**Project Report On**

**Gesture Recognition System for Adobe PDF Viewer**

**Submitted By:**

Sharma Rishabh Rishi 111070036

Menezes Sachin Mike 1110700044

Pandhare Shripad Shankar 111070065

Nirgude Amey Sunil 1111070051

Doshi Janmejay Janak 1111070022

Bachelor of Technology, Computer Engineering

2014-2015

**Under The Guidance Of**

Prof. Varshapriya Jyotinagar



**Department of Computer Engineering And Information Technology**

**Veermata Jijabai Technological Institute**

(Autonomous Institute Affiliated To Mumbai University)

Matunga, Mumbai – 400019

**STATEMENT BY THE CANDIDATE**

We wish to state that the work embodied in this report titled “**Gesture Recognition System for Adobe PDF Viewer**” forms our group’s contribution to the work carried out under the guidance of **Prof. Varshapriya Jyotinagar** at **Veermata Jijabai Technological Institute**. This work has not been submitted for any Degree or Diploma of any University/Institute. Wherever references have been made to previous works of others, it has been clearly indicated.

**Mr. Menezes Sachin Mike Mr. Sharma Rishabh Rishi**

**Mr. Pandhare Shripad Shankar Mr. Nirgude Amey Sunil**

**Mr. Doshi Janmejay Doshi**



VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

(Autonomous Institute Affiliated To Mumbai University)

Matunga, Mumbai – 400019

**CERTIFICATE**

This is to certify that the following students have satisfactorily carried out work for the project entitled

**Gesture Recognition System for Adobe PDF Viewer**

in partial fulfilment of B.Tech in Computer Engineering of the University of Mumbai during the academic year 2011-2015

Name Id No

Mr. Rishabh Sharma 1110700036

Mr. Sachin Menezes 111070044

Mr. Shripad Pandhare 111070065

Mr. Amey Nirgude 111070051

Mr. Janmejay Doshi 111070022

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Internal Examiner External Examiner

**CERTIFICATE**

This is to certify that this student of B.Tech as a member of team of students has completed the project **“Gesture Recognition System for Adobe PDF Viewer”**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Mrs. Varshapriya Jyotinagar Dr. G.B. Bhole

Project Guide Head of Department

**ACKNOWLEDGMENT**

We would like to thank our project guide and mentor, **Prof. Varshapriya Jyotinagar** for contributing her time and effort to help us with our project. Her constant support and encouragement, along with numerous inputs and suggestions she gave, have been invaluable and have been a driving force which has constantly motivated us to explore different aspects of our project. She has always been involved by discussing our project at each phase to make sure that the project is designed and implemented in an appropriate manner and that our conclusions are appropriate.

**ABSTRACT**

With the ever-increasing diffusion of computers into the society, it is widely believed that present popular mode of interactions with computers (mouse and keyboard) will become a bottleneck in the effective utilization of information flow between the computers and the human. Vision based Gesture recognition has the potential to be a natural and powerful tool supporting efficient and intuitive interaction between the human and the computer. Visual interpretation of hand gestures can help in achieving the ease and naturalness desired for Human Computer Interaction (HCI). The task of gesture recognition is highly challenging due to complex background, presence of non-gesture hand motions, and different illumination environments.

The aim of this project is to develop interface that can be connect to digital world to physical world. Now people are used to keyboard and mouse to get daily information from internet such as daily news, financial news, weather etc. To improve these situations proposed system user can interact with system by using hand gesture. System can give hand gesture as input. Once system get particular hand gesture as input, the system will report information from internet as a output from system to user. It takes video stream as input and system having predefined hand gestures. When it recognizes the gesture, system stimulates the appropriate action associated with particular gesture.

**TABLE OF CONTENTS**

1. Introduction

1.1. Background and Motivation

1.2. Approaches to Gesture Recognition

1.3. Requirements

1.4. Application Domains

1.5. Problem Statement

2. Literature Review

2.1. Computer Vision and Gesture Recognition

2.2. Existing Algorithms

2.3. OpenCV Library

2.4. Accuracy Issues in Gesture Recognition

3. Proposed Solution

4. System Analysis and Design

3.1. Use Case Diagram

3.2. Activity Diagram

3.3. DFD Diagram

3.4. Class Diagram

3.5. Flowchart

4. Implementation

4.1. Software tools

4.2. Setting up OpenCV

4.3. Configuring OpenCV with Visual Studio

4.4. Functions Implemented

4.5 Source Code

5. Conclusion

5.1. Conclusion

5.2. Future work

**1. INTRODUCTION**

**1.1 Background and motivation**

With the ever-increasing diffusion of computers into the society, it is widely believed that present popular mode of interactions with computers (mouse and keyboard) will become a bottleneck in the effective utilization of information flow between the computers and the human. Vision based Gesture recognition has the potential to be a natural and powerful tool supporting efficient and intuitive interaction between the human and the computer. Visual interpretation of hand gestures can help in achieving the ease and naturalness desired for Human Computer Interaction (HCI).

With the development of information technology in our society, we can expect that computer systems to a larger extent will be embedded into our environment. These environments will impose needs for new types of human computer-interaction, with interfaces that are natural and easy to use.

The user interface (UI) of the personal computer has evolved from a text-based command line to a graphical interface with keyboard and mouse inputs. However, they are inconvenient and unnatural. The use of hand gestures provides an attractive alternative to these cumbersome interface devices for human-computer interaction (HCI). User’s generally use hand gestures for expression of their feelings and notifications of their thoughts. In particular, visual interpretation of hand gestures can help in achieving the ease and naturalness desired for HCI. Vision has the potential of carrying a wealth of information in a nonintrusive manner and at a low cost, therefore it constitutes a very attractive sensing modality for developing hand gestures recognition. Recent researches in computer vision have established the importance of gesture recognition systems for the purpose of human computer interaction. The primary goal of gesture recognition research is to create a system which can identify specific human gestures and use them to convey information or for device control. A gesture may be defined as a physical movement of the hands, arms, face, and body with the intent to convey information or meaning. Gesture recognition, then, consists not only of the tracking of human movement, but also the interpretation of that movement as semantically meaningful commands.

There are already various systems in place which use gesture recognition technology for playing games on consoles such as wii. But there is no suitable system which uses gesture recognition techniques for functions in text documents. Hence, we came up with the idea to create such a system that individuals can use while browsing or reading through a text document.

**1.2 Approaches to Gesture recognition**

Two approaches are commonly used to interpret gestures for Human Computer interaction.

They are

1. Methods which Use Data Gloves: This method employs sensors (mechanical or optical) attached to a glove that transduces finger flexions into electrical signals for determining the hand posture. This approach forces the user to carry a load of cables which are connected to the computer and hinders the ease and naturalness of the user interaction.
2. Methods which are Vision Based: Computer vision based techniques are non invasive and based on the way human beings perceive information about their surroundings. Although it is difficult to design a vision based interface for generic usage, yet it is feasible to design such an interface for a controlled environment.

**Classification of Hand Gestures**

Hand Gestures can be classified using the following two approaches.

1. Rule based Approaches: Rule-based approaches consist of a set of manually encoded rules between feature inputs. Given an input gesture a set of features are extracted and compared to the encoded rules, the rule that matches the input is outputted as the gesture. As an example, in [23, 24] predicates related to low-level features of the motion of the hands are defined for each of the actions under consideration. When a predicate of a gesture is satisfied over a fixed number of consecutive frames the gesture is returned. A major problem with rule based approaches is that they rely on the ability of a human to encode rules.
2. Machine Learning based Approaches: A popular machine learning approach is to treat a gesture as the out-put of a stochastic process. Of this class of approach Hidden Markov Models (HMMs) by far have received the most attention in the literature for classifying gestures.

