Intelligent Human-Computer Interaction System

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Computer Science

by

Abhinav Paliwal – 061203

Sahil Gupta – 061284

Sahil Jain – 061285

under the Supervision of

Sh. Satish Chandra

विद्या तत्व ज्योतिगमः
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

May 2010

Jaypee University of Information Technology
Waknaghat, Solan - 173 234, Himachal Pradesh

Acknowledgement

The Intelligent Human Computer Interaction project marks the culmination of all the concepts assimilated while studying concepts of Image Processing. It has presented us with an opportunity to use the technical know-how to create a real time system.

Learning through the project under the guidance of our esteemed mentor Sh. Satish Chandra, whose expertise knowledge in the domain of Artificial Intelligence as well as Image Processing, not only cleared all our ambiguities but also generated a high level of interest and gusto in the subject. We are truly grateful for his guidance and support throughout the project. We would also like to thank our Head of the Department, Brig (Retd.) SP Ghrera for his undying faith in the department of computer science and allocation of the project as well as its resources.

The prospect of working in a group with high level of accountability fostered a spirit of team work and created a feeling of oneness which thus, expanded our ken, motivated us to perform to the best of our ability and create a report of the highest quality.

To do the best quality work, with utmost sincerity and precision has been our constant endeavor.

Date: 19/05/10

Abhinav Paliwal(061203)

Sahil Gupta (061284)

Sahil Jain (061285)

Abstract

Head Tracking technology is an alternative to the mouse that allows the person control over the movement of the cursor by using only the movement of his head. We present a simple prototype system for real time tracking of a human head. This system uses a simple yet an effective Face tracking algorithm.

The general requirements of a real time tracking algorithm – it should be computationally inexpensive, should possess the ability to perform in different environments and should be able to start and initialize itself with minimum knowledge about the environment, are well addressed by the elliptical head tracking algorithm. The objective of this project is to create an alternative user interface uniquely using real time video of the user's face captured using an off-the-shelf web-camera. The position of the head is tracked and converted into two-dimensional coordinates on a computer screen; additionally, it is intended to enable the recognition of a deliberate blink in order that this could be considered as a command from a user.

People who are quadriplegic and somewhat physically disabled, for example from cerebral palsy or traumatic brain injury or stroke, have limited motions they can make voluntarily. Some people can move their heads. Some can blink or wink voluntarily. A low cost system is proposed to assist such people in handling computers and/or connected peripherals. Still, there are many people with no reliable means to access the computer.

Table of Content

List of Figures

Chapter 1
General Discussion

1.1 Introduction

In the recent years there has been a major effort to develop new tools that allow an advanced human-computer interaction. These type of interfaces are commonly addressed as Perceptual User Interface for Computers. Interfaces that allow a computer to be operated without the use of one's hands are particularly interesting as they can be considered a step towards a 'transparent computerized assistant'. A system capable of receiving input from natural human behavior, instead of forcing the user to learn to utilise devices such keyboard and mouse.

People who are quadriplegic and somewhat physically disabled, for example from cerebral palsy or traumatic brain injury or stroke, have limited motions they can make voluntarily. Some people can move their heads. Some can blink or wink voluntarily. Some can move the eyes or tongue. Family, friends, and other care providers usually detect these motions visually.

Many computer access methods have been developed to help people who are quadriplegic and nonverbal: external switches, devices to detect small muscle movements or eye blinks, head pointers, infrared or near infrared camera based systems to detect eye movements, electrode based systems to measure the angle of the eye in the head, even systems to detect features in EEG. These have helped many people access the computer and have made tremendous improvements in their lives. Still, there are many people with no reliable means to access the computer. We are interested in developing computer vision systems that work under normal lighting to provide computer access to people who are quadriplegic and physically disabled.

There are various systems capable of interpreting a human face as a form of input, an application which is particularly important for people with severe disabilities, but who are able to move their head, and for able-bodied users that require an additional input source while they operate a personal computer with both hands.

1.2 Problem Statement

Develop a system that uses a camera to visually track a feature on a person's face, for example the tip of the nose, and use the movement of the tracked feature to directly control the mouse pointer on a computer.

