b>	<b>20-24</b>
	3.1. Task decomposition	
	3.2. Incremental Model	
	3.3. Project schedule	
<b>4.</b>	<b>Detection</b>	<b>25-28</b>
	4.1. Choice Of Camera	
	4.2. Hardware Setup	
	4.3. Method Of Colour Detection	
<b>5.</b>	<b>Recognition and Refinement</b>	<b>29-32</b>
	5.1. Recognition Strategy	
	5.2. Selection Of Gesture Set	
	5.3. Analysis of distortion and noise	
	5.4. Removal of background objects	
	5.5. Removal of unwanted colours	
	5.6. Overview of Algorithms	
<b>6.</b>	<b>Conclusion</b>	<b>33-34</b>
	6.1. Project Goals	
	6.2. Further Work	




<b>7. References</b>	<b>35</b>
<b>8. Annexure</b>	<b>36-53</b>
8.1. Codes and Snapshots	
<b>9. Appendix</b>	<b>54-55</b>
9.1. Appendix A- Glossary	
9.2. Appendix B - Entire Gesture Set	



CERTIFICATE

This is to certify that the work titled "**Gesture Recognition System**" submitted by "**Pulkita Jain, Mayank Goyal, Sandarsh Srivastava**" in partial fulfillment for the award of degree of B. Tech of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor:



Name of Supervisor:

Mr. Suman Saha.

Designation:

Senior Lecturer.

Date:

17<sup>th</sup> May 2012.



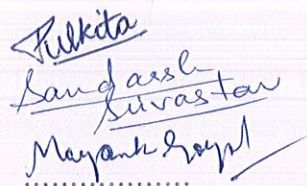
**ACKNOWLEDGEMENT**

At the outset, it is our duty to acknowledge with gratitude the generous help that we received from our supervisor **Mr. Suman Saha** (Senior Lecturer, Computer Science and Engineering, JUIT) without whom the implementation of this project would not have been a success.

Signature of the students:

Name of Student:

Date:



.....

Pulkita Jain.

Sandarsh Srivastava.

Mayank Goyal.

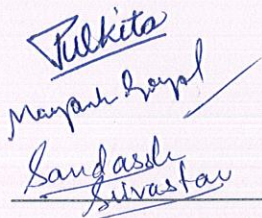
17<sup>th</sup> May 2012.



### SUMMARY

Considerable effort has been put towards developing intelligent and natural interfaces between users and computer systems. This is done by means of a variety of modes of information (visual, audio, pen, etc.) either used individually or in combination. The use of gestures as means to convey information is an important part of human communication. The automatic recognition of gestures enriches Human-Computer Interaction by offering a natural and intuitive method of data input.

This project presents a new technique for human gesture recognition, for the Human-Computer Interaction (HCI) that can be easily is actually used in a Home Personal Computer with basic hardware and Windows. The objective of this effort was to provide a Human Gesture Recognition software that can be easily used to interact with computer in a fast, simple and, importantly, in a natural way to perform various operations like opening a file, navigating to a website, moving mouse cursor and performing left and right click, increase/decrease volume, shutdown/hibernate/log-off/sleep a PC, changing powerpoint slides, zooming in and out of images, etc. The overall model is designed to be a simple and robust gestural interface prototype for various PC applications.



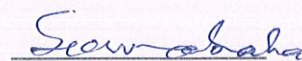
Signature of the Students

Name: Pulkita Jain

Mayank Goyal

Sandarsh Srivastava

Date: 17<sup>th</sup> May 2012



Signature of Supervisor

Name: Mr. Suman Saha

Date: 17<sup>th</sup> May 2012



**LIST OF FIGURES**

- [1].       Figure 1: Existing systems of gesture recognition
- [2].       Figure 2: Data flow diagram - Part 1
- [3].       Figure 3: Data flow diagram - Part 2
- [4].       Figure 4: Use Case diagram
- [5].       Figure 5: Incremental Model
- [6].       Figure 6: Gantt chart
- [7].       Figure 7: Process of Blob Detection
- [8].       Figure 8: Color filtering method
- [9].       Figure 9: Channel filtering method
- [10].      Figure 10: HSL filtering method
- [11].      Figure 11: Euclidean filtering method
- [12].      Table 1: Basic controls gestures and associated operations
- [13].      Table 2: Basic controls gestures and associated operations
- [14].      Table 3: Basic controls gestures and associated operations



# **Chapter 1: Introduction**

Gesture Recognition is a topic in the field of computer science which deals with the goal of interpreting human gestures via some mathematical algorithms. Gestures can originate from any bodily motion or the state but do commonly originate from the face or hand of a person. The current focuses in the field include emotion recognition from the face and the hand gesture recognition. The identification and recognition of some posture, gait and human behaviour is also the subject of the gesture recognition techniques.

Gesture recognition can be seen as a way for the computer to begin to understand the human body language, thus helping to build a richer bridge between the machines and humans than the primitive text user interfaces or even GUIs (Graphical User Interfaces), which still limit the majority of input to keyboard and mouse.

Gesture recognition also enables humans to interface with the machine (HMI) and interact naturally without any mechanical devices. Using the concept of gesture recognition, it is also possible to point a finger at the computer screen so that the cursor will move accordingly. This could potentially make the various conventional input devices such as mouse, keyboards and even touch-screens redundant. Gesture recognition can be conducted using techniques from computer vision and image processing.

Some of the basic operations of the computer include closing or opening a file, making a presentation, cursor movements, playing a game, playing a song in a media player, controlling the volume, shut down or restart a computer ,or watching a movie. Human gesture recognition supports all this with the use of our hand and facial gestures, thus bounding the use of mouse keyboard and the other hardware devices and making it more user friendly and unique.

Gestures and gesture recognition are the terms which are increasingly encountered in the discussions of human-computer interaction. For many (if not most) people this term includes character recognition, the recognition of proof readers symbols, shorthand, Marking Interfaces. In fact mostly every physical action involves a gesture of some sort in order to be articulated. Furthermore, the nature of that gesture is generally an important component in establishing the quality of feel to the action. Nevertheless, what we want to isolate for discussion in this chapter are interactions where the gesture is what is articulated and recognized, rather than a consequence of expressing something through a transducer.



There are various types of gestures that can be identified by computers and thus can be used to perform computer operations.

**Sign language recognition:** Just as we know speech recognition can transcribe speech to text, certain types of gesture recognition software can also transcribe the symbols represented through the sign language into text.

**Directional indication through pointing:** Pointing has a very specific purpose in our society, to reference to an object or location based on its position relative to ourselves. The use of gesture recognition to determine where a person is pointing is useful for identifying the context of statements or instructions.

**Control through facial gestures:** Controlling a computer through the facial gestures is a useful application of gesture recognition for all those users who may not physically be able to use a mouse or keyboard. Eye tracking in particular may be of use for controlling the cursor motion or focusing on elements of a display.

**Alternative computer interfaces:** Foregoing the traditional keyboard and mouse setup to interact with a computer, strong gesture recognition could allow the users to accomplish some frequent or common tasks using hand or face gestures to a camera.

**Immersive game technology:** Gestures can also be used to control interactions within video games to try and make the game player's experience more interactive or immersive.

**Virtual controllers:** For systems where the act of finding or acquiring a physical controller could require too much time, gestures can be used as an alternative control mechanism. Controlling the secondary devices in a car, or controlling a television set are examples of such usage.

### **Tracking Technologies**

Gesture-only interfaces with the syntax of many gestures typically require precise hand pose tracking. A common technique is to instrument the hand with a glove which is equipped with a number of sensors that provide information about the hand position, orientation, and flex of the fingers. The first commercially available hand tracker, the Dataglove, is described in Zimmerman, Lanier, Blanchard, Bryson and Harvill (1987), and is illustrated in the video by Zacharey, G. (1987). This uses thin fibre optic cables running down the back of each hand, each with a small crack in it. Light is shone down the cable so when the fingers are bent, the light leaks out through the cracks. Measuring light loss gives us an accurate reading of hand pose. The Dataglove could measure each joint bend to an accuracy of 5 to 10 degrees (Wise et. al. 1990), but not the sideways movement of the fingers (finger abduction). However, the CyberGlove developed by Kramer



(Kramer 89) uses strain gauges placed between the fingers to measure abduction as well as for more accurate bend sensing (Figure 1). Since the development of the Dataglove and CyberGlove many other glove based input devices have appeared as described by Sturman and Zeltzer (1994).

The keyboard and mouse are currently the main interfaces between a man and a computer. In other areas where the 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, 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 can not only communicate from a distance, but also no need to have physical contact with the computer. However, unlike audio commands, a visual system would be preferable in noisy environments or in situations where the sound would cause a disturbance. The visual system chosen was the recognition of hand gestures. The amount of computation required to process the hand gestures is much greater than that of the mechanical devices, however the standard desktop computers are now quick enough to make this project — hand gesture recognition using computer vision — a viable proposition. 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.

**3D animation:** Rapid and simple conversion of the hand movements into 3D computer space for the purposes of computer animation.

**Visualization:** Just as the objects can be visually examined by rotating them with the hand, so it would be advantageous if the 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 of the applications.

**Control of mechanical systems (such as robotics):** Using the hand to remotely control a manipulator.

## 1.1 Project Scope and Objectives

**Scope:** The project is a desktop application for systems running Windows Operating System using OpenCV libraries.



**Objectives:** To provide a platform where humans can interact with computers in a way as they interact with each other; which is also more technologically advanced, easy to use, fast and more interactive. Computer is made to recognize and interpret various gestures made by humans and perform an operation accordingly.

## **1.2 Project Summary**

The Gesture Recognition System project sees its user using the integrated or dedicated webcam and performs operations according to gestures by its user. It is a desktop application for Microsoft Windows operating systems. The gestures are made using an object of a specific color. The application tracks the colored objects and according to their color and their motion, a gesture is recognized. The following parameters are used for detecting a gesture and performing an operation:

### **Color:**

The color of the object to be detected has already been specified in the application. But later with an additional functionality, this can be done by selecting the color from color dialog box or by color-picking in the image itself.

### **Color Range:**

A value for color variation must be specified between 0 to 255. This allows the amount of deviation from the color selected. Whenever we select a color then as the objects move in front of the camera its color may slightly change in every frame due in change of light conditions in room, the contrast, the light falling on the object, a shine or reflection from the object surface, etc. Thus by specifying the range of variation we can keep on detecting the object even after a slight color variation. This is also the way the projects maintains its robustness, acting perfectly in all conditions. Higher color range will allow more deviation while may also cause errors by detecting any unwanted objects. Lower range allows less deviation but may not always detect the object. Hence values must be set by moving the object here and there in front of the camera and then choosing the best option.

### **Minimum Size:**

The minimum size of the object to be detected must be set. This avoids the detection of small particles, objects, etc in the background of the same color as our specified color. This reduces errors and noise.



## 1.3 Literature Review

### Image Processing

The Digital image processing is the use of computer algorithms to perform image processing on digital images. As a subcategory of the field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and thus can avoid the problems such as the build-up of noise and signal distortion during processing. Since images are defined over two dimensions (perhaps more) digital image processing may be modelled in the form of multidimensional systems. In electrical engineering and computer science, image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it.

Image processing usually refers to the digital image processing, but optical and analog image processing are also possible. This article is about the general techniques that apply to all of them. The acquisition of images (producing the input image in the first place) is referred to as imaging. Typical operations:

- Euclidean geometry transformations such as enlargement, reduction, and rotation
- Color corrections such as brightness and contrast adjustments, color mapping, color balancing, quantization, or color translation to a different color space
- Digital compositing or optical compositing (combination of two or more images), which is used in film-making to make a "matte"
- Interpolation, demosaicing, and recovery of a full image from a raw image format using a Bayer filter pattern
- Image registration, the alignment of two or more images
- Image differencing and morphing
- Image recognition, for example, may extract the text from the image using optical character recognition or checkbox and bubble values using optical mark recognition
- Image segmentation
- High dynamic range imaging by combining multiple images
- Geometric hashing for 2-D object recognition with affine invariance



## **Computer Vision**

Computer vision is the field concerned with the automated computer based processing of images to extract and interpret some information. It is the science and the technology of machines that see. Here see means the machine is able to extract the information from an image, to solve some task, or perhaps "understand" the scene in either a broad or limited sense.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract useful information from the images. The image data can take various different forms, such as the video sequences, views from the multiple cameras, or multi-dimensional data from a medical scanner.

As a technological discipline, computer vision also seeks to apply its theories and models to the construction of many computer vision systems. Examples of the various other applications of computer vision include systems for:

- Controlling the processes (e.g., an industrial robot).
- Navigation purposes (e.g. by an autonomous vehicle or mobile robot.)
- Detecting the events (e.g., for visual surveillance or people counting).
- Organizing the information (e.g., for indexing databases of images and image sequences).
- Modelling the objects or environments (e.g., medical image analysis or topographical modelling).
- Interaction (e.g., as the input to a device for computer-human interaction).

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

In most of the practical computer vision applications, the computers are generally pre-programmed for solving a particular task, but methods based on learning are now becoming increasingly common.

## **Typical tasks of computer vision**

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



## Recognition

The classical problem in computer vision, image processing, and the machine vision is that of determining whether or not the image data do contain some specific object, feature, or activity. This task can normally be solved robustly and without much effort by a human, but is still not satisfactorily solved in the field of computer vision for the general case: arbitrary objects in the arbitrary situations. The methods existing for dealing with this type of problem can at best find a solution only for the specific objects, such as some of the simple geometric objects (e.g., polyhedra), human faces, printed or hand-written characters, or the vehicles, and in some of the specific situations, which are typically described in terms of the well-defined illumination, background, and pose of the object relative to the camera.

Different varieties of the recognition problem are described in the literature:

**Object recognition:** one or the several pre-specified or the learned objects or the object classes can be recognized, usually together in the image with their 2D positions or 3D poses in the scene. Google Goggles does provide a stand-alone program illustration of this function.