**1.3 Requirements**

Vision-based interaction is a challenging interdisciplinary research area, which involves computer Vision and graphics, image processing, machine learning, bio-informatics, and psychology. To make a successful working system, there are some requirements which the system should have:

1. Robustness: In the real-world, visual information could be very rich, noisy, and incomplete, due to changing illumination, clutter and dynamic backgrounds, occlusion, etc. Vision-based systems should be user independent and robust against all these factors.
2. Computational Efficiency: Generally, Vision based interaction often requires real-time systems. The vision and learning techniques/algorithms used in Vision-based interaction should be effective as well as cost efficient.
3. User’s Tolerance: The malfunctions or mistakes of Vision-based interaction should be tolerated. When a mistake is made, it should not incur much loss. Users can be asked to repeat some actions, instead of letting the computer make more wrong decisions.
4. Scalability: The Vision-based interaction system should be easily adapted to different scales of applications. For eg. the core of Vision-based interaction should be the same for desktop environments, Sign Language Recognition, robot navigation and also for VE.

Most of the systems reviewed rely on the simple idea of detecting and segmenting the gesturing hand from the background using motion detection or skin color. Proper selection of features or clues, and their combination with sophisticated recognition algorithms, can affect the success or failure of any existing and future work in the field of Human Computer interaction using hand gestures.

**1.4 Application Domains**

In this section, as the gesture recognition can be used in many more areas, we present an overview of the some of the application domains that employ gesture interactions.

1. Virtual Reality: Gestures for virtual and augmented reality applications have experienced one of the greatest levels of uptake in computing. Virtual reality interactions use gestures to enable realistic manipulations of virtual objects using ones hands, for 3D display interactions [4] or 2D displays that simulate 3D interactions.
2. Robotics and Telepresence: Telepresence and telerobotic applications are typically situated within the domain of space exploration and military-based research projects. The gestures used to interact with and control robots are similar to fully-immersed virtual reality interactions, however the worlds are often real, presenting the operator with video feed from cameras located on the robot [6]. Here, gestures can control a robots hand and arm movements to reach for and manipulate actual objects, as well its movement through the world.
3. Desktop and Tablet PC Applications: In desktop computing applications, gestures can provide an alternative interaction to the mouse and keyboard [7]. Many gestures for desktop computing tasks involve manipulating graphics, or annotating and editing documents using pen-based gestures.
4. Games: When, we look at gestures for computer games. Freeman et al. [9] tracked a player’s hand or body position to control movement and orientation of interactive game objects such as cars. Konrad et al. [10] used gestures to control the movement of avatars in a virtual world, and Play Station 2 has introduced the Eye Toy, a camera that tracks hand movements for interactive games.
5. Sign Language: Sign language is an important case of communicative gestures. Since sign languages are highly structural, they are very suitable as testbeds for vision algorithms [12]. At the same time, they can also be a good way to help the disabled to interact with computers. Sign language for the deaf (e.g. American Sign Language) is an example that has received significant attention in the gesture literature.

**1.5 Problem Statement**

There are many gesture recognition system technologies that are in use in the current world. Some are used for playing games on the console such as wii. Other applications include using gestures to understand sign language and providing a better interaction experience with the machine. However, there is no suitable system which uses gesture recognition techniques for functions in text documents while reading a novel or so. Therefore, we came up with the idea to build such a system that individuals can use while browsing or reading through a text document.

Our project makes use of the standard webcam of a laptop or even an external webcam to recognize the gestures. The gesture recognition system is used to capture the hand gesture through the webcam. Once the image of the hand is captured, various image processing techniques are used. These techniques are used to convert the image to a HSV (Hue, Saturation, value) image which filters out the disturbances and enables more accurate processing.

The application we are interfacing our gesture recognition system is the Adobe PDF Viewer. Each gesture is mapped to an appropriate action that can be performed in the PDF application. Various actions such as opening, closing, scrolling, printing, etc. can be performed using our gesture recognition system.  
  
This will enable an individual using the system to perform other tasks such as eating and read, scroll, close a document and so on. This achieves our goal of improving the interaction between man and machine as well reduce the need for peripheral devices, improve multi-tasking and reduce human efforts.

**2. LITERATURE REVIEW**

**2.1 Computer Vision & Gesture recognition**

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions. A theme in the development of this field has been to duplicate the abilities of human vision by electronically perceiving and understanding an image.[[5]](http://en.wikipedia.org/wiki/Computer_vision#cite_note-Sonka-Hlavac-Boyle-2008-5) This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.[[6]](http://en.wikipedia.org/wiki/Computer_vision#cite_note-Forsyth-Ponce-2003-6) Computer vision has also been described as the enterprise of automating and integrating a wide range of processes and representations for vision perception.

As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner. As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems.

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

The classical problem in computer vision, image processing, and machine vision is that of determining whether or not the image data contains some specific object, feature, or activity. Different varieties of the recognition problem are described in the literature:

* Object recognition (also called object classification) – one or several pre-specified or learned objects or object classes can be recognized, usually together with their 2D positions in the image or 3D poses in the scene. Google goggles or LikeThat provide a stand-alone programs that illustrate this function.
* Identification – an individual instance of an object is recognized. Examples include identification of a specific person's face or fingerprint, hands or eyes, identification of handwritten digits, or identification of a specific vehicle.
* Detection – the image data are scanned for a specific condition. Examples include detection of possible abnormal cells or tissues in medical images or detection of a vehicle in an automatic road toll system. Detection based on relatively simple and fast computations is sometimes used for finding smaller regions of interesting image data which can be further analyzed by more computationally demanding techniques to produce a correct interpretation.

Gesture recognition is a topic in computer science and language technology with the goal of interpreting human gestures via mathematical algorithms. Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Current focuses in the field include emotion recognition from face and hand gesture recognition. Many approaches have been made using cameras and computer vision algorithms to interpret sign language. However, the identification and recognition of posture, gait, proxemics, and human behaviors is also the subject of gesture recognition techniques. Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans than primitive text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse.

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

Gesture recognition can be conducted with techniques from computer vision and image processing.

The literature includes ongoing work in the computer vision field on capturing gestures or more general human pose and movements by cameras connected to a computer.

**2.2 Existing algorithms**

**2.2.1 Hidden Markov Models**

This method (Hidden Markov Model) deals with the dynamic aspects of gestures. Gestures are extracted from a sequence of video images by tracking the skin-color blobs corresponding to the hand into a body– face space centered on the face of the user. The goal is to recognize two classes of gestures: deictic and symbolic. The image is filtered using a fast look–up indexing table of skin colour pixels in YUV colour space. After filtering, skin colour pixels are gathered into blobs. Blobs are statistical objects based on the location (x, y) and the colourimetry (Y,U,V) of the skin color pixels in order to determine homogeneous areas. A skin colour pixel belongs to the blob which has the same location and colourimetry component.

**2.2.2 YUV Colour Space and CAMSHIFT Algorithm**

This method deals with recognition of hand gestures. It is done in the following five steps.

1. First, a digital camera records a video stream of hand gestures.

2. All the frames are taken into consideration and then using YUV colour space skin colour based segmentation is performed. The YUV colour system is employed for separating chrominance and intensity. The symbol Y indicates intensity while UV specifies chrominance components.

3. Now the hand is separated using CAMSHIFT [2] algorithm .Since the hand is the largest connected region, we can segment the hand from the body.

4. After this is done, the position of the hand centroid is calculated in each frame. This is done by first calculating the zeroth and first moments and then using this information the centroid is calculated.

5. Now the different centroid points are joint to form a trajectory .This trajectory shows the path of the hand movement and thus the hand tracking procedure is determined.

**2.2.3 Using Time Flight Camera**

This approach uses x and y-projections of the image and optional depth features for gesture classification. The system uses a 3-D time-of-flight (TOF) [3] [4] sensor which has the big advantage of simplifying hand segmentation. The gestures used in the system show a good separation potential along the two image axes. Hence, the projections of the hand onto the x- and y-axis are used as features for the classification. The region of the arm is discarded since it contains no useful information for the classification and due to strong variation between human beings. Additionally, depth features are included to distinguish certain gestures: gestures which have same projections, but different alignments.