1.3 Objective & Scope of the Project

This project explores the field of computer vision with the broad aim of developing a system capable of interpreting the movements of human facial features. The facial movements considered are rotation of the head and blinking of the eyes. The three dimensional position of the head is tracked and converted to 2D coordinates on the computer screen; at the same time, intentional blinks are recognized and interpreted as an action. The tracker works uniquely using the real time video of the person sat in front of the Screen.

- To design an Intelligent Agent for real time tracking of head movements
- To provide a Computer vision system to provide computer access to people who are quadriplegic and physically disabled.
- Performance requirements: computationally inexpensive, adaptability, self initializing, use of minimum knowledge about the environment.
- The system would detect the Head Movement as Input and Process to give corresponding mouse movement as the output.

1.4 Existing System

A number of techniques and tools are available for assisting the disabled. Most of these make use of one or more of the following:

- External switches.
- Head pointers.
- Infrared or near infrared camera based systems to detect eye movements.
- Electrode based systems that measure the angle of the eye in the head.
- Electric Encephalography (ECG) based systems.

These have helped many people access the computer and have made tremendous improvements in their lives. Still, there are many people with no reliable means to access the computer.

In the modern world, computer use has become essential for many everyday tasks such as electronic communications, information gathering, and recreational activities. The current computer interface set up of a mouse and keyboard requires the user to have full use of his or her hands. Unfortunately, many people do not have sufficient use of their hands due to injury or illness and are thus unable to use a computer using traditional hardware

Some alternative interfaces have been developed using electroencephalograms (EEGs) and eye motion, however these require a great deal of expensive hardware, require significant processing time, and only give the user limited control. More recently, development has focused on systems that monitor head motion either electromechanically or optically. These systems can provide faster speeds and more control, but they are often very expensive and difficult or awkward to use. Additionally, many of these systems utilize mouth controls to some degree, which can by hygienically troublesome.

A system called Quick Glance monitors the location of a user's pupils using an infrared emitter/receiver. In this way, direction of the eyes can be estimated, and the mouse moved accordingly. The trigger for a mouse click is a consecutive eye blink. However, in

order to use this system a user must maintain their head completely still, for this reason the Applied Science Laboratory with their Mobile Eye use a similar
technique but the camera is installed directly on a pair of special glasses, making head movements possible.
Other control devices that monitors eye movements uses the electrooculographic potential. Electrodes are attached to the user's face in order to measure all the electrical changes that occur.

The problem with all of these systems is twofold: firstly, they all make use of some sort of non-standard hardware, which in some cases is expensive and for which there may be very limited support. Secondly, a user must wear pieces of hardware that can be considered invasive, such as an infrared emitter, reflectors, electrodes or helmets.

In contrast, an interface based on an image captured by a normal web-camera is completely 'transparent' to the user, who may use it without not even notice how it is working. Such hardware is now increasingly found in most of the personal computers configurations.

Chapter 2
Literature survey

The implementation of this project requires an extensive knowledge of –

1) Optical flow
2) Face detection
3) Feature Extraction
4) OpenCV library

2.1 Optical flow

It is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene. Using the optical flow methods we try to calculate the motion between two image frames which are taken at times t and $t + \delta t$ at every position. Sequences of ordered images allow the estimation of motion as either instantaneous image velocities or discrete image displacements.