**Identification:** An individual instance of an object can also be recognized. Examples: identification of a particular person's face or fingerprint, or identification of a specifically mentioned vehicle.

**Detection:** the image data is scanned for some specific conditions. Examples: detection of possible abnormal cells or tissues in the medical images or the detection of a vehicle in an automatic road toll system. Detection which is based on relatively simple and fast computations is sometimes used for finding the smaller regions of interesting image data that can be further analysed by some more techniques, which are computationally demanding, to produce a correct interpretation.

Several specialized tasks based on recognition exist, such as:

**Pose estimation:** estimating the orientation or position of a specific object relative to the camera. An example or the application for this technique would be assisting the arm of a robot in retrieving the objects from a conveyor belt in an assembly line situation or picking some parts from a bin.

**Optical character recognition (OCR):** identifying characters in the images of some printed or handwritten text, usually done with a view to encode the text in such a format which is more amenable to editing or indexing (e.g. ASCII). 2D Code Reading of the 2D codes such as the data matrix and the QR codes.



### **Motion analysis**

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

**Egomotion:** determining the rigid 3D motion (translation and rotation) of the camera from an image sequence which is produced by the camera.

**Tracking:** following the movements of usually a smaller set of the interest points or objects (e.g., vehicles or humans) in the image sequence.

**Optical flow:** to determine, for each point in the image, that how that point is moving relative to the image plane, i.e. determining its apparent motion. This motion is a result of both how the corresponding 3D point is moving in the scene and how the camera is moving relative to the scene.

### **Computer vision system methods**

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

**Image acquisition:** A digital image is produced by one or the several image sensors, which, besides various types of the light-sensitive cameras, also include the range sensors, tomography devices, radar, ultra-sonic cameras, etc. Depending on the type of sensor, the resulting image data is either an ordinary 2D image, a 3D volume, or an image sequence. The pixel values typically correspond to the light intensity in one or the several spectral bands (gray images or colour images), but can also be related to the various physical measures, such as depth, absorption or reflectance of sonic or electromagnetic waves, or nuclear magnetic resonance.



**Pre-processing:** Before any of the computer vision method can be applied to the image data in order to extract some specific piece of information, it is usually necessary to process the data in order to assure that it satisfies the certain assumptions implied by the method. Examples are

- Re-sampling in order to assure that the image coordinate system is correct.
- Noise reduction in order to assure that the sensor noise does not introduce false information.
- Contrast enhancement to assure that the relevant information can be detected.
- Scale-space representation to enhance the image structures at locally appropriate scales.
- Feature extraction: Image features at the various levels of complexity are extracted from the image data. Typical examples of such features are:
  - Lines, edges and ridges.
  - Localized interest points such as corners, blobs or points.
  - More complex features may be related to texture, shape or motion.

**Detection/segmentation:** At some point in the processing a decision is made regarding which image points or regions of the image are relevant for the further processing. Examples are

- Selection of some specific set of interest points
- Segmentation of one or multiple image regions which contain a specific object of interest.
- High-level processing: At this step the input is typically a small set of data, for example a set of points or an image region which is assumed to contain a specific object.
- Verification that the data satisfy model-based and application specific assumptions.
- Estimation of application specific parameters, such as object pose or object size.
- Image recognition: classifying a detected object into different categories.
- Image registration: comparing and combining two different views of the same object.
- Decision making the final decision required for the application,[3] for example:
  - Pass/fail on automatic inspection applications
  - Match / no-match in recognition applications
  - Flag for further human review in medical, military, security and recognition applications

### **Applications for computer vision**

One of the most prominent application fields in the field of computer vision is the medical computer vision or the medical image processing. This area is characterized by the extraction of information from the image data for the purpose of making the medical diagnosis of a patient. Generally, the image data is in the form of microscopy images, X-ray images, angiography images, ultrasonic



images, and tomography images. An example of information that can be extracted from such image data is the detection of tumours, arteriosclerosis or other malign changes. It can also be the measurements of organ dimensions, blood flow, etc. This application area also supports the medical research by providing new information, e.g., about the structure of the brain, or about the quality of medical treatments etc.

A second application area in the field of computer vision is in industry, sometimes also known as machine vision, where the information is extracted for the purpose of supporting a manufacturing process. One example is quality control where the details of the final products are being automatically inspected in order to find the defects. Machine vision is also used heavily in the agricultural process to remove the undesirable food stuff from bulk material, a process called optical sorting.

Military applications are probably one of the largest areas for computer vision. The obvious examples are the detection of enemy soldiers or the vehicles and the missile guidance. Modern military concepts, such as "battlefield awareness", imply that the various sensors, including image sensors, provide a rich set of information about a combat scene that can be used to support the strategic decisions. In this case, automatic processing of the data is used to reduce the complexity and to fuse the information from multiple sensors to increase reliability.

One of the newer application areas is the autonomous vehicles, which include submersibles, land-based vehicles (small robots with wheels, cars or trucks), aerial vehicles, and unmanned aerial vehicles (UAV). The level of autonomy ranges from fully autonomous (unmanned) vehicles to the vehicles where the computer vision based system can support a driver or a pilot in various situations. Fully autonomous vehicles typically use computer vision for navigation purpose, i.e. for knowing where it is, or for producing a map of its environment (SLAM) and for detecting the obstacles. There are ample examples of military autonomous vehicles ranging from the advanced missiles, to UAVs for recon missions or missile guidance. Space exploration is already being made with autonomous vehicles using computer vision, e. g., NASA's Mars Exploration Rover and ESA's ExoMars Rover.

### **Design features**

**Interoperability:** Because mostly the computer systems commonly require interaction between the new and older applications, the .NET Framework provides the means to access functionality which is implemented in the programs that execute outside the .NET environment. Access to COM



components is provided in the System.Runtime.InteropServices and the System.EnterpriseServices namespaces of the framework; access to any other functionality is provided using the P/Invoke feature.

**Common Language Runtime Engine:** The Common Language Runtime (CLR) is the execution engine of the .NET Framework. All the .NET programs execute under the supervision of the CLR, guaranteeing certain properties and behaviours in the areas of memory management, security, and exception handling.

**Language Independence:** The .NET Framework introduces a Common Type System, or CTS. The CTS specification defines all the possible programming constructs and data types supported by the CLR and how they may or may not interact with each other conforming to the Common Language Infrastructure (CLI) specification. Because of this feature, the .NET Framework also supports the exchange of types and object instances between the libraries and applications written using any conforming .NET language.

**Base Class Library:** The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions, including file reading and writing, graphic rendering, database interaction, XML document manipulation and so on.

**Simplified Deployment:** The .NET Framework also includes some design features and tools that help to manage the installation of a computer software to ensure that it does not interfere with previously installed software, and that it conforms to security requirements.

**Portability:** While Microsoft has never implemented the full framework on any system except Microsoft Windows, the framework is engineered to be platform agnostic and some cross-platform implementations are available for other operating systems (see Silverlight and the Alternative implementations section below). Microsoft submitted the specifications for the Common Language Infrastructure (which includes the core class libraries, Common Type System, and the Common Intermediate Language), the C# language and the C++/CLI language to both ECMA and the ISO, making them available as open standards. This makes it possible for third parties to create compatible implementations of the framework and its languages on other platforms.

### **Common Language Infrastructure (CLI-Amer)**



The purpose of the Common Language Infrastructure (CLI) is to provide a language-neutral platform for the application development and execution, including the functions for Exception handling, Garbage Collection, security, and interoperability. By implementing the core aspects of the .NET Framework within the scope of the CLI, this functionality will not be tied not only to a single language but will be available across the many languages supported by the framework. Microsoft's implementation of the CLI is called the Common Language Runtime, or CLR.

#### 1.4 Existing Systems

A simplification used in this project, which was not found in any of the recognition methods researched, is the use of a wrist band to remove the several degrees of freedom. This enabled the three new recognition methods to be devised. The recognition frame rate achieved is comparable to most of the systems in existence (after allowance for processor speed) but the number of different gestures recognised and the recognition accuracy are amongst the best found. Figure below shows several of the existing gesture recognition systems along with recognition statistics and method.

Paper	Primary method of recognition	Number of gestures recognised	Background to gesture images	Additional markers required (such as wrist band)	Number of training images	Accuracy	Frame rate
[Bauer & Hienz, 2000]	Hidden Markov Models	97	General	Multi-coloured gloves	7-hours signing	91.7%	-
[Starnes, Weaver & Pentland, 1998]	Hidden Markov Models	40	General	No	400 training sentences	97.6%	10
[Bowden & Sarhadi, 2000]	Linear approximation to non-linear point distribution models	26	Blue screen	No	7441 images	-	-
[Davis & Shah, 1994]	Finite state machine / model matching	7	Static	Markers on glove	10 sequences of 200 frames each	98%	10

Figure 2: Existing systems of gesture recognition



# **Chapter 2 : System Requirements and Specifications**

## **2.1 Required system behaviour**

The following are the features and the facilities which a user expects of the system.

The software must be efficient enough to recognize the gestures correctly. Such softwares are often not perfect and are good enough only for the demonstration purposes. This system is expected to be actually useful for a Home PC. Thus perfection is the greatest requirement.

There must not be any constraints on the background color. Usually whatever color we specify for an object also appears somewhere in the background too in most of the cases. Thus there is always a restriction of using a plain single colored background and not a multicolored background and the use of an object of such a color so that it can easily be seen against the background. Such a restriction must not be present because a user cannot always appear in front of a plain background just to make a gesture.

It must be robust enough to cope with the illumination conditions and the contrast changes. Whenever we select a color then as the object moves in front of the camera its color may change slightly in every frame due to the change of light conditions in the room, the contrast, the light falling on the object, a shine or reflection from the object surface, etc. This will cause some problem in the color detection and hence the object would not be successfully tracked, thereby spoiling the performance.

Algorithms must be speedy enough to process the frames with high speed so that the software doesn't lag behind. When we take lower frame rate from the camera to process the gesture, the application doesn't look to be of real time. But with the higher frame rate the computer has to process more number of frames in a given time. This slows down the performance of the system. In such a case whenever we make a gesture it takes a significant amount of time to appear in the video and then an operation is performed.

## **2.2 System Functionalities**



All the functionalities of the project can be categorized into 3 types:

- Basic operations
- Mouse Control
- Keyboard Functionalities

We are providing a separate area for switching between these operations.

#### **BASIC OPERATIONS:**

This category includes various operations like opening Web Browser, Microsoft PowerPoint, Calculator, Notepad, Increasing/Decreasing Volume, Shutting/Hibernating/Sleeping/Logging-Off the computer, etc.

Certain gesture is allocated to each action so as to perform that action. When that gesture is appeared in front of the camera, it detects that and performs that certain action which is allocated to that gesture in the program.

#### **MOUSE CONTROL:**

These include the motion and the clicks of the mouse; moving the mouse cursor, performing left and right clicks.

It is a very typical process because it requires a lot of precision and also continuous scanning is required for performing it. When any selected object is moved in front of the camera then the mouse pointer moves accordingly and the left click and right click is also done by performing a certain gesture with that object.

#### **KEYBOARD FUNCTIONALITIES:**

In this we are controlling the arrow keys of keyboard by a certain gesture like moving our hand in a particular direction so as to press the arrow key of that direction. We are then sending the key presses to a selected running application. Hence we can change the PowerPoint presentation slides, zoom in and out of the images, play games, navigate with gestures instead of arrow keys.

### **2.3 System interfaces, inputs, and outputs**

**Gestures and corresponding action performed:**



The system can interpret a total of 13 gestures.

**For Keyboard:**

Maximum no. of objects can be 1.

Total no of directions = 4

So total gestures =  $1 * 4 = 4$

**For Mouse:**

Maximum no. of objects can be 3.

Total no of directions = 4

Right Click = 1

Left Click = 1

So total gestures =  $1 * 4 + 2 = 6$

**For Basic Gestures:**

Total gestures = No. of objects scanned.

Maximum no. of scanned objects can be 3.

So total gestures = 3.

In case of the basic gestures we can set the action performed by a particular gesture by an option provided in software.

Moreover we can also give our own process name, URL, folder path or any such things to make a function of your own.

## **2.4 Concept of moments**

A lot of useful information about a binary object can be gained from the moments of the object.

- Area of the object (Zeroth order moments)
- Centre of mass (First order moments)
- Its Orientation in the image (Second order moments)

**Zeroth order moment ( $M_{0,0}$ ):**



The Zeroth order moment is the area:  $A = \sum \sum I(x,y)$

This is analogous to calculating areas of continuous surfaces using double integral.

#### **First order moment:**

The center of mass can be found by the first order moments ( $M_{0,1}$  and  $M_{1,0}$ ).

$$M_{1,0} = \sum \sum X[I(x,y)]$$

$$M_{0,1} = \sum \sum Y[I(x,y)]$$

The coordinates of the center of mass are calculated as:

$$X_{COM} = M_{1,0}/M_{0,0}$$

$$Y_{COM} = M_{0,1}/M_{0,0}$$

Now with the coordinates of the center of mass, we can have a reference point to track.

## **2.5 System models:**

### **Data-Flow Model**

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. Often DFDs 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).

A DFD shows what kinds of data will be input to and output from the system, and where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).