The algorithm can be divided into five steps:

1. Segmentation of the hand and arm via distance values: The hand and arm are segmented by an iterative seed fill algorithm.

2. Determination of the bounding box: The segmented region is projected onto the x- and y-axis to determine the bounding box of the object.

3. Extraction of the hand.

4. Projection of the hand region onto the x- and y-axis.

**2.2.4 Naïve Bayes’ Classifier**

This method is an effective and fast method for static hand gesture recognition. This method is based on classifying the different gestures according to geometric-based invariants which are obtained from image data after segmentation; thus, unlike many other recognition methods, this method is not dependent on skin colour. Gestures are extracted from each frame of the video, with a static background. The segmentation is done by dynamic extraction of background pixels according to the histogram of each image. Gestures are classified using a weighted K-Nearest Neighbors Algorithm which is combined with a Naïve Bayes [5] approach to estimate the probability of each gesture type. When this method was tested in the domain of the JAST Human Robot dialog system, it classified more than 93% of the gestures correctly.

This algorithm proceeds in three main steps. The first step is to segment and label the objects of interest and to extract geometric invariants from them. Next, the gestures are classified using a K-nearest neighbor algorithm with distance weighting algorithm (KNNDW) to provide suitable data for a locally weighted Naïve Bayes’ classifier. The input vector for this classifier consists of invariants of each region of interest, while the output is the type of the gesture. After the gesture has been classified, the final step is to locate the specific properties of the gesture that are needed for processing in the system—for example, the fingertip for a pointing gesture or the center of the hand for a holding-out gesture.

**2.2.5 Vision Based Hand Gesture Recognition**

In vision based hand gesture recognition system, the movement of the hand is recorded by video camera(s). This input video is decomposed into a set of features taking individual frames into account. The hands are isolated from other body parts as well as other background objects. The isolated hands are recognized for different postures. Since, gestures are nothing but a sequence of hand postures connected by continuous motions, a recognizer can be trained against a possible grammar. With this, hand gestures can be specified as building up out of a group of hand postures in various ways of composition, just as phrases are built up by words. The recognized gestures can be used to drive a variety of applications.

The approaches to Vision based hand posture and gesture recognition are:

(i) 3D hand model based approach

(ii) Appearance based approach

**3D Hand Model Based Approach**

Three dimensional hand model based approaches rely on the 3D kinematic hand model with considerable DOF’s, and try to estimate the hand parameters by comparison between the input images and the possible 2D appearance projected by the 3D hand model. Such an approach is ideal for realistic interactions in virtual environments. This approach has several disadvantages that have kept it from real-world use. First, at each frame the initial parameters have to be close to the solution, otherwise the approach is liable to find a suboptimal solution (i.e. local minima). Secondly, the fitting process is also sensitive to noise (e.g. lens aberrations, sensor noise) in the imaging process. Finally, the approach cannot handle the inevitable self-occlusion of the hand.

**Appearance Based Approach**

This method use image features to model the visual appearance of the hand and compare these parameters with the extracted image features from the video input. Generally speaking, appearance based approaches have the advantage of real time performance due to the easier 2D image features that are employed. There have been a number of research efforts on appearance based methods in recent years. A straightforward and simple approach that is often utilized is to look for skin colored regions in the image. Although very popular, this has some drawbacks like skin colour detection is very sensitive to lighting conditions. While practicable and efficient methods exist for skin colour detection under controlled (and known) illumination, the problem of learning a flexible skin model and adapting it over time is challenging. This only works if we assume that no other skin like objects is present in the scene. Another approach is to use the eigenspace for providing an efficient representation of a large set of high-dimensional points using a small set of basis vectors.

**2.3 OpenCV Library**

OpenCV (*Open Source Computer Vision*) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel Russia research center in Nizhny Novgorod, and now supported by Willow Garage and Itseez. It is free for use under the open-source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.  Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

* **core** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
* **imgproc** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
* **video** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
* **calib3d** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
* **features2d** - salient feature detectors, descriptors, and descriptor matchers.
* **objdetect** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
* **highgui** - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.
* **gpu** - GPU-accelerated algorithms from different OpenCV modules.

**2.4 Accuracy Issues in Gesture Recognition**

* Computers are limited when it comes to understanding scenes, as they lack the ability to analyze the world around them. Key problems that computers have in understanding scenes include segmentation, object representation, machine learning and recognition. Because computers are limited by their 2D representation of scenes, a gesture recognition system has to apply various cues to acquire more accurate results and more valuable information. While the possibilities include whole-body tracking and other techniques that combine multiple cues, it is difficult to sense scenes using only 2D representation that do not include known 3D models of objects that they identify, such as human hands, bodies or faces.
* Depth information, or “z,” enables capabilities well beyond gesture recognition. The challenge in incorporating 3D vision and gesture recognition into technology has been obtaining this third “z” coordinate. The human eye naturally registers x, y and z coordinates for everything it sees, and the brain then interprets those coordinates into a 3D image. In the past, lack of image analysis technology prevented electronics from seeing in 3D. Today, there are three common technologies that can acquire 3D images, each with its own unique strengths and common use cases: stereoscopic vision, structured light pattern and time of flight (TOF). With the analysis of the 3D image output from these technologies, gesture-recognition technology becomes a reality.
* There are many challenges associated with the accuracy and usefulness of gesture recognition software. For image-based gesture recognition there are limitations on the equipment used and image noise. Images or video may not be under consistent lighting, or in the same location. Items in the background or distinct features of the users may make recognition more difficult.
* The variety of implementations for image-based gesture recognition may also cause issue for viability of the technology to general usage. For example, an algorithm calibrated for one camera may not work for a different camera. The amount of background noise also causes tracking and recognition difficulties, especially when occlusions (partial and full) occur. Furthermore, the distance from the camera, and the camera's resolution and quality, also cause variations in recognition accuracy.
* In order to capture human gestures by visual sensors, robust computer vision methods are also required, for example for hand tracking and hand posture recognition or for capturing movements of the head, facial expressions or gaze direction.

**3. PROPOSED SOLUTION**

**3.1 Solution**

Firstly, the aim of our project is to enforce user interaction with the applications without physically pressing the buttons or the keys or using peripheral devices such as mouse or keyboard. The application can find use in an event where distant objects are to be remotely controlled or interacting with different software and stimulating an action on detection of motion of specific objects. The scope of the project is too vast to an extent that the use of physical devices such as a mouse, keypad, keyboard to control the application can eliminated as the same work can be done by gestures like a swipe of the hand or just an eye wink! Gesture recognition is a topic in [computer science](http://en.wikipedia.org/wiki/Computer_science) and [language technology](http://en.wikipedia.org/wiki/Language_technology) with the goal of interpreting human [gestures](http://en.wikipedia.org/wiki/Gesture) via mathematical [algorithms](http://en.wikipedia.org/wiki/Algorithms). Gestures can originate from any bodily motion or state but commonly originate from the [face](http://en.wikipedia.org/wiki/Face) or [hand](http://en.wikipedia.org/wiki/Hand). Current focuses in the field include emotion recognition from face and hand gesture recognition. Many approaches have been made using cameras and [computer vision](http://en.wikipedia.org/wiki/Computer_vision) algorithms to interpret [sign language](http://en.wikipedia.org/wiki/Sign_language). We achieved the same by the method of gesture recognition where the user interacts with the application by hand movements that are recorded by the web camera. The gesture recognition system that we developed would continuously process the images captured by the web camera to find a matching pre-defined gesture that interacts with the application in the desired way.