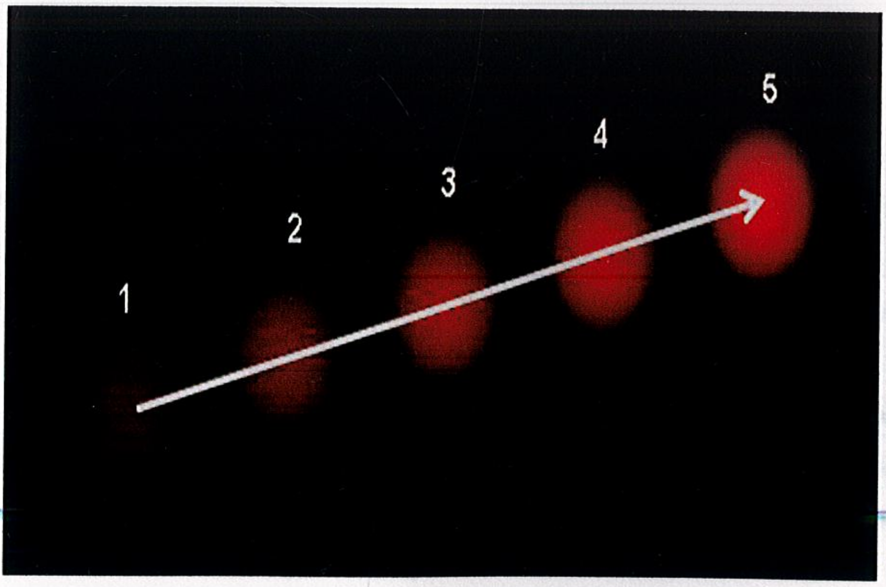


Fig 1: Optical flow example

2.1.1Uses of optical flow

Motion estimation and video compression have developed as a major aspect of optical flow research. While the optical flow field is superficially similar to a dense motion field derived from the techniques of motion estimation, optical flow is the study of not only the determination of the optical flow field itself, but also of its use in estimating the three-dimensional nature and structure of the scene, as well as the 3D motion of objects and the observer relative to the scene.

Optical flow was used by robotics researchers in many areas such as: object detection and tracking, image dominant plane extraction, movement detection, robot navigation and visual odometry.

The application of optical flow includes the problem of inferring not only the motion of the observer and objects in the scene, but also the structure of objects and the environment. Since awareness of motion and the generation of mental maps of the structure of our environment are critical components of animal (and human) vision, the conversion of this innate ability to a computer capability is similarly crucial in the field of machine vision.

The optical flow vector of a moving object in a video sequence.

Consider a five-frame clip of a ball moving from the bottom left of a field of vision, to the top right. Motion estimation techniques can determine that on a two dimensional plane the ball is moving up and to the right and vectors describing this motion can be extracted from the sequence of frames. For the purposes of video compression (e.g., MPEG), the sequence is now described as well as it needs to be. However, in the field of machine vision, the question of whether the ball is moving to the right or if the observer is moving to the left is unknowable yet critical information. Not even if a static, patterned background were present in the five frames, we could confidently state that the ball was moving to the right, because the pattern might have an infinite distance to the observer.

2.2 Face detection

Face detection is a computer technology that determines the locations and sizes of human faces in arbitrary (digital) images. It detects facial features and ignores anything else, such as buildings, trees and bodies. Face detection can be regarded as a specific case of object-class detection; In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars.

Face detection can be regarded as a more general case of face localization; In face localization, the task is to find the locations and sizes of a known number of faces (usually one). In face detection, one does not have this additional information.

Early face-detection algorithms focused on the detection of frontal human faces, whereas newer algorithms attempt to solve the more general and difficult problem of multi-view face detection. That is, the detection of faces that are either rotated along the axis from the face to the observer (in-plane rotation), or rotated along the vertical or left-right axis (out-of-plane rotation),or both.

2.2.1 Techniques

Face detection as a pattern-classification task: Many algorithms implement the face-detection task as a binary pattern-classification task. That is, the content of a given part of an image is transformed into features, after which a classifier trained on example faces decides whether that particular region of the image is a face, or not.

Often, a window-sliding technique is employed. That is, the classifier is used to classify the (usually square or rectangular) portions of an image, at all locations and scales, as either faces or non-faces (background pattern).

Controlled background

Images with a plain or a static background are easy to process. Remove the background and only the faces will be left, assuming the image only contains a frontal face.

By color

Using skin color to find face segments is a vulnerable technique. The database may not contain all the skin colors possible. Lighting can also affect the results. Non-animate objects with the same color as skin can be picked up since the technique uses color segmentation. The advantages are the lack of restriction to orientation or size of faces and a good algorithm can handle complex backgrounds.