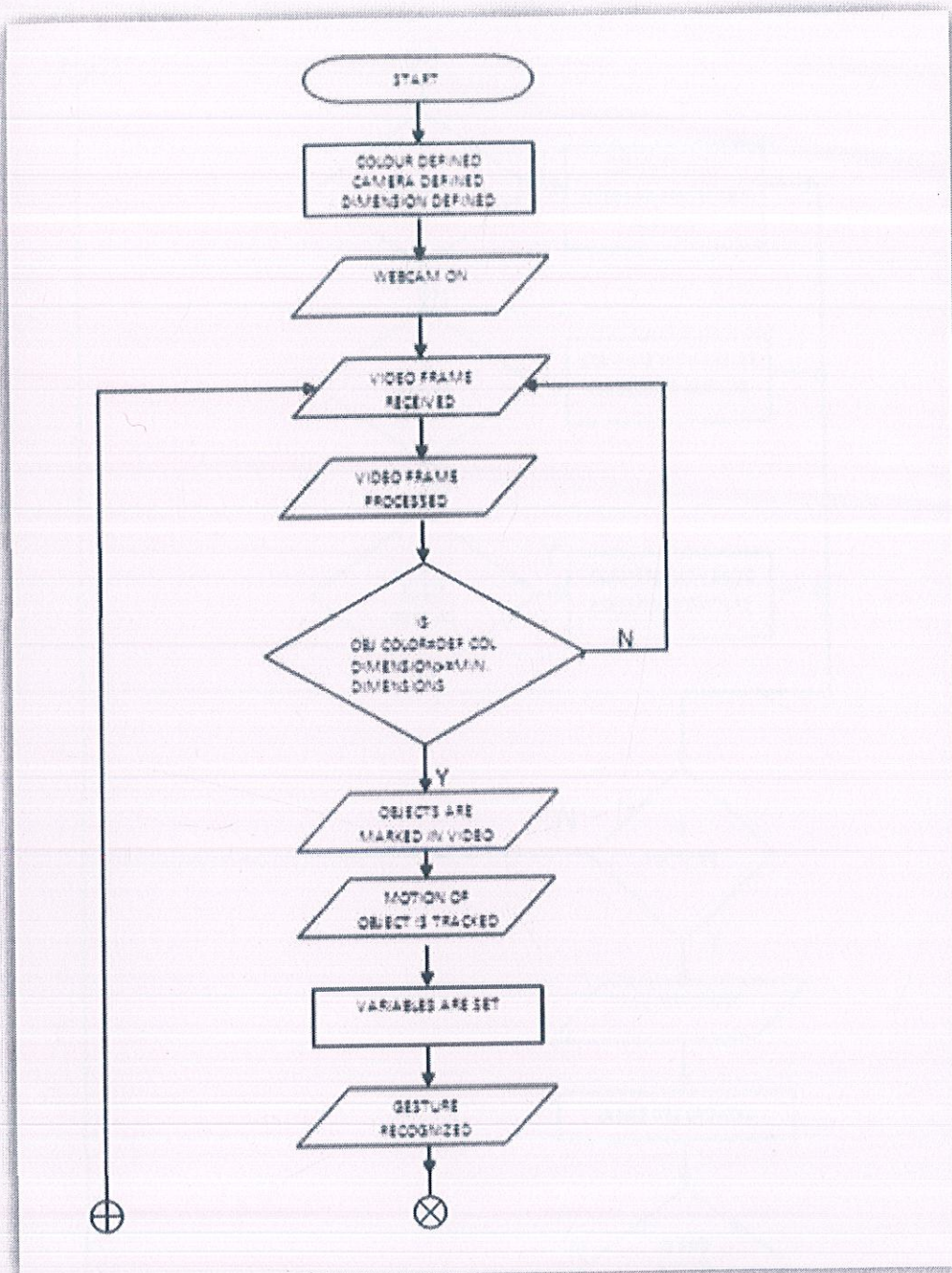


Figure 2: Data flow diagram - Part 1



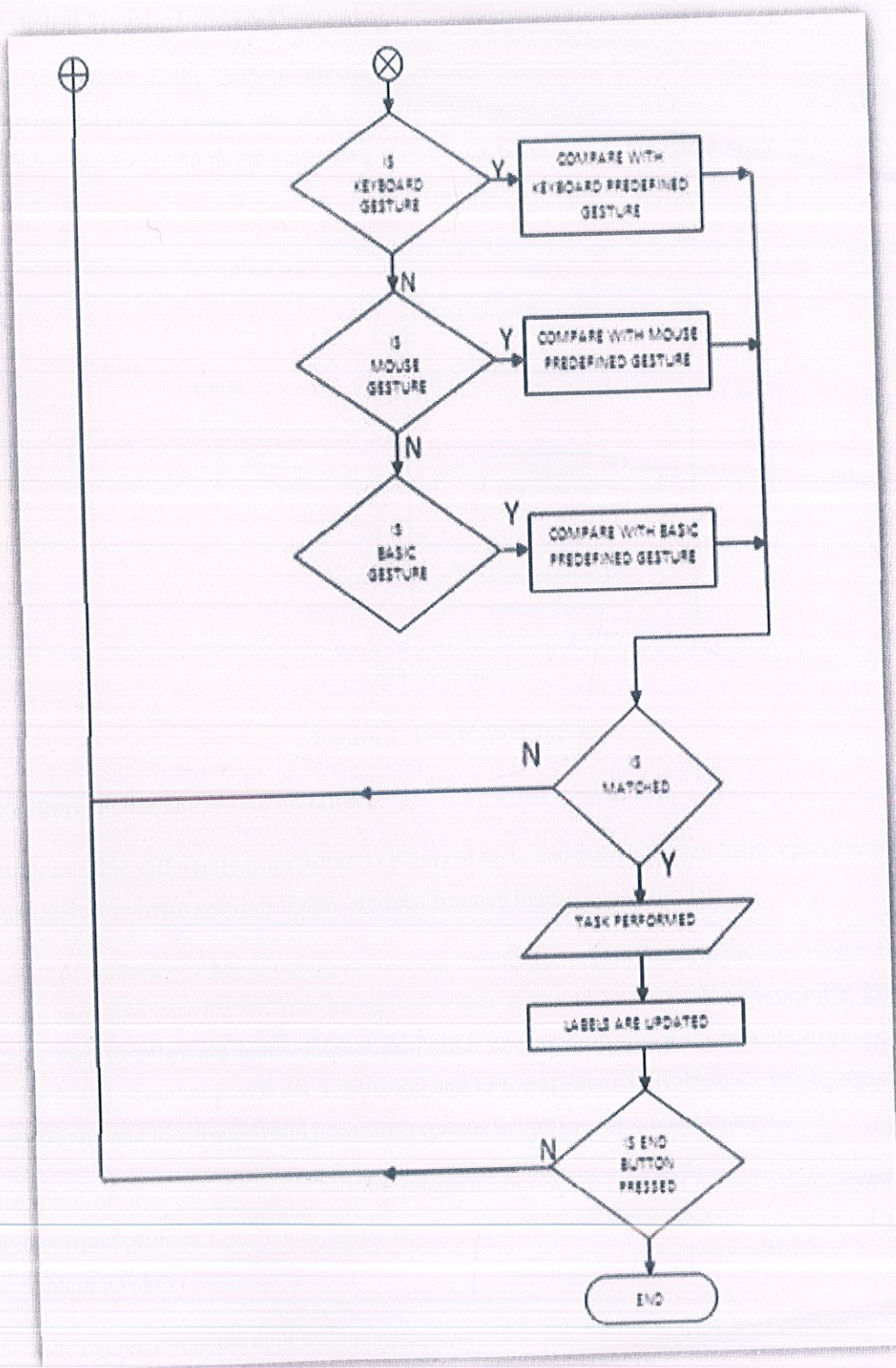
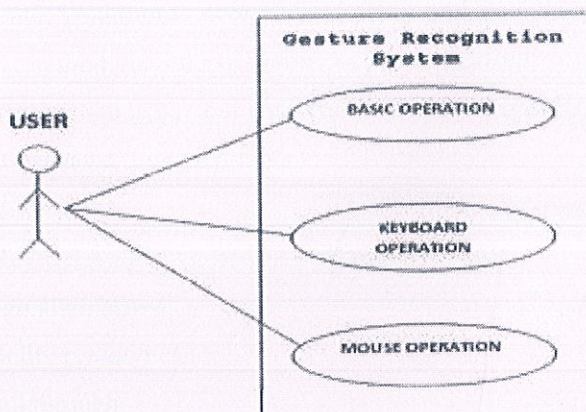


Figure 3: Data flow diagram - Part 2



### Behavioural Model – Use case Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



Use case diagram

Figure 4: Use Case diagram

### 2.6 Failure modes and action on failure

Gestures appear differently from different points of view and even a human being cannot recognize or can mistake certain gestures if they are seen from an unusual point of view.

Quality of camera is a big concern, resolution, sensitivity to light, color tone, etc. Lower camera resolution would provide inferior images in which it would be difficult to recognize gestures, shapes and the other attributes. Higher resolution images will take a lot of time to process. To handle this, the resolution was set to 640x480 and FPS was set to 15. Robustness to the image noise (background and foreground) and occlusions (partial or full) are also challenging issues.

The speed of detected gestures is usually related to the frame-rate: we cannot expect recognizing rapid gestures with an insufficient frame-rate (e.g. four frames per second is required for detecting the blink of an eye).

For reducing the errors the application is programmed to ask for the repetition of gesture to confirm execution.



## **Chapter 3 : Task Analysis and Schedule of Activities**

Task analysis is the analysis of how a task is accomplished, including a detailed description of both the manual and mental activities, task and element durations, task frequency, task allocation, task complexity, environmental conditions, necessary clothing and equipment, and any other unique factors involved in or required for one or more people to perform a given task. Task analysis emerged from research in applied behaviour analysis and still has considerable research in that area.

Step1. Creating a list of expected users and tasks

Step2. Validating the tasks

Step3. Deciding upon key users and a tentative list of requirements

Step4. Brainstorm design alternatives

Step5. Develop low fidelity prototypes

Step6. Task-centred walkthrough

Step7. Evaluation with users

### **3.1 Task decomposition**

Decomposition of project scope generally involves the following activities:

- Gather information on major project deliverables and analyse related tasks
- Start development of work breakdown structure(WBS) at the highest level
- Decompose the upper WBS levels into lower level detailed components
- Identify each work package & WBS components with unique code, and
- Verify if the degree of decomposition of the work is necessary and sufficient
- No. of Levels of WBS need not be same for all deliverables

But excessive decomposition may lead to more work without much value for the time spent. It can also leads to inefficient use of resources, and decreased work efficiency. So, knowing few basics about work package helps us in deciding the level of decomposition. Few of them are:

- Work package is the lowest level of WBS.
- Usually, a work package is the quantum of work which is assigned to a single resource as a whole and produces a verifiable outcome.



- Project's cost and schedule estimation is done at work package level.
- The accuracy of these estimations depends on the level of detailed work package that is defined.
- The level to which work packages need to be detailed vary from project to project.

In Time Management, each work package within the WBS is decomposed into the activities required to produce the work package deliverables. Decomposition is a technique used in project management that breaks down the workload and tasks before the creation of the work breakdown structure. This important step can save time in the long run.

### Overview of the Decomposition Process

Once you have determined the project objectives, you will need to gather the information involving the project's deliverables and the tasks that have already been determined. Knowing what needs to be produced as the end products and knowing the important milestones will help guide the project to keep it on course.

Step1. Identify Project Deliverables

Step2. Give each Deliverable its own Sheet of Paper

Step3. Deal with Each Deliverable Individually

Step4. Know When to Stop

Step5. Estimate Durations and Arrange Tasks into Work Packages

Step6. Estimate Costs



### 3.2 Incremental Model

Iterative and Incremental development is the heart of a cyclic software development process which was developed in response to the weaknesses of the waterfall model. It generally starts with an initial planning and ends with deployment with the cyclic interactions in between. The basic idea behind the agile method is to develop a system through the repeated cycles (iterative) and in the smaller portions at a time (incremental), allowing the software developers to take advantage of what was learnt during the development of earlier parts or versions of the system. Learning comes from both the development and the use of the system, where the possible key steps in the process start with a simple implementation of a subset of the software requirements and then iteratively enhance the evolving versions until the full system is implemented. At each iteration, the design modifications are made and some new functional capabilities are added.



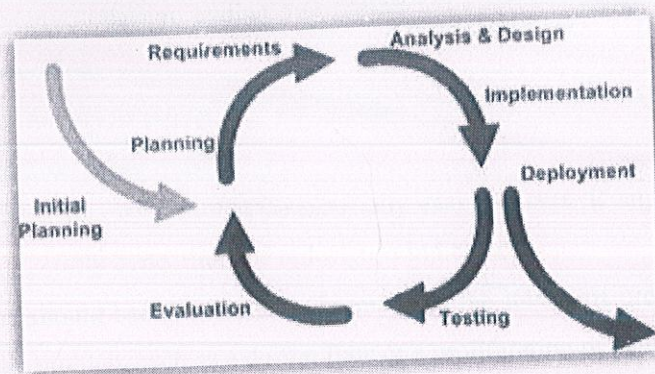


Figure 5: Incremental model

The procedure itself consists of the initialization step, the iteration step, and the Project Control List. The initialization step creates a base version of the system. The goal for this initial implementation is to create a product to which the user can react. It should offer a sampling of the key aspects of the problem and provide a solution that is simple enough to understand and easily implement. To guide the iteration process, a project control list is created that contains a record of all those tasks that are needed to be performed. It includes such items as the new features to be implemented and the areas of redesign of the existing solution. The control list is constantly being revised as a result of the analysis phase.

The iteration involves the redesign and implementation of a task from the project control list, and the analysis of the current version of the system. The analysis of iteration involves analysis of the structure, modularity, usability, reliability, efficiency, & achievement of the goals. Incremental development slices the system functionality into increments (portions). In each increment, a slice of functionality is delivered through cross-discipline work, from the requirements to the deployment. The unified process groups increments/iterations into different phases: inception, elaboration, construction, and transition.

Inception identifies the project scope, risks, and the requirements (functional and non-functional) at a high level but in enough detail that the work can be estimated.


Elaboration delivers a working architecture that mitigates the top risks and fulfils the non-functional requirements.