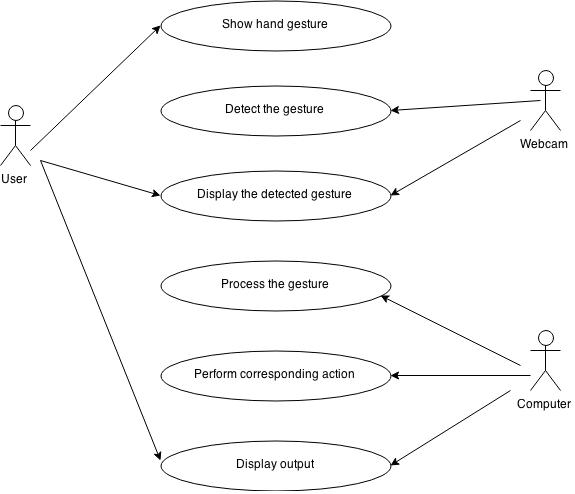
We researched that the best way to achieve the hand gesture recognition system was to use the OpenCV library in C++. We began by going through the OpenCV documentation which gave us insight into how images and videos can be processed by a program. OpenCV Library has multitudes of functions to process the videos, images, transforming the image configurations for removing external interferences, and improving the clarity of images for faster and accurate processing. As our project needs processing on images and videos, we started implementing simple programs to capture images and stream videos from the internal webcam in a laptop or an external webcam. Once, we were able to do this, we started researching about the various methodologies to process the image by capturing each frame of the video so as to detect different gestures of hands and invoke specific actions assigned to specific hand gestures. Each frame of the streamed videos displayed through webcam was processed for identifying possible hand movements.

The first method that we employed detected static hand gestures that demanded for a restricted background. Restricted background means that the background should be constant throughout the execution of the software code. In this method, the code captures an initial snapshot of the environment. All the subsequent captured frames are then compared to the initial snapshot. It is imperative that the background remains the same so that hand movement is detected appropriately. The difference in the frames are highlighted by a white color. The static part of the frame continues to remain black. A contour is then drawn around largest possible continuous area of the white color and is assumed to be the hand. After borders are drawn around the white area, convex defects are searched in the frame. These defects give an exact estimate of the number of fingers that are upright and hence, the type of the gesture delineated can be deduced. The con of this method is that the largest possible white space is not necessarily the hand and assuming the contiguous white space with maximum area as the hand can be flawed as the difference between the initially captured static environment and the current frame could be a “non-hand” area. Another drawback of the method is that the initial background should always be the same so that the difference taken remains consistent and detects new objects in that environment. Hence, the approach to detect the hand was revised.

In the revised approach, blue colored pixels were highlighted in each frame and the non-blue colored pixels were filtered off by considering a tight range of pixels to be analyzed. OpenCV library has implicit functions for specifying the range of pixels within the image to be considered. Post filtering, the processed images was again transformed to a black and white binary image where in the blue colored pixels were represented by white color and those non blue ones were made black. Now, borders are drawn along the white portion of the image with the maximum area and assumed to be the hand. Like the previous approach, with the aid of OpenCV library, contours are drawn along the convex defects of the bordered portion of the image and the number of contours drawn determines the number of fingers. Based on the angle between the fingers and the number of the fingers determines the action to be invoked, which could be opening, closing, scrolling up and down, or printing the pdf file. This technique is independent of the background and hence the restriction that was laid down in the 1st approach was eliminated. Therefore, this approach can be used in portable devices where the environment is altering during the software execution. Hence, this image was then compared to the difference image and it was found that the accuracy of detecting the hand improved drastically as the white colors which are non-blue colored were ruled out.

**3. SYSTEM ANALYSIS AND DESIGN**

**4.1 Use Case diagram**

****

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. Use case diagrams can provide a high-level view of the system and act as a blueprint for the entire system.

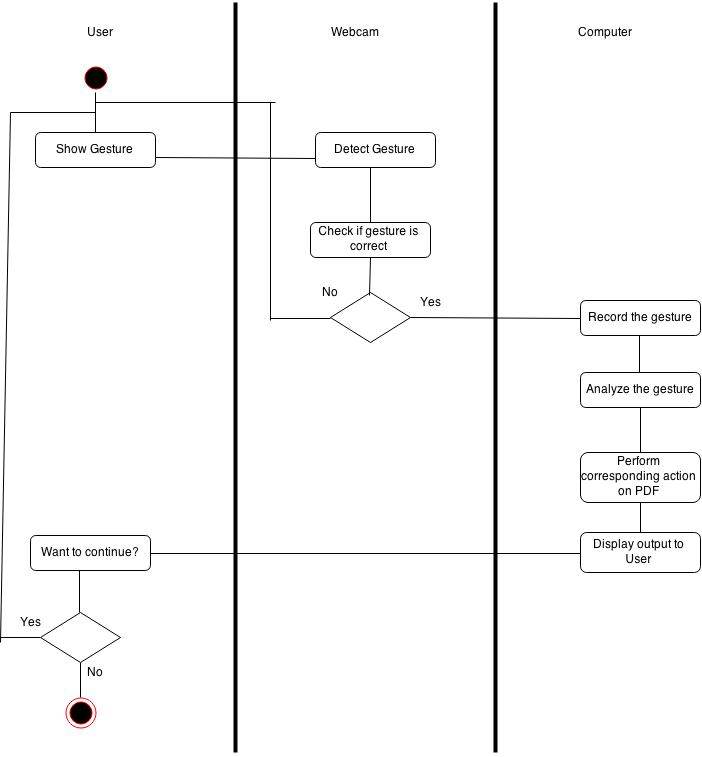
Use case: Show hand gesture  
Actors: User  
Purpose: To give a gesture to the webcam for processing.

Use case: Detect gesture  
Actors: Webcam  
Purpose: The webcam is used to detect and store the gesture

Use case: Perform corresponding action  
Actors: Computer   
Purpose: The computer program performs the corresponding action such as scrolling or opening the PDF application based on the gesture.

Use case: Display output  
Actors: User, Computer  
Purpose: The action is performed and the user is made aware of the result.

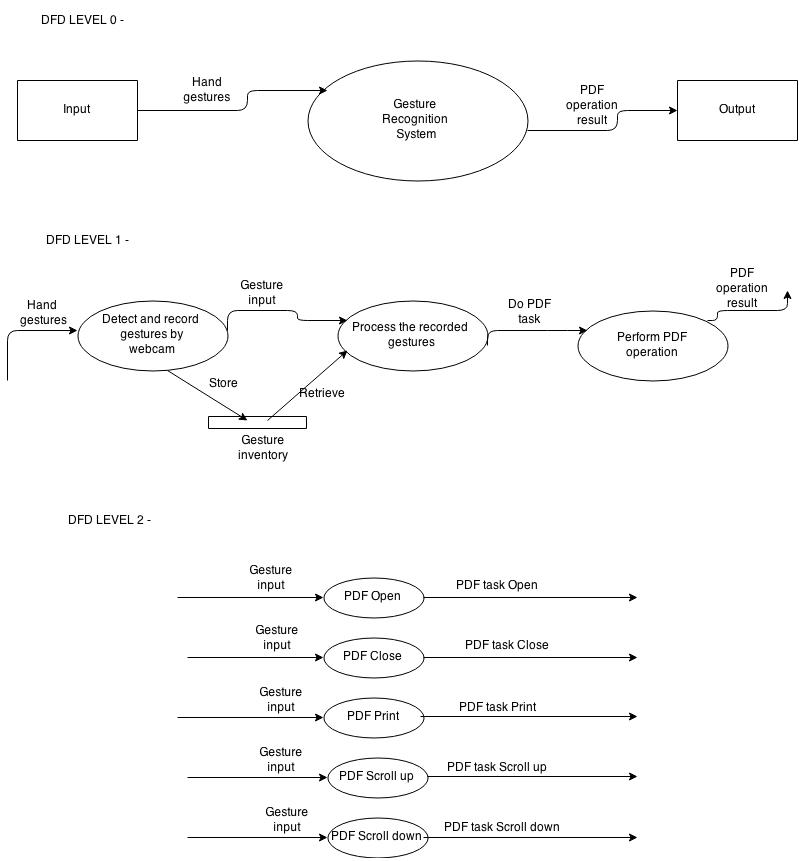
**4.1 Activity diagram**

****

Activity diagrams are graphical representations of [workflows](http://en.wikipedia.org/wiki/Workflow) of stepwise activities and actions with support for choice, iteration and concurrency. In the [Unified Modelling Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language), activity diagrams are intended to model both computational and organizational processes. Activity diagrams show the overall flow of control Activity diagrams are constructed from a limited number of shapes, connected with arrows.