By motion

Faces are usually moving in real-time videos. Calculating the moving area will get the face segment. However, other objects in the video can also be moving and would affect the results. A specific type of motion on faces is blinking. Detecting a blinking pattern in an image sequence can detect the presence of a face. Eyes usually blink together and symmetrically positioned, which eliminates similar motions in the video. Each image is subtracted from the previous image. The difference image will show boundaries of moved pixels. If the eyes happen to be blinking, there will be a small boundary within the face.

Model-based

A face model can contain the appearance, shape, and motion of faces. There are several shapes of faces. Some common ones are oval, rectangle, round, square, heart, and triangle. Motions include, but not limited to, blinking, raised eyebrows, flared nostrils, wrinkled forehead, and opened mouth. The face models will not be able to represent any person making any expression, but the technique does result in an acceptable degree of accuracy. The models are passed over the image to find faces, however this technique works better with face tracking. Once the face is detected, the model is laid over the face and the system is able to track face movements.

2.3 Feature Extraction

Edge detection is a terminology in image processing and computer vision, particularly in the areas of feature detection and feature extraction, to refer to algorithms which aim at identifying points in a digital image at which the image brightness changes sharply or more formally has discontinuities.

The purpose of detecting sharp changes in image brightness is to capture important events and changes in properties of the world. It can be shown that under rather general assumptions for an image formation model, discontinuities in image brightness are likely to correspond to:

- discontinuities in depth,
- discontinuities in surface orientation,
- changes in material properties and
- variations in scene illumination.

In the ideal case, the result of applying an edge detector to an image may lead to a set of connected curves that indicate the boundaries of objects, the boundaries of surface markings as well curves that correspond to discontinuities in surface orientation. Thus, applying an edge detector to an image may significantly reduce the amount of data to be processed and may therefore filter out information that may be regarded as less relevant, while preserving the important structural properties of an image. If the edge detection step is successful, the subsequent task of interpreting the information contents in the original image may therefore be substantially simplified. Unfortunately, however, it is not always possible to obtain such ideal edges from real life images of moderate complexity. Edges extracted from non-trivial images are often hampered by *fragmentation*, meaning that the edge curves are not connected, missing edge segments as well as *false edges* not corresponding to interesting phenomena in the image – thus complicating the subsequent task of interpreting the image data.

2.3.1 Edge properties

The edges extracted from a two-dimensional image of a three-dimensional scene can be classified as either viewpoint dependent or viewpoint independent. A *viewpoint independent edge* typically reflects inherent properties of the three-dimensional objects, such as surface markings and surface shape. A *viewpoint dependent edge* may change as the viewpoint changes, and typically reflects the geometry of the scene, such as objects occluding one another.

A typical edge might for instance be the border between a block of red color and a block of yellow. In contrast a line (as can be extracted by a ridge detector) can be a small number of pixels of a different color on an otherwise unchanging background. For a line, there may therefore usually be one edge on each side of the line.

Edges play quite an important role in many applications of image processing, in particular for machine vision systems that analyze scenes of man-made objects under controlled illumination conditions. During recent years, however, substantial (and successful) research has also been made on computer vision methods that do not explicitly rely on edge detection as a pre-processing step.

2.3.2 A simple edge model

Although certain literature has considered the detection of ideal step edges, the edges obtained from natural images are usually not at all ideal step edges. Instead they are normally affected by one or several of the following effects:

- focal blur caused by a finite depth-of-field and finite point spread function.
- penumbral blur caused by shadows created by light sources of non-zero radius.
- shading at a smooth object

and a number of researchers have used a Gaussian smoothed step edge (an error function) as the simplest extension of the ideal step edge model for modeling the effects of edge blur in practical applications.

2.3.3 Approaches to edge detection

There are many methods for edge detection, but most of them can be grouped into two categories, search-based and zero-crossing based. The search-based methods detect edges by first computing a measure of edge strength, usually a first-order derivative expression such as the gradient magnitude, and then searching for local directional maxima of the gradient magnitude using a computed estimate of the local orientation of the edge, usually the gradient direction. The zero-crossing based methods search for zero crossings in a second-order derivative expression computed from the image in order to find edges, usually the zero-crossings of the Laplacian or the zero-crossings of a non-linear differential expression, as will be described in the section on differential edge detection following below. As a pre-processing step to edge detection, a smoothing stage, typically Gaussian smoothing, is almost always applied (see also noise reduction).