Construction incrementally fills-in the architecture with the production-ready code produced from analysis, design, implementation, and testing of all the functional requirements. Each of the phases may be divided into 1 or more iterations, which are usually time-boxed rather than feature-boxed. Architects and analysts work one iteration ahead of the developers and testers to keep their work-product backlog full. Guidelines that drive the implementation and analysis include:



- Any difficulty in the design, coding and testing a modification should signal the need for redesign or re-coding.
- Modifications should fit easily into the isolated and easy-to-find modules. If they do not, some redesign is possibly needed.
- Modifications to the tables should be especially easy to make. If any table modification is not quickly and easily done, then the redesign is indicated.
- Modifications should become easier to be made as the iterations progress. If they are not, there is a basic problem such as a design flaw or a proliferation of patches.
- Patches should normally be allowed to exist for only one or two iterations. Patches may be necessary to avoid the redesigning during an implementation phase.
- The existing implementation should be analysed frequently to determine how well it measures up to the project goals.
- Program analysis facilities should be used whenever available to aid in the analysis of partial implementations.
- User reaction should be solicited and analysed for the indications of deficiencies in the current implementation.

#### **Advantages**

1. After every iteration any faulty piece software can be easily identified as very few changes are done after every iteration.
2. It is easier to test and debug as testing and debugging can be performed after each iteration.
3. This model does not affect anyone's business values because they provide core of the software which a customer needs, which will indeed help that person to run his business.
4.  After establishing an overall architecture, the system is developed and delivered in increments.

#### **Disadvantages**

1. If initially, the requirements are thought to be stable but at later stages are realized to be unstable then the increments have to be withdrawn and have to be reworked.
2. The resulting cost may exceed the cost of the organization.
3. Problems may arise related to the system architecture.



### 3.3 Project schedule

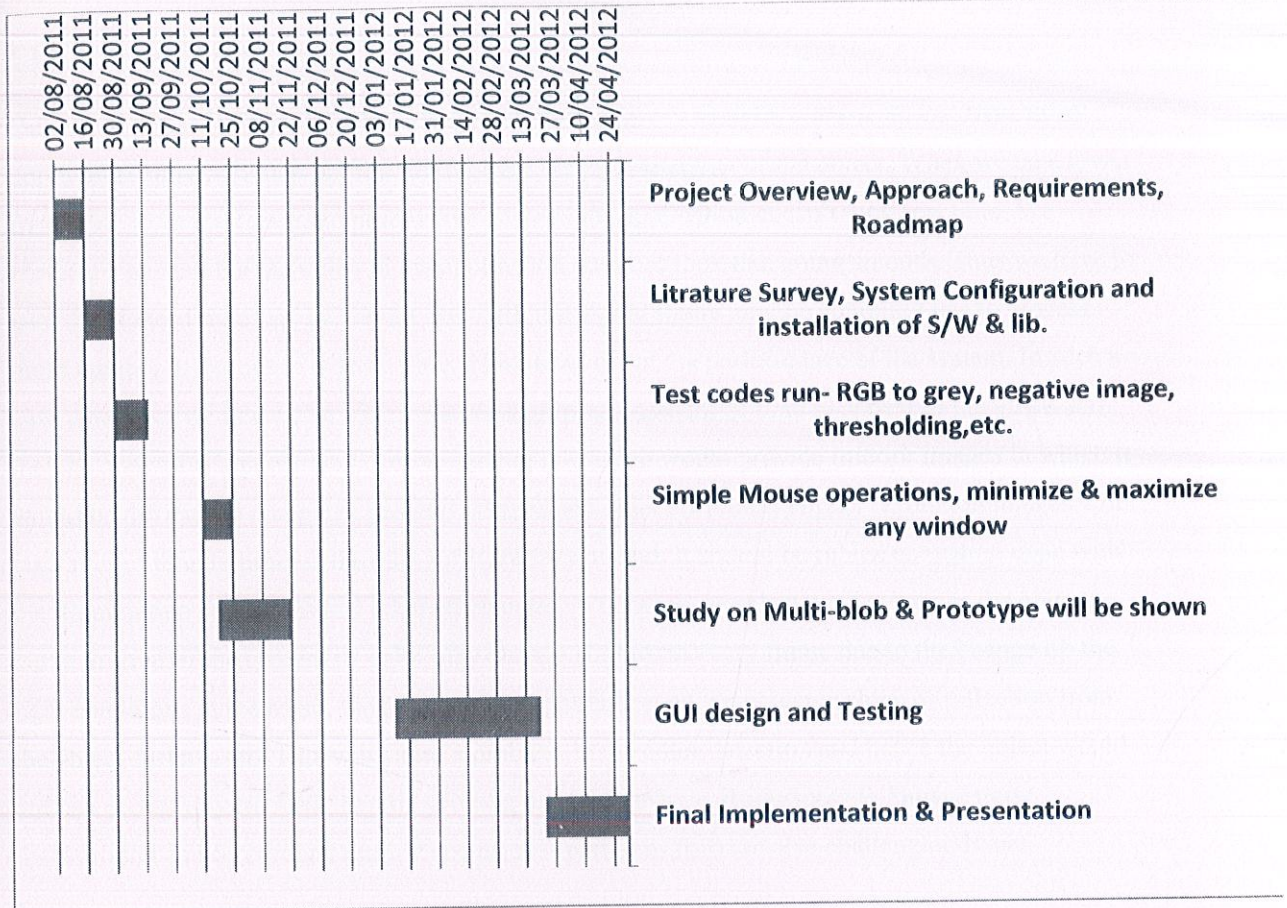


Figure 6: Gantt chart



## Chapter 4: Detection

### 4.1 Choice of Camera

Quality of camera is a big concern. Its resolution, sensitivity to light, color tone, etc. have to be optimum (neither too high nor too low) for the optimal detection of the object. When we take lower frame rate from the camera to process the gesture, the application doesn't look real time. We perform a gesture but the video in the application does not look like going smooth. Thus we have to take the higher frame rate i.e.,  $\geq 20$ . But with the higher frame rate the computer has to process more number of frames in a given time. This slows down the performance of the system. In such a case whenever we make a gesture it takes a significant amount of time to appear in the video and then an operation is performed. Lower camera resolution would provide inferior images in which it would be difficult to recognize gestures, shapes and other attributes. Higher resolution images will take a lot of time to process thus the software lags behind. It should be robust enough to cope with the illumination conditions and contrast changes. Whenever we select a color then as the objects move in front of the camera its color may change slightly in every frame due to the change of the light conditions in the room, the contrast, the light falling on the object, a shine or reflection from the object surface, etc. This will cause a problem in the color detection and hence the object would not be successfully tracked, thereby spoiling the performance. Robustness to \*image noise (background and foreground) and \*\*occlusions (partial or full) are also challenging issues.

There is no perfect low-level vision algorithm.

**Image Noise:** Image noise is the random variation of brightness or the color information in the images produced by the sensor and circuitry of a scanner or digital camera.

**Occlusion:** Occlusion means that there is something you want to see, but can't due to some property of your sensor setup, or some event.

### 4.2 Hardware Setup

Apart from the camera the other minimum hardware requirements for the application are as follows:

- RAM at least 512MB
- Hard disk at least 10GB



- Processor at least Pentium 4, 1.2GHz
- Integrated/Dedicated camera

The above hardware requirements are the minimum requirements for the .NET framework 3.0 which is an essential requirement for running .NET applications. Hence these hardware specifications have been also taken as the minimum requirements for our project.

### 4.3 Method of Colour Detection

#### Blob Detection

In the area of computer vision, blob detection refers to the visual modules that are aimed at detecting the points and/or the regions in the image that differ in the properties like brightness or color compared to the surrounding. There are two main classes of blob detectors (i) differential methods based on derivative expressions and (ii) methods based on local extrema in the intensity landscape. With the more recent terminology used in the field, these operators can also be referred to as the interest point operators, or alternatively the interest region operators (see also interest point detection and corner detection).

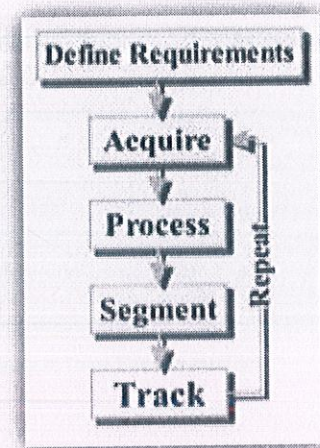


Figure 7: Process of Blob Detection

There are several motivations for studying and developing the blob detectors. One main reason is to provide some complementary information about the regions, which is not obtained from the edge detectors or corner detectors. In early work in the area, blob detection was used to obtain the regions of interest for further processing. These regions could signal the presence of the objects or the parts of objects in the image domain with application to the object recognition and/or object tracking. In other domains, such as histogram analysis, blob descriptors can also be used for peak detection by applying the concept of segmentation. Another common use of blob descriptors is as the main primitives for texture analysis and texture recognition. In more recent work, blob descriptors are being increasingly popular and used as interest points for wide baseline stereo matching and to signal the presence of informative image features for appearance-based object recognition based on the local image statistics. There is also the related notion of ridge detection to signal the presence of elongated objects.



## Pixel filtering by color

OpenCV provides a number of image processing filters, which allow to filter pixels depending on their color values. These image processing filters may be used to keep the pixels, which color falls inside or outside of specified range, and fill the rest of pixels with specified color.

Below is the list of implemented pixel filters and the result of their application to the source image.

### Color Filtering

The image processing filter filters pixels inside/outside of the specified RGB color range - it keeps pixels with colors inside/outside of specified range and fills the rest with specified color.

```
// create filter
ColorFiltering filter = new ColorFiltering( );
// set color ranges to keep
filter.Red = new IntRange( 100, 255 );
filter.Green = new IntRange( 0, 75 );
filter.Blue = new IntRange( 0, 75 );
// apply the filter
filter.ApplyInPlace( image );
```

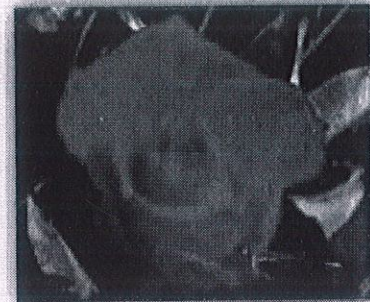


Figure 8: Color filtering method

### Channel Filtering

The channel filtering filter is similar to the above color filtering. It also operates in RGB color space, but does filtering of not entire pixels, but their RGB values. This means that pixel itself may not be filtered (will be kept), but one of its RGB values may be filtered if they are inside/outside of specified range.

```
// create filter
ChannelFiltering filter = new ChannelFiltering( );
// set channels' ranges to keep
filter.Red = new IntRange( 0, 255 );
filter.Green = new IntRange( 100, 255 );
filter.Blue = new IntRange( 100, 255 );
```

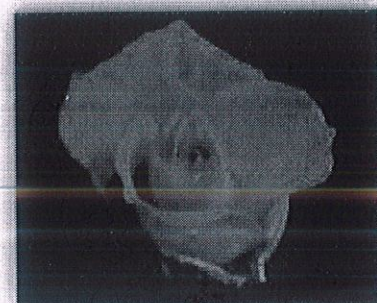


Figure 9: Channel filtering method



```
// apply the filter
filter.ApplyInPlace( image );
```

### **HSL Filtering**

The image processing filter operates in the HSL color space and filters pixels, which color is inside/outside of the specified HSL range - it keeps pixels with colors inside/outside of the specified range and fills the rest with specified color.

```
// create filter
HSLFiltering filter = new HSLFiltering( );
// set color ranges to keep
filter.Hue = new IntRange( 335, 0 );
filter.Saturation = new Range( 0.6f, 1 );
filter.Luminance = new Range( 0.1f, 1 );
// apply the filter
filter.ApplyInPlace( image );
```

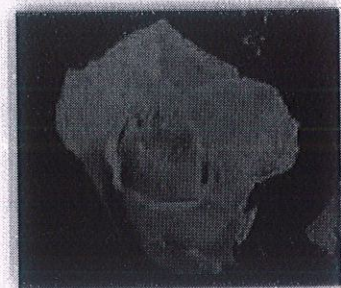


Figure 10: HSL filtering method

### **Euclidean Color Filtering**

The image processing filter filters pixels, which color is inside/outside of RGB sphere with specified center and radius - it keeps pixels with colors inside/outside of the specified sphere and fills the rest with specified color.

```
// create filter
EuclideanColorFiltering filter = new EuclideanColorFiltering( );
// set center color and radius
filter.CenterColor = Color.FromArgb( 215, 30, 30 );
filter.Radius = 100;
// apply the filter
filter.ApplyInPlace( image );
```



Figure 11: Euclidean filtering method



## Chapter 5: Recognition and Refinement

### 5.1 Recognition Strategy

**Direct method based on geometry:** Knowing that the hand is made up of bones of fixed width connected by joints which can only flex in certain directions and by limited angles it would be possible to calculate the silhouettes for a large number of hand gestures. Thus, it would be possible to take the silhouette information provided by the detection method and find the most likely gesture that corresponds to it by direct comparison. The benefit of this method is that it would require very little training and would be easy to extend to any number of gestures as required. However, the model for calculating the silhouette for any given gesture would be hard to construct and in order to attain a high degree of accuracy it would be necessary to model the effect of all light sources in the room on the shadows cast on the hand by itself.

**Learning method:** With this method the gesture set to be recognised would be “taught” to the system beforehand. Any given gesture could then be compared with the stored gestures and a match score calculated. The highest scoring gesture could then be displayed if its score was greater than some match quality threshold. The advantage of this system is that no prior information is required about the lighting conditions or the geometry of the hand for the system to work, as this information would be encoded into the system during training. The system would be faster than the above method if the gesture set was kept small. The disadvantage with this system is that each gesture would need to be trained at least once and for any degree of accuracy, several times. The gesture set is also likely to be user specific.

Another method was to take up a few parameters and make different combinations of it to perform different gestures. Number of objects, relative positions of objects, horizontal motion direction, vertical motion direction, speed of motion and distance of motion are the parameters whose different combinations mean different gestures. We decided to take up this method.