The initial state of workflow begins from the user’s end, where the specific gesture is captured in the web cam. Every gesture is associated with a specific task in the application. In the next step the gesture shown by the user is cross checked for the assigned task in the database if match is found the corresponding activity is performed for the valid gesture captured the work flow then goes according to the steps which are pre-defined. The user is provided with the option of ending the task by a particular gesture unless that gesture is encountered the code is active in background.

**4.3 DFD**

****

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its *process* aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing.

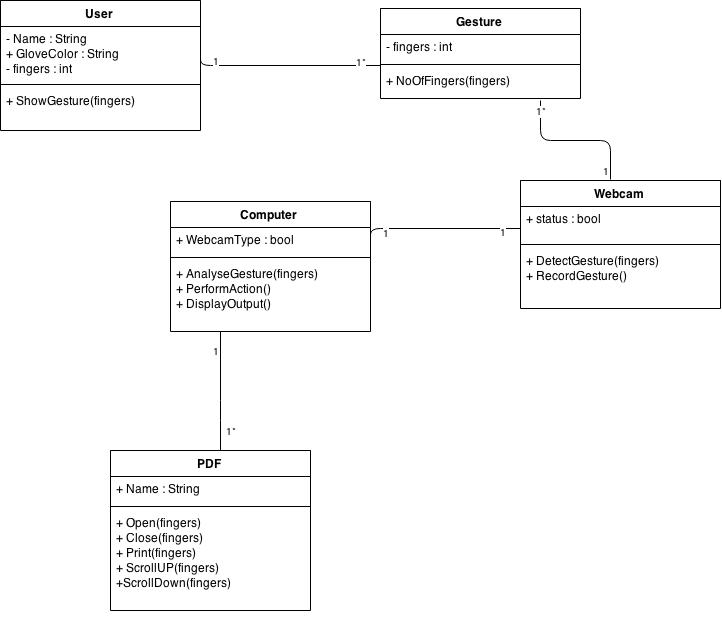
A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel.

The level 0 DFD shows the input to the Gesture Recognition System which are the hand gestures from the users. The processing is done and specific tasks are performed in the PDF application for the gesture specified.

Level 1 DFD shows three modules where the first module detects and records hand gestures which are provided as input in the first module. In the second module the hand gesture is processed and particular task assigned for a gesture is performed in the PDF application.

Level 2 DFD is a detailed breakdown of the tasks assigned in the PDF application like PDF open tasks etc.

**4.4 Class Diagram**

****

The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

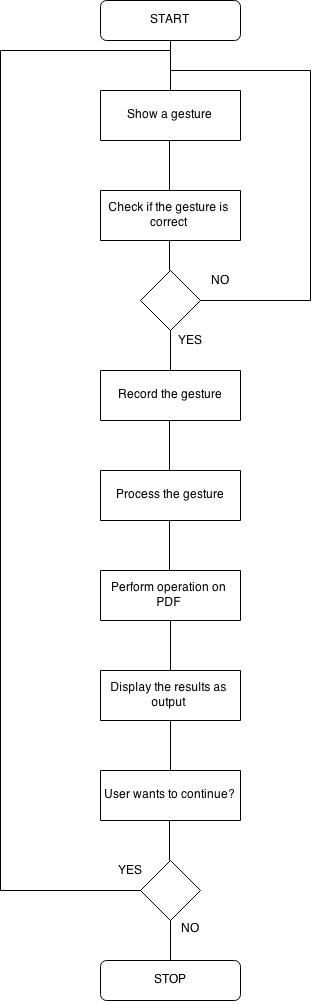
The package comprises of 5 classes

* User
* Gesture
* Computer
* Web cam
* PDF

The visibility of the member function i.e. fingers is kept private and also glove colour is taken into consideration for further processing. The cardinality is set to 1\* so that the no of gestures captured can be either 1 or zero. Thus multiple gestures are restricted at a single moment. The class computer has member functions to analyze and detect gestures. The class computer is associated with the PDF.

Class PDF has tasks associated with every gesture some of which are stated as open, close, scroll-up, scroll-down. The cardinality associated from the computer class is also 1\* itself as the task associated for a gesture is singular.

**4.5 Flowchart**

****

**3. IMPLEMENTATION**

**5.1 Software/Hardware tools used**

* Visual Studio 2013: Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web applications and web services. Visual Studio supports different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists.
* External/Internal Webcam: A webcam is a video camera that feeds or streams its image in real time to or through a computer to computer network. A webcam is generally connected by a USB cable, or similar cable, or built into computer hardware, such as laptops.
* Processor: Intel(r) core(tm) i5-3230m CPU @ 2.60ghz
* Installed Memory (RAM): 4.00GB
* System type: 64-bit operating system
* Languages used: C++
* Blue Glove: We have used a blue colored glove for easy detection of the hand which results in minimum interference. Plain hands are not being used since the skin color of each user varies which leads to inaccuracy in the detection of the hand. Also, some brown objects are also detected as skin

**5.2 Setting up OpenCV library on Visual Studio**

The most suitable way is the installation by using the pre-built libraries. So, we have installed OpenCV by using pre-built libraries.

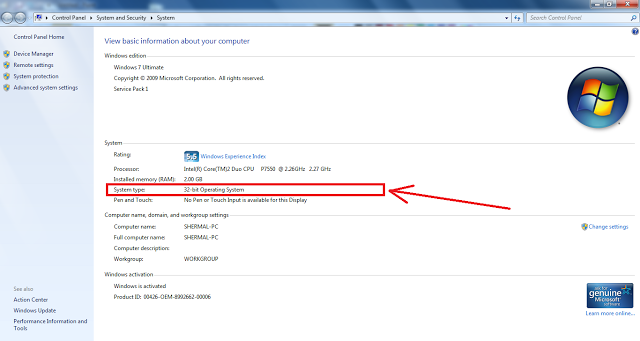
We have installed Visual Studio Express 2013 which is a free edition which can be easily downloaded from the official Microsoft visual studio website.

The next step is to download OpenCV from the official <http://opencv.org/downloads.html> website. Once downloaded, double click the downloaded 'OpenCV.exe' file and extract it to a particular location preferably C:\ as the location.

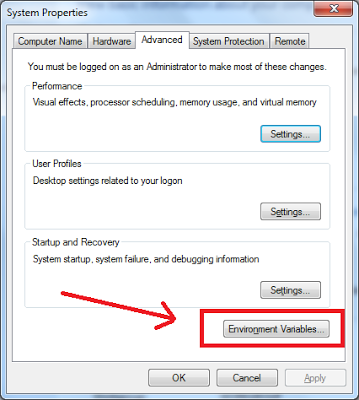
After this the environment variable needs to be set up to use OpenCV correctly.

**Steps to set up environment variable**

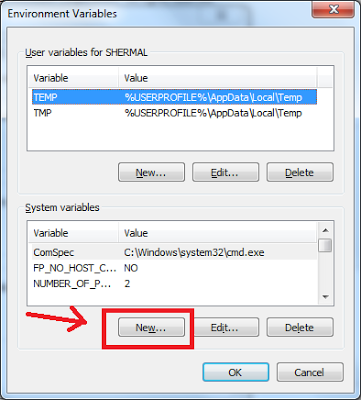
* Right click on '**My Computer**' and click '**Properties**' in the drop down menu. You will see a window like this.