The edge detection methods that have been published mainly differ in the types of smoothing filters that are applied and the way the measures of edge strength are computed. As many edge detection methods rely on the computation of image gradients, they also differ in the types of filters used for computing gradient estimates in the x- and y-directions.

2.4 OpenCV Library

OpenCV is a open source computer vision library released by Intel which provides a collection of basic algorithms and some sample data.
In particular it provided the complete trained ANN used to identify the location of the initial faces. Although it is completely implemented in C , it offers an abstraction layer to access the camera driver in a platform independent fashion.

OpenCV is an open source computer vision library originally developed by Intel. It is free for commercial and research use under a BSD license. The library is cross-platform, and runs on Mac OS X, Windows and Linux. It focuses mainly towards real-time image processing, as such, if it finds Intel's Integrated Performance Primitives on the system, it will use these commercial optimized routines to accelerate itself.

This implementation is not a complete port of OpenCV. Currently, this library supports :

- real-time capture
- video file import
- basic image treatment (brightness, contrast, threshold, ...)
- object detection (face, body, ...)
- blob detection

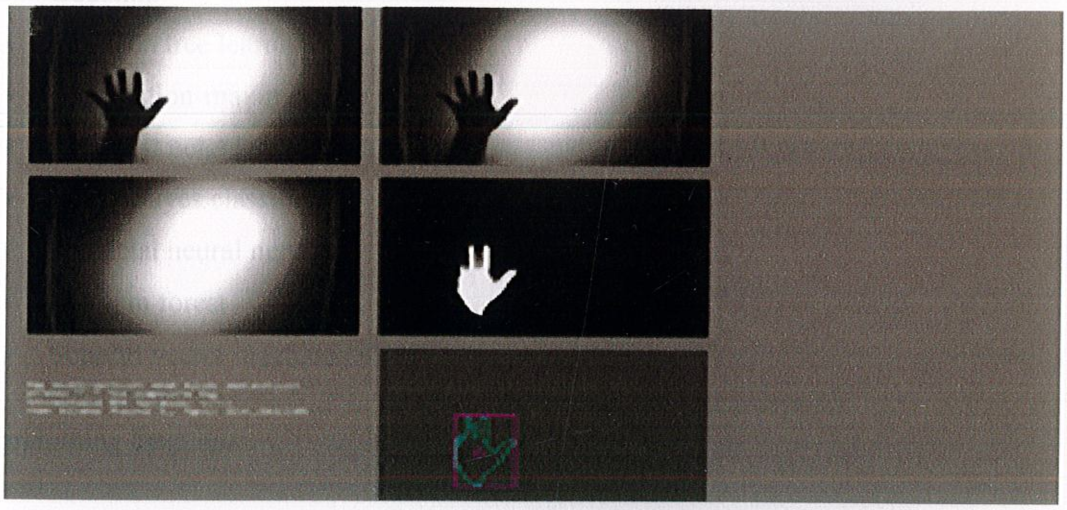


Fig 2: Open Frameworks running the OpenCV add-on example.

2.4.1Applications

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human-Computer Interface (HCI)
- Mobile robotics
- Motion understanding

- Object Identification
- Segmentation and Recognition
- Stereopsis Stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion tracking

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)

Programming language

The library is mainly written in C, which makes it portable to some specific platforms such as Digital signal processor. But wrappers for languages such as C#, Python and Ruby have been developed to encourage adoption by a wider audience.

OS Support

OpenCV runs under FreeBSD, Linux, Mac OS and Windows. The user can get official releases from sourceforge, or take the current snapshot under SVN from there. OpenCV now uses CMake.

The BaseClasses from DirectShow SDK is required to build some camera input-related parts of OpenCV on Windows. This SDK is found in the *Samples\Multimedia\DirectShow\BaseClasses* subdirectory of the Microsoft Platform

SDK (or DirectX SDK 8.0 to 9.0c / DirectX Media SDK prior to 6.0), which must be built prior to the building of OpenCV.