### 5.2 Selection of Gesture Set

In order to test any comparison metric devised it is important to have a constant set of easily reproducible gestures. It is also important to ensure that the gestures are not chosen to be as



dissimilar as possible so that the chances of confusion for the system and user and probability of errors are reduced.

### **5.3 Analysis of distortion and noise**

**Image distortion:** If the camera's visual axis is not perpendicular to the floor plane, a given gesture would appear different depending on the position and yaw of the hand (a given length in one area of the frame would appear longer or shorter in another area of the frame). This is termed as projective distortion. Also, if the camera lens is of poor quality then the straight sides of a true square in the frame would appear to be curved. This is termed radial as distortion.

**Image noise** is random (not present in the object imaged) variation of brightness or color information in the images, and is usually an aspect of electronic noise. It can be produced by the sensor and circuitry of a scanner or digital camera. Image noise can also originate in film grain and in the unavoidable shot noise of an ideal photon detector.

Image noise is an undesirable by-product while capturing image that adds spurious and extraneous information.

The original meaning of "noise" was and remains "unwanted sound"; unwanted electrical fluctuations in the signals received by AM radios caused audible acoustic noise ("static"). By analogy unwanted electrical fluctuations themselves came to be known as "noise".

#### **Image noise reduction**

Most algorithms for converting the image sensor data to an image, whether in-camera or on a computer, involve some form of noise reduction. There are many procedures for this, but all attempt to determine whether the actual differences in pixel values constitute noise or real photographic detail, and average out the former while attempting to preserve the latter. However, no algorithm can make this judgment perfectly, so there is often a tradeoff made between noise removal and preservation of fine, low-contrast detail that may have characteristics similar to noise. Many cameras have settings to control the aggressiveness of the in-camera noise reduction.

A simplified example of the impossibility of unambiguous noise reduction: an area of uniform red in an image might have a very small black part. If this is a single pixel, it is likely (but not certain) to be spurious and noise; if it covers a few pixels in an absolutely regular shape, it may be a defect in a group of pixels in the image-taking sensor (spurious and unwanted, but not strictly noise); if it



is irregular, it may be more likely to be a true feature of the image. But a definitive answer is not available.

This decision can be assisted by knowing the characteristics of the source image and of the human vision. Most noise reduction algorithms perform much more aggressive chroma noise reduction, since there is the little important fine chroma detail that one risks losing. Furthermore, many people find luminance noise less objectionable to the eye, since its textured appearance mimics the appearance of film grain.

The high sensitivity image quality of a given camera (or RAW development workflow) may depend greatly on the quality of the algorithm used for the noise reduction. Since noise levels increase as the ISO sensitivity is increased, most camera manufacturers increase the noise reduction aggressiveness automatically at higher sensitivities. This leads to the breakdown of image quality at higher sensitivities in two ways: noise levels increase and fine detail is smoothed out by the more aggressive noise reduction.

In cases of extreme noise, such as the astronomical images of very distant objects, it is not so much a matter of noise reduction as of extracting a little information buried in a lot of noise; techniques are different, seeking small regularities in massively random data.

#### **5.4 Removal of background objects**

The application has an adjustment for the minimum size of the objects to be detected. The size is defined in terms of pixels. If any object (of the same color as our specified color) is detected and its size does not satisfy the minimum height and minimum width parameters then this object is not detected. Often it happens that there are several objects in the background that happen to match with our specified color and are hence detected causing a complete misinterpretation of the gesture. Thus by specifying these size parameters, we can get rid of the background objects.

#### **5.5 Removal of unwanted colours**

The color of the object to be detected must be specified in the application. This can be done by selecting the color from the color dialog box or by color-picking in the image itself.

A value for color variation must be specified between 0 to 250. This allows the amount of deviation from the selected color. Whenever we select a color then as the objects move in front of the camera



its color may change slightly in every frame due to the change of light conditions in the room, the contrast, the light falling on the object, a shine or reflection from the object surface, etc. Thus by specifying the range of variation we can keep on detecting the object even after a slight color variation. This is also the way the projects maintains its robustness, acting perfectly in all conditions. Higher color range will allow more deviation while may also cause errors by detecting any unwanted objects. Lower range allows less deviation but may not always detect the object. Hence values must be set by moving the object here and there in front of the camera and then choosing the best value.

## **5.6 Overview of Algorithms**

**(See annexure 1 for the codes.)**



## Chapter 6: Conclusion

### 6.1 Project Goals

The goal of this project was to create a system to recognise a set of gestures, 3 for basic controls, 4 gestures for keyboard control and 6 gestures for mouse control.

It was considered that a modern computer system with Microsoft Windows operating system would allow the project goals to be exceeded.

### 6.2 Further Work

**Collection of additional gesture information:** The final system developed can recognise gestures using silhouette information alone. Although this was sufficient for the number of trained gestures, the accuracy would doubtless suffer if the number of gestures were increased. In order to remedy this, extra information about the test gesture would have to be gathered, such as edge information.

**Removal of coloured object:** The system relies on the using a coloured object to remove various degrees of freedom, making recognition, via comparison, possible. It would be advantageous if this were not the case. There are methods that could be used to perform the recognition without any object directly with the hand but they would be unlikely to be as accurate.

**Detection of more than one color:** If the system is able to detect more than one color then we will have one more parameter of color which can be used to increase the number of gestures by a factor of 'n' where n is the no. of different colors that can be detected.

**Increase of the number of recognised gestures:** For the purposes of a man-machine interface a relatively small set of gestures ( $\approx 100$ ) would be sufficient. However, if detection of hand gestures for computer animation is needed (for instance), then the no. of trained gestures required would be around thousands. A system which relies on both training and comparison of all gestures used would not be sufficient for this task..

**Multi-stage gestures:** It would be possible to represent a much larger number of gestures if each consisted of two or more gestures combined with hand position changes. For instance, the "wave hello" label could correspond to the open hand gesture with an alternating increase and decrease of hand yaw angle and the "thumbs-up" label could correspond to some other operation followed by the space gesture.

**Two-handed gestures:** It would be possible to detect the gesture signed by both hands in the frame. A method would have to be devised to detect a gesture (or range of gestures) that is represented by a partially occluded hand. This method would be considerably harder to implement. It should be



noted that although the gesture of both hands could be recognised this would not permit the recognition of the full gesture.



## 7. References

### **Websites:**

- [1]. <http://www.cs.iit.edu/~agam/cs512/lect-notes/opencv-intro/>
- [2]. [http://www.roborealm.com/tutorial/color\\_object\\_tracking\\_2/slide010.php](http://www.roborealm.com/tutorial/color_object_tracking_2/slide010.php)
- [3]. [http://www.scholarpedia.org/article/Multiple\\_object\\_tracking](http://www.scholarpedia.org/article/Multiple_object_tracking)
- [4]. [http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/COHEN/gesture\\_overview.html](http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/COHEN/gesture_overview.html)
- [5]. <http://channel9.msdn.com/coding4fun/kinect/Open-source-Kinect-gesture-recognition-project-Kinect-DTW>
- [6]. <http://philosophe.com/design/requirements/>
- [7]. <http://www.brighthub.com/office/project-management/articles/15314.aspx>
- [8]. <http://www.fdsc.net/development-lifecycles-and-models/incremental-model.html>
- [9]. <http://www.softdevteam.com/Incremental-lifecycle.asp>
- [10]. <http://www.smartdraw.com/resources/tutorials/data-flow-diagrams/>
- [11]. <http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm>
- [12]. <http://www.sprawls.org/ppmi2/NOISE/>
- [13]. <http://www.cambridgeincolour.com/tutorials/image-noise.htm>
- [14]. <http://www.sciencedaily.com/releases/2010/06/100617132218.htm>
- [15]. <http://www.imagemagick.org/Usage/distorts/>
- [16]. <http://www.cosmin.com/colordetector/>

### **Books:**

- [1]. Face Detection and Gesture Recognition for Human-Computer Interaction - by Ming-Hsuan Yang (Author), Narendra Ahuja (Author).
- [2]. Learning OpenCV; computer vision with OpenCV library – Gary Bradski, Adrian Kaehler.
- [3]. Hidden Markov Models for Gesture Recognition by - Donald O. Tanguay, Jr.
- [4]. Digital Image Processing (Second Edition) by Rafael C. Gonzalez, MedData Interactive.



## Chapter 8: Annexure

### Codes and Snapshots:

// Segmentation : Scribbling Application

```
#include "stdafx.h"
```

```
#include <opencv2\opencv.hpp>
```

```
IplImage* GetThresholdedImage(IplImage* img)
{
    IplImage* imgHSV = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV, CV_BGR2HSV);
    IplImage* imgThreshed = cvCreateImage(cvGetSize(img), 8, 1);
    cvInRangeS(imgHSV, cvScalar(109, 100, 100), cvScalar(111, 255, 255), imgThreshed);
    cvReleaseImage(&imgHSV);
    return imgThreshed;
}
```

```
int main()
```

```
{
    // Initialize capturing live feed from the camera
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(0);
    if(!capture)
    {
        printf("Could not initialize capturing...\n");
        return -1;
    }

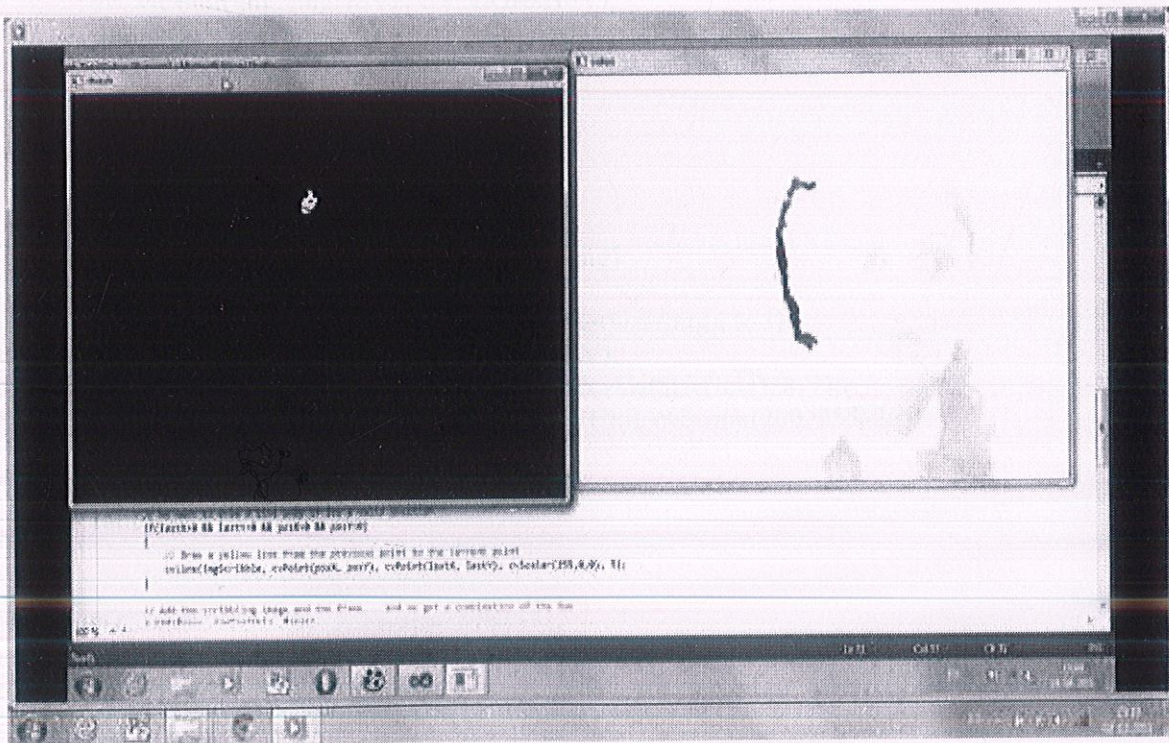
    cvNamedWindow("video");
    cvNamedWindow("ThreshWin");
    IplImage* imgScribble = NULL;
    while(true)
    {
        IplImage* frame = 0;
        frame = cvQueryFrame(capture);
        if(!frame)
            break;
        if(imgScribble == NULL)
        {
            imgScribble = cvCreateImage(cvGetSize(frame), 8, 3);
        }
        IplImage* imgBlueThresh = GetThresholdedImage(frame);
        CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(imgBlueThresh, moments, 1);
        double moment10 = cvGetSpatialMoment(moments, 1, 0);
        double moment01 = cvGetSpatialMoment(moments, 0, 1);
        double area = cvGetCentralMoment(moments, 0, 0);
    }
}
```



```

static int Xpos = 0;
static int Ypos = 0;
int Xlast = Xpos;
int Ylast = Ypos;
Xpos = (moment10/area);
Ypos = moment01/area;
printf("position (%d, %d)\n", Xpos, Ypos);
if(Xlast>0 && Ylast>0 && Xpos>0 && Ypos>0 && area>225)
{
    cvLine(imgScribble, cvPoint(Xpos, Ypos), cvPoint(Xlast, Ylast), cvScalar(255,0,0), 5);
}
cvAdd(frame, imgScribble, frame);
cvShowImage("ThreshWin", imgBlueThresh);
cvShowImage("video", frame);
int c = cvWaitKey(10);
if(c!=-1)
{
    break;
}
cvReleaseImage(&imgYellowThresh);
delete moments;
}
cvReleaseCapture(&capture);
return 0;
}