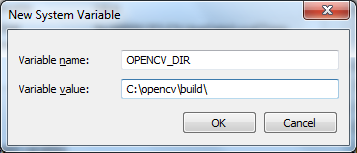
* Now you are ready to set the environment variables. This can be done in the command line or by using GUI. I am going to explain the GUI method.
* Right click on '**My Computer**' and click '**Properties**' in the drop down menu. You will see a window like this.
* Click '**Advance System Settings'**in the above window
* Then click **'Environment Variables'**



* Then click **'New'**button at the bottom of the window



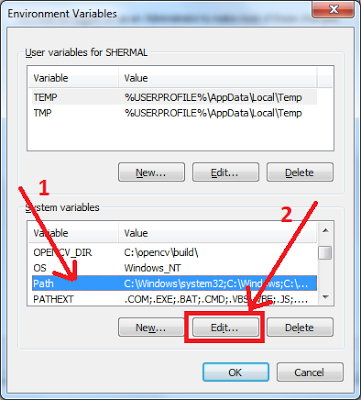
* Type **OPENCV\_DIR**against **Variable name** And type the location **C:\opencv\build\** against the **Variable value**



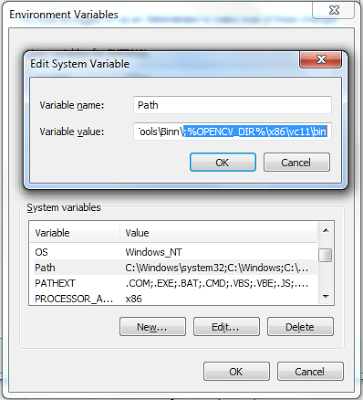
* Then press **OK**

Now you have added a new environment variable to the system.

* Click **'Path' or 'PATH'**inside the list of System variables and thenclick **'Edit'**button at the bottom of the window



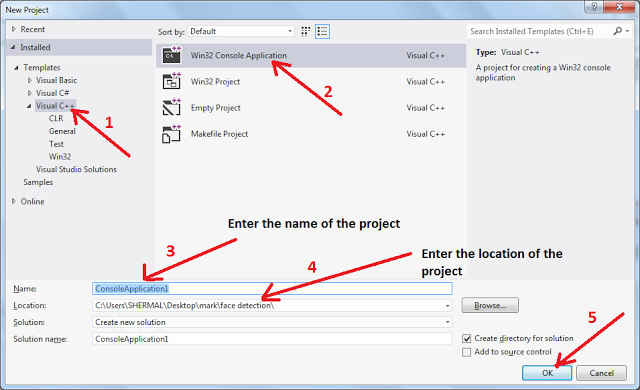
* Then add a ; to the end of the line and following %OPENCV\_DIR%\x86\vc11\bin  after the semicolon. X86 is your system architecture and vc11 is the compiler type. How to find your system architecture and compiler type is explained at the beginning of the post.



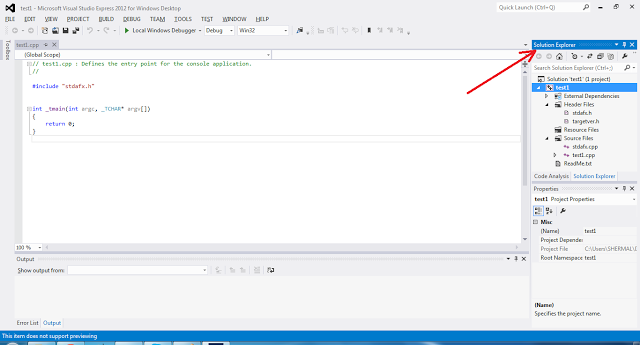
* This completes the configuration of OpenCV.

**5.3 Configuring Visual Studio**

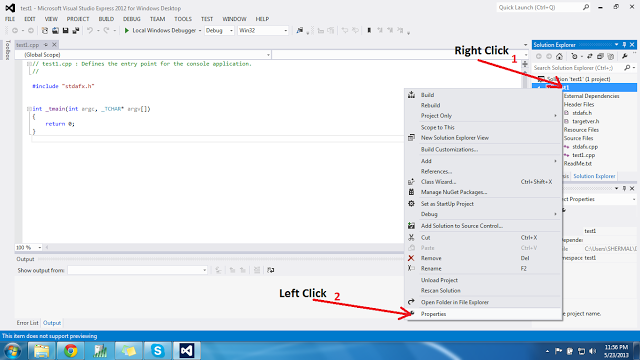
* Start Microsoft Visual Studio
* Go to File>New Project
* Do everything as shown in the below image and click OK and then click Finish.



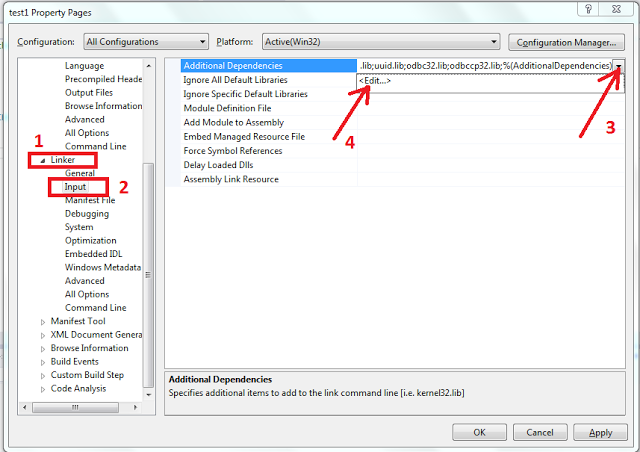
* Press 'Ctrl' key and then 'W' while holding the 'Ctrl' key in your keyboard. Release both keys. Then press 'S' in your keyboard. Then you will see the 'Solution Explorer' pane in your right or left side of the Visual Studio window.



* Then right click on the project name (you have entered this name in a previous step) and then click 'Properties'

****

* Click as illustrated below



* After you click the <Edit...> (4th arrow in the above image), dialog box will appear and you have to copy and paste following library file names.

opencv\_calib3d245d.lib

opencv\_contrib245d.lib

opencv\_core245d.lib

opencv\_features2d245d.lib

opencv\_flann245d.lib

opencv\_gpu245d.lib

opencv\_haartraining\_engined.lib

opencv\_highgui245d.lib

opencv\_imgproc245d.lib

opencv\_legacy245d.lib

opencv\_ml245d.lib

opencv\_nonfree245d.lib

opencv\_objdetect245d.lib

opencv\_photo245d.lib

opencv\_stitching245d.lib

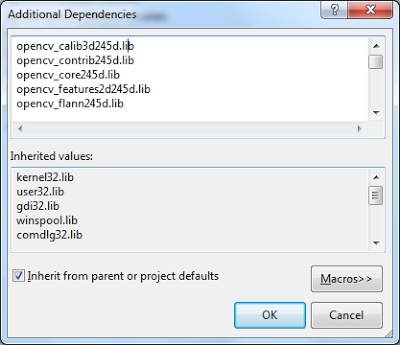
opencv\_superres245d.lib

opencv\_ts245d.lib

opencv\_video245d.lib

opencv\_videostab245d.lib

These are some of the file names, you can find in the 'C:\opencv\build\x86\vc11' location.



Once this is done, all your configurations of your IDE is done and you can start your Application.

**5.4 Functions implemented**

1. Imread(Mat)

Imread takes the first image object captured by the web camera as the input which it reads and processes to be stored in another instance of type Mat for further processing.

1. <VideoCapture>read(Mat)

The read function is invoked with an instance of VideoCapture. It takes instance of Mat as input which is the frame currently captured by the Web Camera. The image is stored in another Mat instance for further processing. This function stores every frame captured by the web camera on a fly for processing and hence is placed in an infinite loop.