2.4.2 Features of OPEN CV

Features:

- Image data manipulation (allocation, release, copying, setting, conversion).
- Image and video I/O (file and camera based input, image/video file output).
- Matrix and vector manipulation and linear algebra routines (products, solvers, eigenvalues, SVD).
- Various dynamic data structures (lists, queues, sets, trees, graphs).
- Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, color conversion, morphological operations, histograms, image pyramids).
- Structural analysis (connected components, contour processing, distance transform, various moments, template matching, Hough transform, polygonal approximation, line fitting, ellipse fitting, Delaunay triangulation).
- Camera calibration (finding and tracking calibration patterns, calibration, fundamental matrix estimation, homography estimation, stereo correspondence).
- Motion analysis (optical flow, motion segmentation, tracking).
- Object recognition (eigen-methods, HMM).
- Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).
- Image labeling (line, conic, polygon, text drawing)

Chapter 3
System Analysis and Design

3.1 System Flow Diagram

The overall system is as depicted in the block diagram given in figure 1. The image acquisition module grabs frames from the video device which are fed into the tracking module. Now the tracking module checks if the grabbed frame is the first frame. A global search is performed for the first frame and the likelihood is computed. This way the tracking module automatically initializes the tracker by performing this global search on the first frame thus eliminating the need for any explicit knowledge of the environment before hand. For all the subsequent frames, the location of head is found by performing a local search in the search range. Finally the object which is tracked in real time is displayed on the Computer screen.

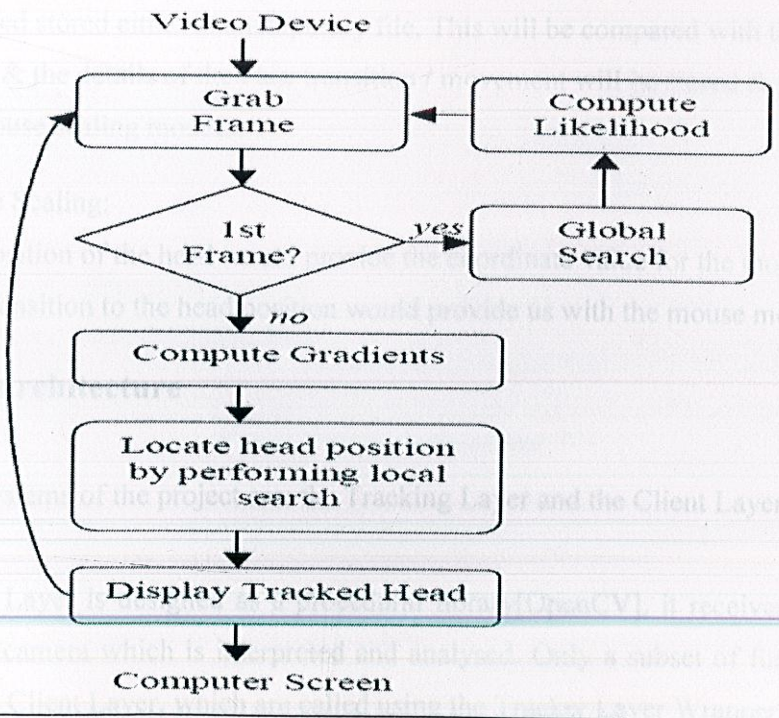


Fig 3: Block diagram for System Flow

3.2 Modular Description

The system comprises of the following modules:

1. Graphical User Interface:

   This consist of creating a GUI of the systems in any higher level programming language like JAVA / .NET. It will be used to provide a User Interface in which the user will be able to initiate various functionalities & view the real time streaming from the web camera. Also user can set various Parameters from this interface so as to run the system under various constraints.

2. Audio / Video Enablement:

   The user interface will give a native call to the core application written in c++. Here we would first enable the hardware device drivers for the Web Camera & Microphone. After this we will capture the live video from the camera.