```





```
// Multi-Blob detection
```

```
#include "stdafx.h"
#include <opencv2\opencv.hpp>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <time.h>
```

```
IplImage* GetThresholdedImageYellow(IplImage* img)
```

```
{
    IplImage* imgHSV1 = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV1, CV_BGR2HSV);
    IplImage* imgThreshed1 = cvCreateImage(cvGetSize(img), 8, 1);
    cvInRangeS(imgHSV1, cvScalar(30, 100, 100), cvScalar(60, 255, 255),
    imgThreshed1); //yellow
    cvReleaseImage(&imgHSV1);
    return imgThreshed1;
}
```

```
IplImage* GetThresholdedImageBlue(IplImage* img)
```

```
{
    IplImage* imgHSV2 = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV2, CV_BGR2HSV);
    IplImage* imgThreshed2 = cvCreateImage(cvGetSize(img), 8, 1);
    cvInRangeS(imgHSV2, cvScalar(140, 100, 100), cvScalar(160, 255, 255),
    imgThreshed2); //blue
    cvReleaseImage(&imgHSV2);
    return imgThreshed2;
}
```

```
IplImage* GetThresholdedImagePink(IplImage* img)
```

```
{
    IplImage* imgHSV3 = cvCreateImage(cvGetSize(img), 8, 3);
    cvCvtColor(img, imgHSV3, CV_BGR2HSV);
    IplImage* imgThreshed3 = cvCreateImage(cvGetSize(img), 8, 1);
    cvInRangeS(imgHSV3, cvScalar(160, 100, 100), cvScalar(200, 255, 255),
    imgThreshed3); //pink
    cvReleaseImage(&imgHSV3);
    return imgThreshed3;
}
```

```
int main()
```

```
{
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(1);
    if(!capture)
    {
        printf("Could not initialize capturing...\n");
    }
}
```



```

return -1;
}
cvNamedWindow("video",CV_WINDOW_AUTOSIZE);
cvNamedWindow("ThreshWin",CV_WINDOW_AUTOSIZE);
while(true)
{
    IplImage* frame = 0;
    frame = cvQueryFrame(capture);
    if(!frame)
        break;
    IplImage* imgYellowThresh = GetThresholdedImageYellow(frame);
    IplImage* imgBlueThresh = GetThresholdedImageBlue(frame);
    IplImage* imgPinkThresh = GetThresholdedImagePink(frame);
    CvMoments *moments1 = (CvMoments*)malloc(sizeof(CvMoments));
    CvMoments *moments2 = (CvMoments*)malloc(sizeof(CvMoments));
    CvMoments *moments3 = (CvMoments*)malloc(sizeof(CvMoments));
    cvMoments(imgYellowThresh, moments1, 1);
    cvMoments(imgBlueThresh, moments2, 1);
    cvMoments(imgPinkThresh, moments3, 1);
    int moment10a = cvGetSpatialMoment(moments1, 1, 0);
    int moment01a = cvGetSpatialMoment(moments1, 0, 1);
    int moment10b = cvGetSpatialMoment(moments2, 1, 0);
    int moment01b = cvGetSpatialMoment(moments2, 0, 1);
    int moment10c = cvGetSpatialMoment(moments3, 0, 1);
    int moment01c = cvGetSpatialMoment(moments3, 0, 1);
    double area1 = cvGetCentralMoment(moments1, 0, 0);
    double area2 = cvGetCentralMoment(moments2, 0, 0);
    double area3 = cvGetCentralMoment(moments3, 0, 0);
    static int Xpos1 = 0;
    static int Xpos2 = 0;
    static int Ypos1 = 0;
    static int Ypos2 = 0;
    static int Xpos3 = 0;
    static int Ypos3 = 0;
    int Xlast1 = Xpos1;
    int Xlast2 = Xpos2;
    int Ylast1 = Ypos1;
    int Ylast2 = Ypos2;
    int Xlast3 = Xpos3;
    int Ylast3 = Ypos3;
    Xpos1 = (moment10a/area1);
    Xpos2 = (moment10b/area2);
    Ypos1 = (moment01a/area1);
    Ypos2 = (moment01b/area2);
    Xpos3 = (moment10c/area3);
    Ypos3 = (moment01c/area3);
    if(Xlast1>0 && Ylast1>0 && Xpos1>0 && Ypos1>0 && area1>100)
    {
        cvCircle(frame,cvPoint(Xpos2,Ypos2),30,cvScalar(0,0,255),3,8,0);
    }
    if(Xlast2>0 && Ylast2>0 && Xpos2>0 && Ypos2>0 && area2>100)

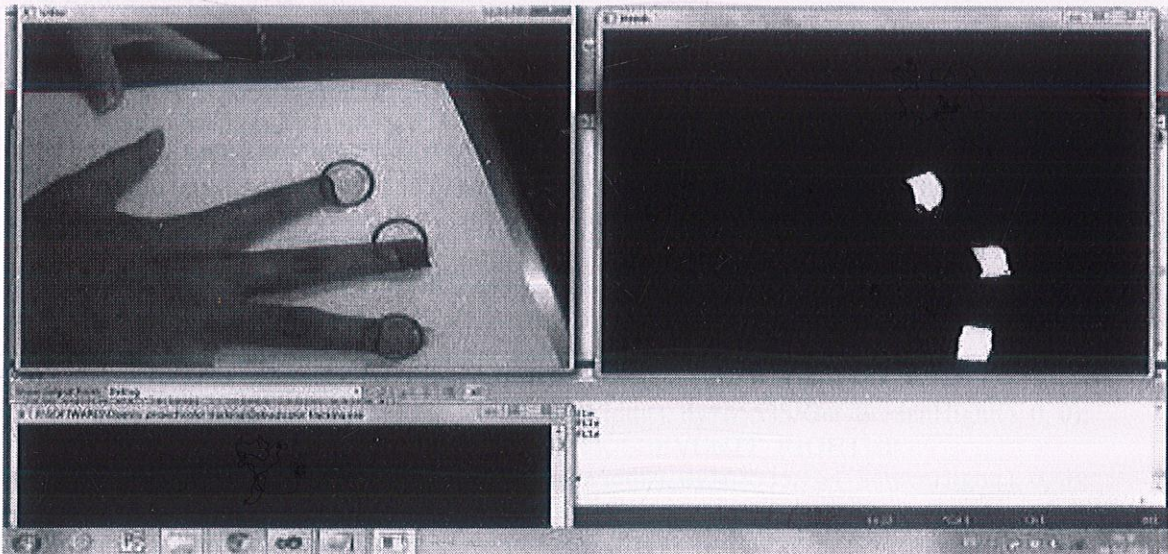
```



```

{
    cvCircle(frame,cvPoint(Xpos2,Ypos2),30,cvScalar(0,0,255),3,8,0);
}
if(Xlast3>0 && Ylast3>0 && Xpos3>0 && Ypos3>0 && area3>100)
{
    cvCircle(frame,cvPoint(Xpos2,Ypos2),30,cvScalar(0,0,255),3,8,0);
}
cvAdd(imgYellowThresh, imgBlueThresh, imgYellowThresh);
cvAdd(imgYellowThresh, imgPinkThresh, imgYellowThresh);
cvShowImage("ThreshWin", imgYellowThresh);
cvShowImage("video", frame);
int c = cvWaitKey(10);
if(c!=-1)
{
    break;
}
cvReleaseImage(&imgBlueThresh);
cvReleaseImage(&imgPinkThresh);
delete moments1;
delete moments2;
delete moments3;
}
cvReleaseCapture(&capture);
return 0;
}

```



//Mouse Movements

```

#include "stdafx.h"
#include <opencv2\opencv.hpp>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

```



```
#include <cv.h>
#include <highgui.h>
#include <time.h>
```

```
LONG Get_ScreenWidth()
```

```
{
    RECT rect;
    GetWindowRect(GetDesktopWindow(), &rect);    // Get Desktop rectangle
    return rect.right - rect.left;
}
```

```
LONG Get_ScreenHight()
```

```
{
    RECT rect;
    GetWindowRect(GetDesktopWindow(), &rect);    // Get Desktop rectangle
    return rect.bottom - rect.top;
}
```

```
void Mouse_Move(DWORD dx, DWORD dy)
```

```
{
    DWORD event=0;
    event = MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_MOVE;
    mouse_event(event, dx*65535/Get_ScreenWidth(), dy*65535/Get_ScreenHight(), 0, 0);
}
```

```
void Left_Click(DWORD dx, DWORD dy)
```

```
{
    DWORD event=0;
    event = MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTDOWN;
    mouse_event(event, dx*65535/Get_ScreenWidth(), dy*65535/Get_ScreenHight(), 0, 0);
    event = MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_LEFTUP;
    mouse_event(event, dx*65535/Get_ScreenWidth(), dy*65535/Get_ScreenHight(), 0, 0);
}
```

```
void Right_Click(DWORD dx, DWORD dy)
```

```
{
    DWORD event=0;
    event = MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_RIGHTDOWN;
    mouse_event(event, dx*65535/Get_ScreenWidth(), dy*65535/Get_ScreenHight(), 0, 0);
    event = MOUSEEVENTF_ABSOLUTE|MOUSEEVENTF_RIGHTUP;
    mouse_event(event, dx*65535/Get_ScreenWidth(), dy*65535/Get_ScreenHight(), 0, 0);
}
```

```
IplImage* GetThresholdedImageYellow(IplImage* img)
```

```
{
    // Same as in the above code
}
```

```
IplImage* GetThresholdedImageBlue(IplImage* img)
```

```
{
    // Same as in the above code }
}
```

```
IplImage* GetThresholdedImagePink(IplImage* img)
```



```

{
    // Same as in the above code
}

int main()
{
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(1);
    if(!capture)
    {
        printf("Could not initialize capturing...\n");
        return -1;
    }
    cvNamedWindow("video",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("ThreshWin",CV_WINDOW_AUTOSIZE);
    time_t initial, finnal, initialp, finnalp;
    int flag=0, flagp=0;
    double initime, fintime,dtime=0, initimep, fintimep,dtimep=0;
    while(true)
    {
        IplImage* frame = 0;
        frame = cvQueryFrame(capture);
        if(!frame)
            break;
        IplImage* imgYellowThresh = GetThresholdedImageYellow(frame);
        IplImage* imgBlueThresh = GetThresholdedImageBlue(frame);
        IplImage* imgPinkThresh = GetThresholdedImagePink(frame);
        CvMoments *moments1 = (CvMoments*)malloc(sizeof(CvMoments));
        CvMoments *moments2 = (CvMoments*)malloc(sizeof(CvMoments));
        CvMoments *moments3 = (CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(imgYellowThresh, moments1, 1);
        cvMoments(imgBlueThresh, moments2, 1);
        cvMoments(imgPinkThresh, moments3, 1);
        int moment10a = cvGetSpatialMoment(moments1, 1, 0);
        int moment01a = cvGetSpatialMoment(moments1, 0, 1);
        int moment10b = cvGetSpatialMoment(moments2, 1, 0);
        int moment01b = cvGetSpatialMoment(moments2, 0, 1);
        int moment01c = cvGetSpatialMoment(moments3, 0, 1);
        int moment10c = cvGetSpatialMoment(moments3, 0, 1);
        double area1 = cvGetCentralMoment(moments1, 0, 0);
        double area2 = cvGetCentralMoment(moments2, 0, 0);
        double area3 = cvGetCentralMoment(moments3, 0, 0);
        static int Xpos1 = 0; static int Xpos2 = 0;
        static int Ypos1 = 0; static int Ypos2 = 0;
        static int Xpos3 = 0; static int Ypos3 = 0;
        int Xlast1 = Xpos1; int Xlast2 = Xpos2;
        int Ylast1 = Ypos1; int Ylast2 = Ypos2;
        int Xlast3 = Xpos3; int Ylast3 = Ypos3;
        Xpos1 = (moment10a/area1); Xpos2 = (moment10b/area2);
        Ypos1 = (moment01a/area1); Ypos2 = (moment01b/area2);
        Xpos3 = (moment10c/area3); Ypos3 = (moment01c/area3);
    }
}

```



```

if(Xpos1<0 || Ypos1<0)
{
    Xpos1=0;    Ypos1=0;
}
if(Xlast2>0 && Ylast2>0 && Xpos2>0 && Ypos2>0 && area2>100)
{
    cvCircle(frame,cvPoint(Xpos2,Ypos2),30,cvScalar(0,0,255),3,8,0);
    Mouse_Move(((1366/640)*Xpos2),((1024/480)*Ypos2));
}
if(Xpos1!=0 && Ypos1!=0 && flag==0 && area1>100)
{
    initime=time(&initial);
    dtime=0;    flag=1;
    cvCircle(frame,cvPoint(Xpos1,Ypos1),30,cvScalar(0,0,255),3,8,0);
}
if(Xpos1==0 && Ypos1==0 && flag==1)
{
    fintime=time(&finnal);
    dtime = fintime-initime;
}
if(Xpos1==0 && Ypos1==0 && (dtime<1 || dtime>=3))
{
    flag=0;
}
if(dtime>=1 && flag==1)
{
    printf("Left Click Occured, %f",dtime);
    Left_Click(((1366/640)*Xpos2),((1024/480)*Ypos2));
    flag=0;
}
if(Xpos3<0 || Ypos3<0)
{
    Xpos3=0;    Ypos3=0;
}
if(Xpos3!=0 && Ypos3!=0 && flagp==0 && area3>100)
{
    initimep=time(&initialp);
    dtimep=0;    flagp=1;
    cvCircle(frame,cvPoint(Xpos3,Ypos3),30,cvScalar(0,0,255),3,8,0);
}
if(Xpos3==0 && Ypos3==0 && flagp==1)
{
    fintimep=time(&finnalp);
    dtimep = fintimep-initimep;
}
if(Xpos3==0 && Ypos3==0 && (dtimep<1 || dtimep>=3))
{
    flagp=0;
}
if(dtimep>=1 && flagp==1)
{

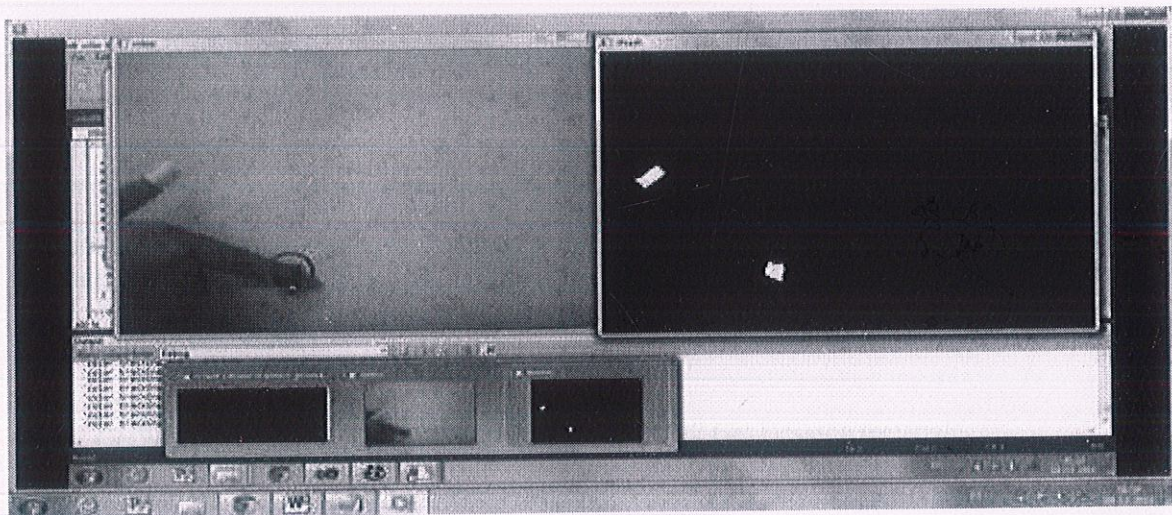
```



```

        printf("Right Click Occured, %f", dtimpep);
        Right_Click(((1366/640)*Xpos3),((1024/480)*Ypos3));
        flagp=0;
    }
    cvAdd(imgYellowThresh, imgBlueThresh, imgYellowThresh);
    cvAdd(imgYellowThresh, imgPinkThresh, imgYellowThresh);
    cvShowImage("ThreshWin", imgYellowThresh);
    cvShowImage("video", frame);
    int c = cvWaitKey(10);
    if(c!=-1)
    { break; }
    cvReleaseImage(&imgBlueThresh);
    cvReleaseImage(&imgPinkThresh);
    delete moments1;
    delete moments2;
    delete moments3;
}
cvReleaseCapture(&capture);
return 0;
}