1. cvtColor(Mat, Mat, conversion\_type)

cvtColor takes as input 2 instances of Mat, one of which contains the image current captured by the web camera and to be processed and another Mat frame to store the transformed image. The third parameter specifies the type of the conversion to be applied to the frame that can range from converting RGB to HSV or RGB to binary image, etc.

1. morphologicalImgProc(Mat &frame)

This function takes as input an instance of Mat which contains the frame captured by the web camera. This function takes the difference between the dilation and erosion of the image so that image details can be read distinctly and the processing and output is accurate.

1. inRange(Mat, Scalar(B,G,R), Scalar(B, G, R), Mat);

inRange function is used as a filter to consider the pixels falling within the range of the colors specified as the parameters of the scalar function. Scalar function takes as input the green, blue and red values that defines uniquely a specific color. Hence, all pixels that fall within the gradients of the 2 color ranges specified are highlighted eliminating the other colored pixels and storing the processed image in another instance of Mat.

1. TrackHand(Mat src, Mat &dest)

trackHand function takes as input, the processed image and finds the maximum area of the contours found. It then finds the convex defects in the contour thus found and assumes it as the hand. It then returns the number of fingers based on the number of contours drawn on the tip of the convex defects

**5.5 Source Code**

#include <functional>

#include <mutex>

#include <set>

#include <thread>

#include <iostream>

#include<Windows.h>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/video/background\_segm.hpp>

//#include <shellapi.h>

#define PI 3.14159265

using namespace cv;

using namespace std;

//declare the local functions

void morphologicalImgProc(Mat &frame);

string integerToString(int num);

int angleToCenter(const Point &v1, const Point &v2);

String doAction(int totalAngleOfFinger, int fingerSize);

void sendResult(String msg);

static int kk = 0,okay=0;

SHELLEXECUTEINFOA exInfo = { 0 };

//calculate the angle between two points

int angleToCenter(const Point &finger, const Point &center) {

float y\_angle = center.y - finger.y; //center = 1;

float x\_angle = finger.x - center.x;// tip =2;

float theta = atan(y\_angle / x\_angle);

int angleFinger = (int)round(theta \* 180 / PI);

return angleFinger;

}

//convert the integer to string

string integerToString(int num) {

stringstream strings;

strings << num;

string s = strings.str();

return s;

}

//morphological Image processing

//Erosion -> dilation -> closing the frame ensure to get the better performance

void morphologicalImgProc(Mat &frame) {

Mat element = cv::getStructuringElement(cv::MORPH\_ELLIPSE, Size(9, 9), Point(5, 5));

Mat element1 = cv::getStructuringElement(cv::MORPH\_ELLIPSE, Size(7, 7), Point(5, 5));

cv::dilate(frame, frame, element);

cv::erode(frame, frame, element);

cv::morphologyEx(frame, frame, MORPH\_OPEN, element);

cv::morphologyEx(frame, frame, MORPH\_CLOSE, element);

}

//the important function to track the hand, the algorithm is described in the report

std::string trackHand(Mat src, Mat &dest) {

//initialization local variables

Rect boundRect;

int largestObj;

int boundingBoxHeight = 0;

vector<vector<Point> > contours; //store all the contours

vector<vector<Point> > contoursSet(contours.size());//store large contours

vector<Vec4i> hierarchy;

vector<Point> convexHullPoint;

vector<Point> fingerPoint;

Point centerP;

int numObjects = 0;

double area = 0;

double maxArea = 0;

bool handFound = false;

String resultMsg = "no-command";

//find all the contours in the threshold Frame

findContours(src, contours, hierarchy, CV\_RETR\_EXTERNAL, CV\_CHAIN\_APPROX\_SIMPLE);

numObjects = hierarchy.size();

for (unsigned int i = 0; i < contours.size(); i++) {

Mat tempContour = Mat(contours[i]);

area = contourArea(tempContour);

if (area > maxArea) {

maxArea = area;

largestObj = i;

}

}

if (maxArea > 4000){

handFound = true;

boundRect = boundingRect(contours[largestObj]);

//draw the boundary of the object

drawContours(dest, contours, largestObj, Scalar(0, 0, 255), 3, 8, hierarchy);

//find the convex points for the largest object which is hand

convexHull(contours[largestObj], convexHullPoint, true, true);

approxPolyDP(Mat(contours[largestObj]), contours[largestObj], 3, true);

//use moment method to find the center point

Moments moment = moments(Mat(contours[largestObj]), true);

int centerX = moment.m10 / moment.m00;

int centerY = moment.m01 / moment.m00;

Point centerPoint(centerX, centerY);

centerP = centerPoint;

Point printPoint(centerX, centerY + 15);

Point printPoint1(boundRect.x, boundRect.y);

circle(dest, centerPoint, 8, Scalar(255, 0, 0), CV\_FILLED);

rectangle(dest, boundRect, Scalar(0, 0, 255), 2, 8, 0);

boundingBoxHeight = boundRect.height;

if (handFound) {

int countHullPoint = convexHullPoint.size();

int maxdist = 0;

int pos = 0;

for (int j = 1; j < countHullPoint; j++) {

pos = j;

if (centerP.y >= convexHullPoint[j].y && centerP.y >= convexHullPoint[pos].y) {

pos = j;

int dist = (centerP.x - convexHullPoint[j].x) ^ 2 + (centerP.y - convexHullPoint[j].y) ^ 2;

if (abs(convexHullPoint[j - 1].x - convexHullPoint[j].x) < 12){

if (dist > maxdist){

maxdist = dist;

pos = j;

}

}

else if (j == 0 || abs(convexHullPoint[j - 1].x - convexHullPoint[j].x) >= 12){

fingerPoint.push\_back(convexHullPoint[pos]);

cv::line(dest, centerP, convexHullPoint[pos], Scalar(0, 255, 0), 3, CV\_AA, 0);

circle(dest, convexHullPoint[pos], 8, Scalar(255, 0, 0), CV\_FILLED);

pos = j;

}

}

}

//get the size the fingers, and calculate the total angle of these fingers

int countFinger = fingerPoint.size();

int angle = 0;

if (countFinger <= 5){

for (int x = 0; x < countFinger; x++){

angle = angle + abs(angleToCenter(fingerPoint[x], centerP));

}

}

Scalar(0, 255, 0), 1, 5, false);

}

}

return resultMsg;

}

//send out the result signal

void sendResult(String msg){

cout << "Command: " << msg << endl;

}

//action performed based on the number of fingers and the total angle

//1. 5 fingers && total angle: 265 - 295

//2. 4 fingers && total angle: 235 - 260

//3. 3 fingers && total angle: 190 - 210

//4. 2 fingers && total angle: 110 - 130

//5. 1 finger && total angle: 65 - 85

String doAction (const int totalAngleOfFinger, const int fingerSize){

String result = "no-command";

//totalAngleOfFinger >= 265 && totalAngleOfFinger <= 295 && (

if (fingerSize == 5)

{

//result = "scroll-up";

if (kk == 0){

//::ShellExecuteA(NULL, "open", "C:\\Program Files (x86)\\Adobe\\Reader 11.0\\Reader\\PDFSigQFormalRep.pdf", NULL, NULL, SW\_SHOW);

kk++;

exInfo.cbSize = sizeof(SHELLEXECUTEINFOA);

exInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

exInfo.hwnd = NULL;

exInfo.lpVerb = "open";

exInfo.lpFile = "C:\\Program Files (x86)\\Adobe\\Reader 11.0\\Reader\\PDFSigQFormalRep.pdf";

exInfo.lpDirectory = NULL;

exInfo.nShow = SW\_SHOW;

exInfo.hInstApp = NULL;

exInfo.lpParameters = "";

ShellExecuteExA(&exInfo);