```









```

        // Same as in the above code
    }

void Right_Click(DWORD dx,DWORD dy)
{
    // Same as in the above code
}

IplImage* GetThresholdedImageYellow(IplImage* img)
{
    // Same as in the above code
}

IplImage* GetThresholdedImageBlue(IplImage* img)
{
    // Same as in the above code
}

IplImage* GetThresholdedImagePink(IplImage* img)
{
    // Same as in the above code
}

int main()
{
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(1);
    if(!capture)
    {
        printf("Could not initialize capturing...\n");
        return -1;
    }
    cvNamedWindow("video",CV_WINDOW_AUTOSIZE);
    cvNamedWindow("ThreshWin",CV_WINDOW_AUTOSIZE);
    time_t initial, finnal, initialp, finnalp;
    int flag=0, flagp=0;
    double initime, fintime,dtime=0, initimep, fintimep,dtimep=0, initime3, fintime3,dtime3=0;
    while(true)
    {
        IplImage* frame = 0;
        frame = cvQueryFrame(capture);
        if(!frame)
            break;
        IplImage* imgYellowThresh = GetThresholdedImageYellow(frame);
        IplImage* imgBlueThresh = GetThresholdedImageBlue(frame);
        IplImage* imgPinkThresh = GetThresholdedImagePink(frame);
        CvMoments *moments1 = (CvMoments*)malloc(sizeof(CvMoments));
        CvMoments *moments2 = (CvMoments*)malloc(sizeof(CvMoments));
        CvMoments *moments3 = (CvMoments*)malloc(sizeof(CvMoments));
        cvMoments(imgYellowThresh, moments1, 1);
        cvMoments(imgBlueThresh, moments2, 1);
        cvMoments(imgPinkThresh, moments3, 1);
        int moment10a = cvGetSpatialMoment(moments1, 1, 0);
    }
}

```



```

int moment01a = cvGetSpatialMoment(moments1, 0, 1);
int moment10b = cvGetSpatialMoment(moments2, 1, 0);
int moment01b = cvGetSpatialMoment(moments2, 0, 1);
int moment01c = cvGetSpatialMoment(moments3, 0, 1);
int moment10c = cvGetSpatialMoment(moments3, 0, 1);
double area1 = cvGetCentralMoment(moments1, 0, 0);
double area2 = cvGetCentralMoment(moments2, 0, 0);
double area3 = cvGetCentralMoment(moments3, 0, 0);
static int Xpos1 = 0; static int Xpos2 = 0;
static int Ypos1 = 0; static int Ypos2 = 0;
static int Xpos3 = 0; static int Ypos3 = 0;
int Xlast1 = Xpos1; int Xlast2 = Xpos2;
int Ylast1 = Ypos1; int Ylast2 = Ypos2;
int Xlast3 = Xpos3; int Ylast3 = Ypos3;
Xpos1 = (moment10a/area1); Xpos2 = (moment10b/area2);
Ypos1 = (moment01a/area1); Ypos2 = (moment01b/area2);
Xpos3 = (moment10c/area3); Ypos3 = (moment01c/area3);
if(Xpos1<0 || Ypos1<0)
{
    Xpos1=0;    Ypos1=0;
}
if(Xpos3<0 || Ypos3<0)
{
    Xpos3=0;    Ypos3=0;
}
if(Xpos2<0 || Ypos2<0)
{
    Xpos2=0;    Ypos2=0;
}
if(Xpos1!=0 && Ypos1!=0 && flag==0 && area1>100 && (Xpos3==0 ||
Ypos3==0))
{
    initime=time(&initial);
    dtime=0;    flag=1;
    cvCircle(frame,cvPoint(Xpos1,Ypos1),30,cvScalar(0,0,255),3,8,0);
}
if(Xpos1==0 && Ypos1==0 && flag==1)
{
    fintime=time(&finnal);
    dtime = fintime-initime;
}
if(Xpos1==0 && Ypos1==0 && (dtime<1 || dtime>=3))
{
    flag=0;
}
if(dtime>=1 && flag==1)
{
    printf("1");
    gest=1;
    flag=0;
}

```



```

if(Xpos3!=0 && Ypos3!=0 && flagp==0 && area3>100 && area1>100 &&
Xpos1!=0 && Ypos1!=0 && Xpos2==0 && Ypos2==0)
{
    initimep=time(&initialp);
    dtimep=0;    flagp=1;
}
if(Xpos3==0 && Ypos3==0 && Xpos1==0 && Ypos1==0 && flagp==1)
{
    fintimep=time(&finnalp);
    dtimep = fintimep-initimep;
}
if(Xpos3==0 && Ypos3==0 && Xpos1==0 && Ypos1==0 && (dtimep<1 ||
dtimep>=3))
{
    flagp=0;
}
if(dtimep>=1 && flagp==1)
{
    printf("2");
    gest=2;
    flagp=0;
}
if(Xpos3!=0 && Ypos3!=0 && flag3==0 && area3>100 && area1>100 &&
Xpos1!=0 && Ypos1!=0 && area2>100 && Xpos2!=0 && Ypos2!=0)
{
    initime3=time(&initial3);
    dtime3=0;    flag3=1;
}
if(Xpos3==0 && Ypos3==0 && Xpos1==0 && Ypos1==0 && Xpos2==0 &&
Ypos2==0 && flag3==1)
{
    fintime3=time(&finnal3);
    dtime3 = fintime3-initime3;
}
if(Xpos3==0 && Ypos3==0 && Xpos1==0 && Ypos1==0 && Xpos2==0 &&
Ypos2==0 && (dtime3<1 || dtime3>=3))
{
    flag3=0;
}
if(dtime3>=1 && flag3==1)
{
    printf("3");
    gest=3;
    flag3=0;
}
switch(gest)
{
case 0: break;
case 1: {
    (void) system("F://final.ppt");
    break;
}
}

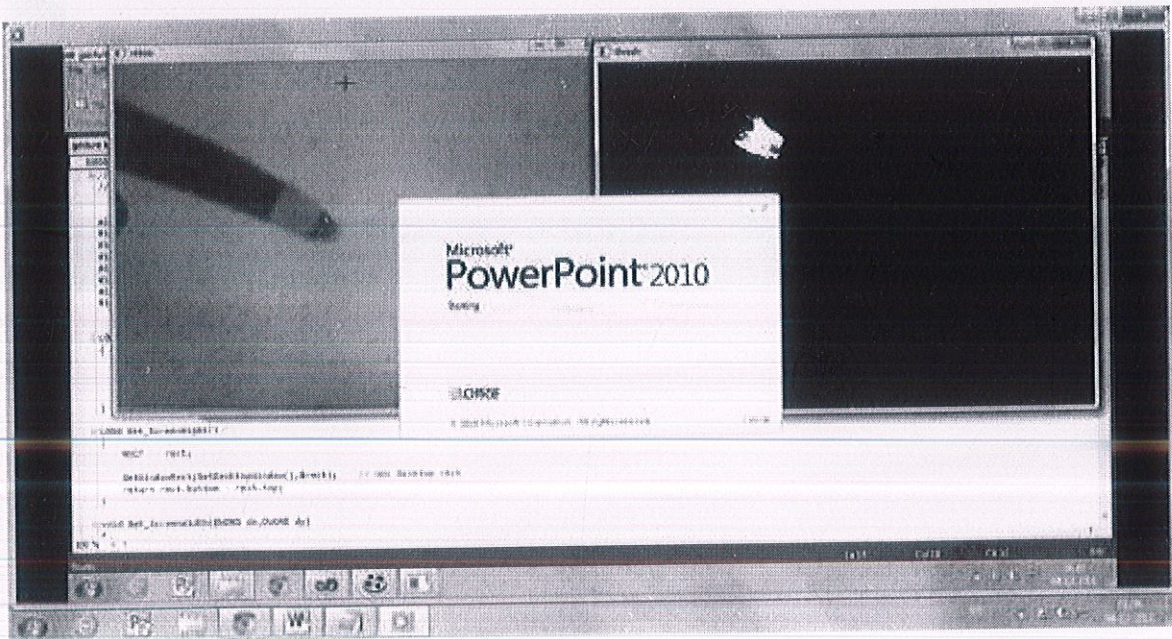
```



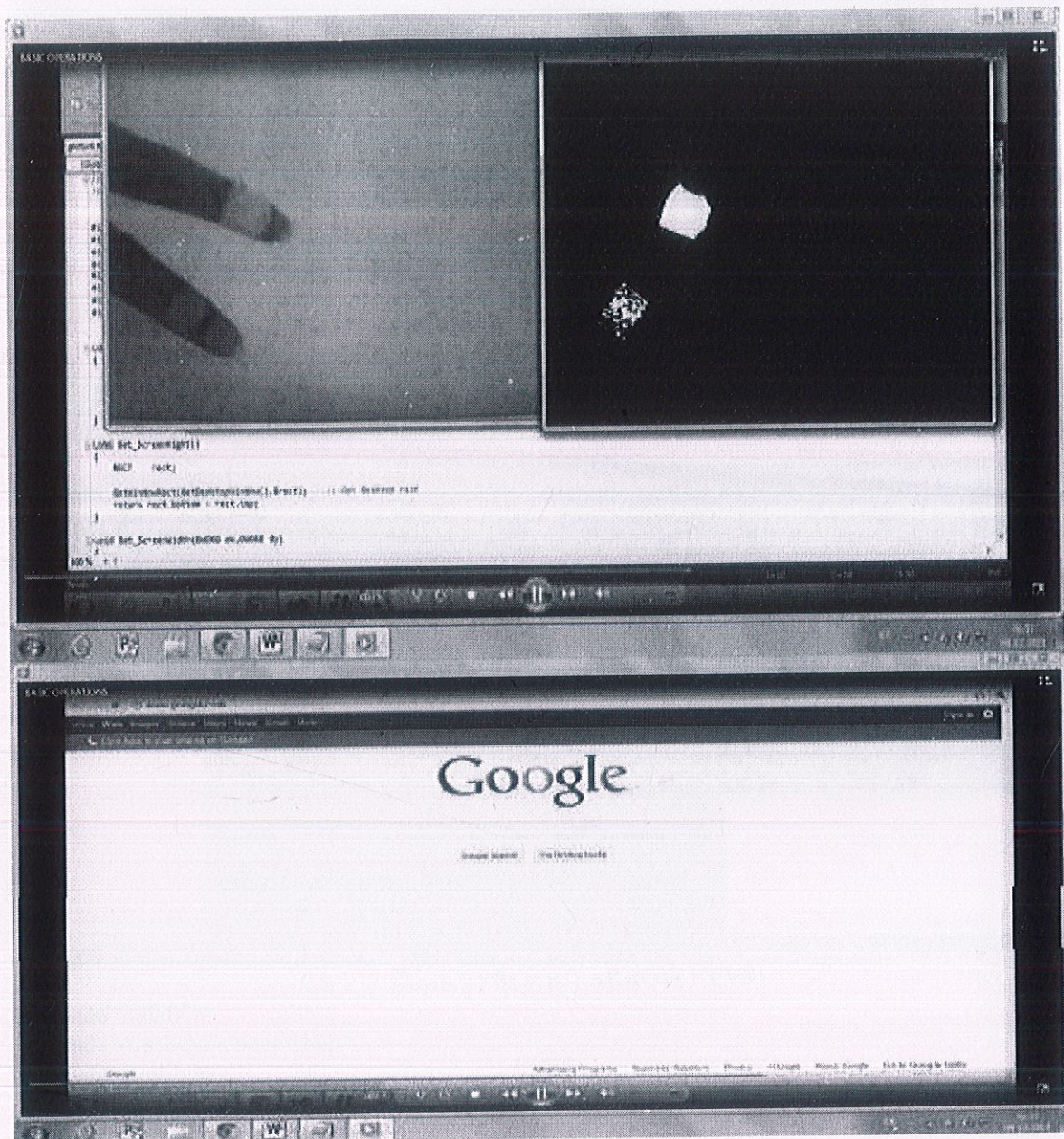
```

    }
    case 2: {
        (void)system("C://Users//AJEET//AppData//Local//Google//Chrome//Applic
        ation//chrome http://www.google.com");
        break;
    }
    case 3: {
        (void) system("calc");
        break;
    }
}
gest=0;
cvAdd(imgYellowThresh, imgBlueThresh, imgYellowThresh);
cvAdd(imgYellowThresh, imgPinkThresh, imgYellowThresh);
cvShowImage("ThreshWin", imgYellowThresh);
cvShowImage("video", frame);
int c = cvWaitKey(10);
if(c==27)
{
    break;
}
cvReleaseImage(&imgBlueThresh);
cvReleaseImage(&imgPinkThresh);
delete moments1;
delete moments2;
delete moments3;
}
cvReleaseCapture(&capture);
return 0;
}

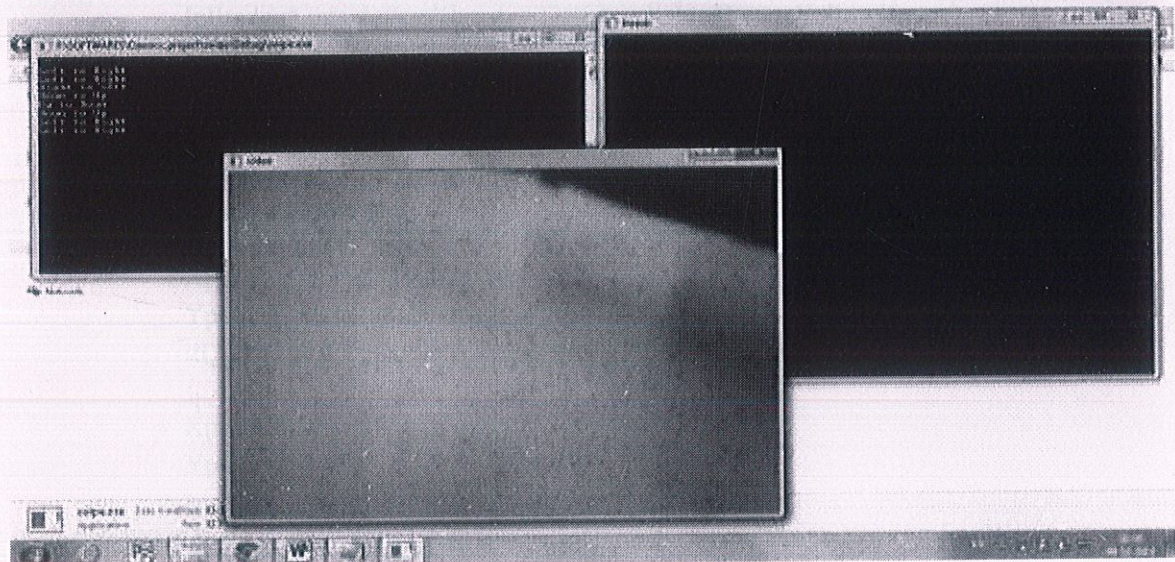
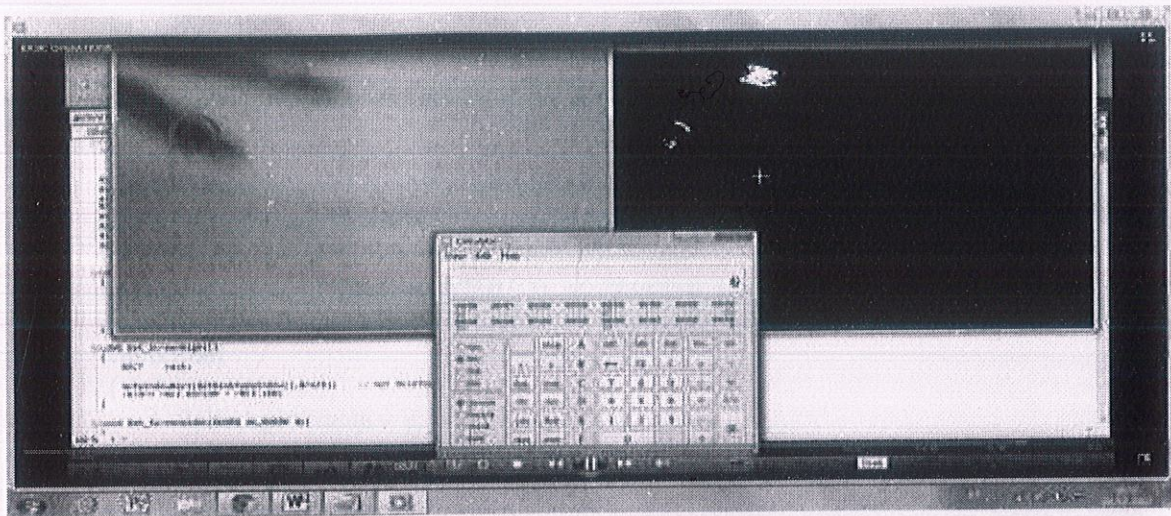
```











// SWIPING (KEYBOARD ARROW KEYS)

```
#include "stdafx.h"
#include <opencv2\opencv.hpp>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <time.h>
IplImage* GetThresholdedImageYellow(IplImage* img)
{
    // Same as in the above code
}
int main()
{
    CvCapture* capture = 0;
    capture = cvCaptureFromCAM(1);
```



```

if(!capture)
{
printf("Could not initialize capturing...\n");
return -1;
}
cvNamedWindow("video",CV_WINDOW_AUTOSIZE);
cvNamedWindow("ThreshWin",CV_WINDOW_AUTOSIZE);
time_t initial, finnal, initialp, finnalp, initial3, finnal3;
int flag1=0, gest=0, Xlast1=0, Ylast1=0, secXlast1=0, secYlast1=0;
double initime, fintime,dtime=0, initimep, fintimep,dtimep=0, initime3, fintime3,dtime3=0;
while(true)
{
    IplImage* frame = 0;
    frame = cvQueryFrame(capture);
    if(!frame)
        break;
    IplImage* imgYellowThresh = GetThresholdedImageYellow(frame);
    CvMoments *moments1 = (CvMoments*)malloc(sizeof(CvMoments));
    cvMoments(imgYellowThresh, moments1, 1);
    int moment10a = cvGetSpatialMoment(moments1, 1, 0);
    int moment01a = cvGetSpatialMoment(moments1, 0, 1);
    double area1 = cvGetCentralMoment(moments1, 0, 0);
    static int Xpos1 = 0;
    static int Ypos1 = 0;
    Xpos1 = (moment10a/area1);
    Ypos1 = (moment01a/area1);
    if(Xpos1<0 || Ypos1<0 || Xpos1>640 || Ypos1>480)
    {
        Xpos1=0;
        Ypos1=0;
    }
    if(area1>400 && secXlast1>Xlast1 && ((secXlast1-Xlast1)*(secXlast1-Xlast1)>(secYlast1-Ylast1)*(secYlast1-Ylast1)) && Xpos1!=0 && Ypos1!=0)
    {
        flag1=1; //HL
    }
    if(area1>400 && Xlast1>secXlast1 && ((secXlast1-Xlast1)*(secXlast1-Xlast1)>(secYlast1-Ylast1)*(secYlast1-Ylast1)) && Xpos1!=0 && Ypos1!=0)
    {
        flag1=2; //HR
    }
    if(area1>400 && secYlast1>Ylast1 && ((secYlast1-Ylast1)*(secYlast1-Ylast1)>(secXlast1-Xlast1)*(secXlast1-Xlast1)) && Xpos1!=0 && Ypos1!=0)
    {
        flag1=3; //VU
    }
    if(area1>400 && secYlast1<Ylast1 && ((secYlast1-Ylast1)*(secYlast1-Ylast1)>(secXlast1-Xlast1)*(secXlast1-Xlast1)) && Xpos1!=0 && Ypos1!=0)

```



```

{
    flag1=4; //VD
}
if(flag1==1 && Xpos1==0 && Ypos1==0)
{
    printf("\nRight to left");
    flag1=0;
    keybd_event(VK_RIGHT,0x27,0,0);
    keybd_event(VK_RIGHT,0x27,KEYEVENTF_KEYUP,0);
}
if(flag1==2 && Xpos1==0 && Ypos1==0)
{
    printf("\nLeft to Right");
    flag1=0;
    keybd_event(VK_LEFT,0x25,0,0);
    keybd_event(VK_LEFT,0x25,KEYEVENTF_KEYUP,0);
}
if(flag1==3 && Xpos1==0 && Ypos1==0)
{
    printf("\nDown to Up");
    flag1=0;
    keybd_event(VK_DOWN,0x28,0,0);
    keybd_event(VK_DOWN,0x28,KEYEVENTF_KEYUP,0);
}
if(flag1==4 && Xpos1==0 && Ypos1==0)
{
    printf("\nUp to Down");
    flag1=0;
    keybd_event(VK_UP,0x26,0,0);
    keybd_event(VK_UP,0x26,KEYEVENTF_KEYUP,0);
}

    secXlast1=Xlast1;
    secYlast1=Ylast1;
    Xlast1 = Xpos1;
    Ylast1 = Ypos1;
    switch(gest)
    {
        case 0: break;
        case 1: {
                    printf("\nDown");
                    break;
                }
        case 2: {

```



```

        printf("\nUp");
        break;
    }
    case 3: {
        printf("\nRight");
        break;
    }
    case 4: {
        printf("\nLeft");
        break;
    }
}

gest=0;
cvShowImage("ThreshWin", imgYellowThresh);
cvShowImage("video", frame);
int c = cvWaitKey(10);
if(c==27)
{
    break;
}
delete moments1;
}

cvReleaseCapture(&capture);
return 0;
}

```

## Chapter 9: Appendix

### Appendix A- Glossary

- **Artifact:** artificial defect in the image, due to problems in sensing equipment (scratches in Xray digitized films), or during examinations (patient motion)
- **Binary image:** image where pixels have only two values, generally 0 and 1
- **Brightness:** The gray level value of a pixel within an image that corresponds to energy intensity. The larger the gray level value, the greater the brightness.
- **Clustering:** concept of grouping data in classes based upon the similarity of the data
- **Compression:** removal of any redundant data that may be present within the image, to reduce amount of data to manipulate or store.
- **Contrast:** the amount of gray level variation within an image



- Digitizer: electronic circuit that converts analog or continuous signals into discrete or digital data
- Dilation: a morphological operation that enlarges the geometrical size of objects within an image
- Discrete convolution: process where 2 images are combined using a shift, multiply and add operation.
- Enhancement: algorithms and processes that improve an image based on subjective measures. The aim is to accentuate certain image features for display or for subsequent analysis.
- Erosion: morphological operation that reduces the geometrical size of objects within an image
- Feature: any of the properties that are characteristic of an image, from which a description, interpretation or understanding of the scene can be provided by a machine.
- Graylevel: value of gray from a black and white (monochrome) image
- Grayscale: range of gray shades, or gray levels corresponding to pixel values that a monochrome image incorporates
- Histogram: distribution of pixel graylevel values. A graph of number of pixels at each graylevel possible in an image.
- Mask: refers to a small image used to specify the area of operation to take place on a larger image in an algorithm
- Noise: degradation of image due to equipment (i.e. Sensor, camera misfocus), type of modality, motion, turbulence.
- Object boundaries: linked edges that characterize the shape of an object
- Pixel: slang for picture element, the smallest element if an image
- Quantization: range of values that a pixel can represent
- Region of interest (ROI): zone under study within the image (2D or 3D) representation: characterization of the quantity that each pixel represents. For example an image could represent the absorption characteristics of the body tissue (Xray imaging) or the temperature profile of a region (infrared imaging)
- Resolution: smallest feature (spatial) or graylevel value (quantization) that an image system can resolve.
- Restoration: algorithms or processes that attempt to remove a degradation (noise, blurring, and defocusing effects) based on an objective criterion
- Sampling: used to describe spatial resolution of an image
- Segmentation: separation of different objects in the image (for example by extracting their Boundaries).
- Slice: 2D image often described as part of a 3D volume
- Skeletonization: algorithm used to identify the central axis (skeleton) of an image object
- Structuring set: set of pixels used to describe the structuring function used in the morphological erosion and dilation
- Texture: structural patterns of surfaces of objects such as wood, grain, sand, grass, cloth. It generally refers to repetition of basic texture elements known as texels. A texel contains several pixels whose placement could be periodic or random. Texture may be coarse, fine, smooth,
- Granulated, regular, irregular, linear, etc...threshold: a value used to segment the graylevel values of an image into two different regions. Also called the binarization of an image.
- Voxel: 3D pixel

## Appendix B - Entire Gesture Set



**Basic Controls:**

Sr. No.	Objects Scanned	Direction of motion	Action performed
1.	1	Any	PowerPoint Presentation
2.	2	Any	Google Chrome Web Browser
3.	3	Any	Calculator

Table 1: Basic controls gestures and associated operations

**Keyboard Control:**

Sr. No.	Objects Scanned	Direction of motion	Action performed
1.	1	UP	Down Arrow Key Pressed
2.	1	RIGHT	Left Arrow Key Pressed
3.	1	LEFT	Right Arrow Key Pressed
4.	1	DOWN	Up Arrow Key Pressed

Table 2: Basic controls gestures and associated operations

**Mouse Control:**

Sr. No.	Objects Scanned	Direction of motion	Action performed
1.	1	UP	Mouse cursor moves upwards
2.	1	RIGHT	Mouse cursor moves towards right
3.	1	LEFT	Mouse cursor moves towards left
4.	1	DOWN	Mouse cursor moves downwards
5.	2	PINK OBJECT COMES IN FRAME FOR 1-2 SEC	Right Click
6.	2	YELLOW OBJECT COMES IN FRAME FOR 1-2 SEC	Left Click

Table 3: Basic controls gestures and associated operations