//DWORD LastError = GetLastError();

WaitForSingleObject(exInfo.hProcess, 10);

}//system("PDFSigQFormalRep.pdf");

//ShellExecute(NULL, "opens", "C:\\Program Files\\Columbia Application Fee.pdf", NULL, NULL, SW\_SHOW);

//ShellExecute(0,"Open","%s\\HELP\\RiverCADPro\_User\_\

anual.pdf",NULL, NULL,SW\_MAXIMIZE);

}

//totalAngleOfFinger >= 240 && totalAngleOfFinger <= 265 &&

else if (fingerSize == 4)

{

if (!okay&&kk == 1){

INPUT ip;

// Pause for 0.5 seconds

Sleep(500);

// Set up a generic keyboard event.

ip.type = INPUT\_KEYBOARD;

ip.ki.wScan = 0; // hardware scan code for key

ip.ki.time = 0;

ip.ki.dwExtraInfo = 0;

// Press the "A" key

ip.ki.wVk = 0x28; // virtual-key code for the down key

ip.ki.dwFlags = 0; // 0 for key press

SendInput(1, &ip, sizeof(INPUT));

// Release the "A" key

ip.ki.dwFlags = KEYEVENTF\_KEYUP; // KEYEVENTF\_KEYUP for key release

SendInput(1, &ip, sizeof(INPUT));

// Exit normally

//totalAngleOfFinger >= 190 && totalAngleOfFinger <= 210 &&

//::ExitProcess(0);

//::PostThreadMessage(piProcessInfo.dwThreadId, WM\_CLOSE, 0, 0);

}

}

//ShellExecuteA(NULL, "open", "C:\\Program Files (x86)\\Adobe\\Reader 11.0\\Reader\\University of Illinois at Urbana-Champaign \_ Rishabh Rishi Sharma.pdf", NULL, NULL, SW\_SHOW);

//ShellExecuteA(NULL, "open", "C:\\Program Files (x86)\\Adobe\\Reader 11.0\\Reader\\AcroRd32.exe", NULL, NULL, SW\_SHOW);

else if (fingerSize == 3)

{

if (!okay&&kk == 1){

// This structure will be used to create the keyboard

// input event.

INPUT ip;

// Pause for 0.5 seconds

Sleep(500);

// Set up a generic keyboard event.

ip.type = INPUT\_KEYBOARD;

ip.ki.wScan = 0; // hardware scan code for key

ip.ki.time = 0;

ip.ki.dwExtraInfo = 0;

// Press the "A" key

ip.ki.wVk = 0x26; // virtual-key code for the down key

ip.ki.dwFlags = 0; // 0 for key press

SendInput(1, &ip, sizeof(INPUT));

// Release the "A" key

ip.ki.dwFlags = KEYEVENTF\_KEYUP; // KEYEVENTF\_KEYUP for key release

SendInput(1, &ip, sizeof(INPUT));

// Exit normally

//totalAngleOfFinger >= 190 && totalAngleOfFinger <= 210 &&

//::ExitProcess(0);

//::PostThreadMessage(piProcessInfo.dwThreadId, WM\_CLOSE, 0, 0);

}

}

else if (totalAngleOfFinger >= 100 && totalAngleOfFinger <= 130 && (fingerSize == 2))

{

result = "refresh-page";

}

else if (totalAngleOfFinger >= 65 && totalAngleOfFinger <= 95 && (fingerSize == 1))

{

// result = "color-divs";

if (!okay&&kk == 1){

//ShellExecuteA(NULL, "print", "C:\\Program Files (x86)\\Adobe\\Reader 11.0\\Reader\\PDFSigQFormalRep.pdf", NULL, NULL, SW\_SHOW);

TerminateProcess(exInfo.hProcess, 1);

okay++;

}

}

return result;

}

int main() {

Mat cameraFrame, blurFrame, threshold1, threshold2, closedFrame, hsvFrame, colorObjectFrame, thresholdFrame;

VideoCapture stream1;

Mat grayscale;

Mat fgMaskMOG;

Mat foreground;

Mat background;

// Ptr <BackgroundSubtractor> pMOG;

//pMOG = new BackgroundSubtractorMOG();

//default the capture frame size to the certain size & open the camera

stream1.set(CV\_CAP\_PROP\_FRAME\_WIDTH, 640);

stream1.set(CV\_CAP\_PROP\_FRAME\_HEIGHT, 480);

stream1.open(0);

background = imread("background.jpg");

//report the error if the camera not connected properly

if (!stream1.isOpened()) {

cout << "cannot open camera";

}

int delay = 0;

while (1) {

//get image from stream

stream1.read(cameraFrame);

//pMOG->operator ()(cameraFrame, foreground);

medianBlur(foreground, foreground, 3);

//cv::cvtColor(cameraFrame,foreground, CV\_GRAY2BGR);

//switch the RGB to HSV space

cv::cvtColor(cameraFrame, hsvFrame, CV\_BGR2HSV);

cv::inRange(hsvFrame, Scalar(58, 58, 95), Scalar(133, 154, 256), thresholdFrame);

medianBlur(thresholdFrame, thresholdFrame, 5);

morphologicalImgProc(thresholdFrame);

string command = trackHand(thresholdFrame, cameraFrame);

namedWindow("Hand\_Gesture\_Detection");

imshow("Hand\_Gesture\_Detection", cameraFrame);

if (waitKey(10) >= 0)

break;

//release the memory

cameraFrame.release();

delay++;

if (delay > 10){

delay = 0;

std::cout << command << std::endl;

}

}

}

**3. CONCLUSION**

**6.1 Conclusion**

The importance of gesture recognition lies in building efficient human–machine interaction. Its applications range from sign language recognition through medical rehabilitation to virtual reality. Given the amount of literature on the problem of gesture recognition and the promising recognition rates reported, one would be led to believe that the problem is nearly solved. Sadly this is not so. A main problem hampering most approaches is that they rely on several underlying assumptions that may be suitable in a controlled lab setting but do not generalize to arbitrary settings. Several common assumptions include: assuming high contrast stationary backgrounds and ambient lighting conditions. Also, recognition results presented in the literature are based on each author’s own collection of data, making comparisons of approaches impossible and also raising suspicion on the general applicability.

The most important question to be answered when using hand gestures is what recognition technique will maximize accuracy and robustness. A number of recognition techniques are available and in some cases, where hand gesture recognition is to be used, the possibilities are narrowed down, as some algorithms are best fitted only for gesture recognition.

Thus, we have implemented a new technique to increase the adaptability of a gesture recognition system. We have implemented a real-time version, using an ordinary workstation with no special hardware beyond a video camera input. The technique works well under different degrees of scene background complexity and illumination conditions with more than 94% success rate.

**6.2 Future Work**

Considering the relative infancy of research related to vision-based gesture recognition, remarkable progress has been made. To continue this momentum, it is clear that further research in the areas of feature extraction, classification methods and gesture representation are required, to realize the ultimate goal of humans interfacing with machines on their own natural terms.

There are a number of interesting areas for future research in hand gesture recognition. The field is not yet very matured – we have a long way to go before this type of concept is robust enough to be seen in commercial, main stream applications. Research into better hardware for data collection is important.

Though the project seems to be promising, further work is required to improve the processing speed and the recognition process. Future work includes not only improvement of the designed strategy but also taking into account more challenges such as dynamic gestures involving both hands and/or multiple cameras. Our final objective involves gestures with a high degree of freedom; which may require detection of fingers and articulated hands.

This project has a vast arena of development, notably the Sixth Sense project of Pranav Mistry which completely revolutionizes the digital world. The code can be extended to incorporate mouse movements as well as still gestures. Further tweaks can be incorporated in the code to increase the efficiency of the gesture recognition process. The code can be improved for better interpretation and recognition of the gestures and newer gestures maybe incorporated for more functionalities.