3. Video Processing:

   The video captured will be processed. The details of the first frame will be captured stored either in a temporary file. This will be compared with the next frame & the details of the Face transition / movement will be stored & provided to the mouse scaling module.

4. Mouse Scaling:

   The position of the head would provide the coordinate value for the mouse cursor. Any transition to the head position would provide us with the mouse movement.

3.3 System architecture

The two subsystems of the project are: the Tracking Layer and the Client Layer.

The Tracking Layer is designed as a procedural library[OpenCV], it receives the input from the web-camera which is interpreted and analysed. Only a subset of functions are exposed to the Client Layer, which are called using the Tracker Layer Wrapper.

The Client Layer is designed as ObjectOriented system in JAVA, it a employs the Wrapper in order to have an internal ObjectOriented representation of the library.

The user can interact with the Framework using the GUI (Graphical User Interface) provided.

There is minimal coupling between these two layers, the only dependency is the 'one way' relationship between Tracker Layer Wrapper and the functions exposed by the Tracking Layer.

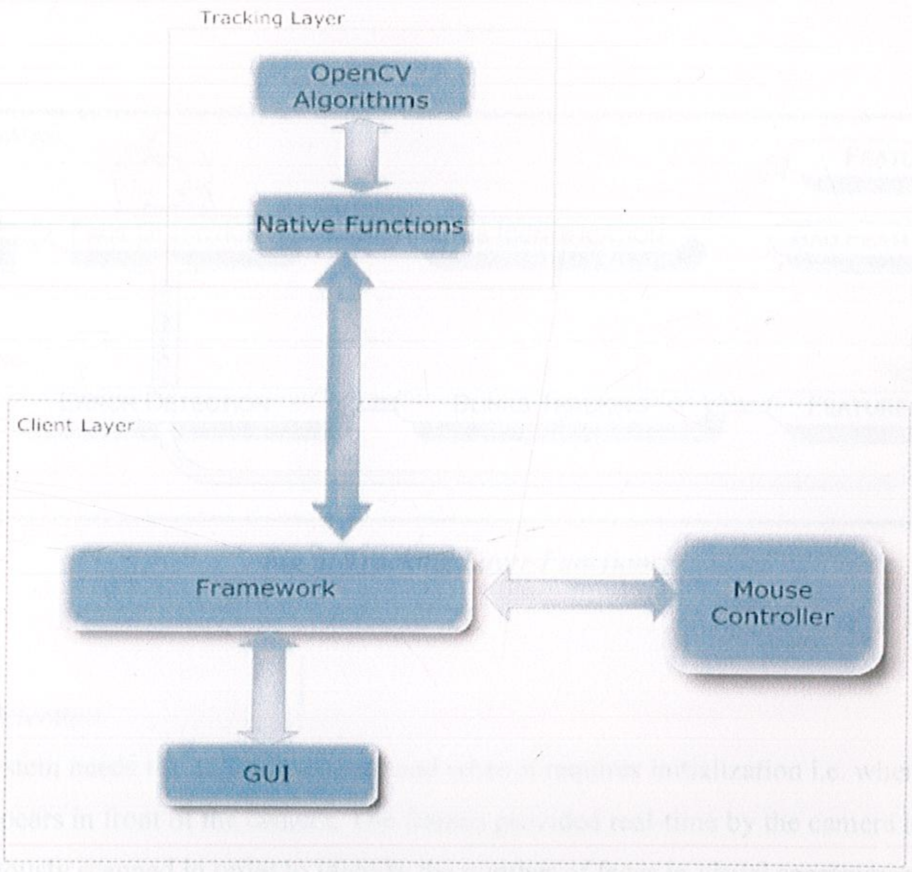


Fig 4: System Architecture

3.3.1 Tracking Layer

The Tracking Layer is responsible for encapsulating all the algorithmic functionality regarding tracking in a single library; it hides all the complexity from the other layer exposing only the high level functions.

Its purpose is to provide a face tracker robust enough to be usable in visually noisy environments and without perfect light conditions. The tracking mechanism must be able to deal efficiently with the natural behavior of a computer user in a typical indoor environment.

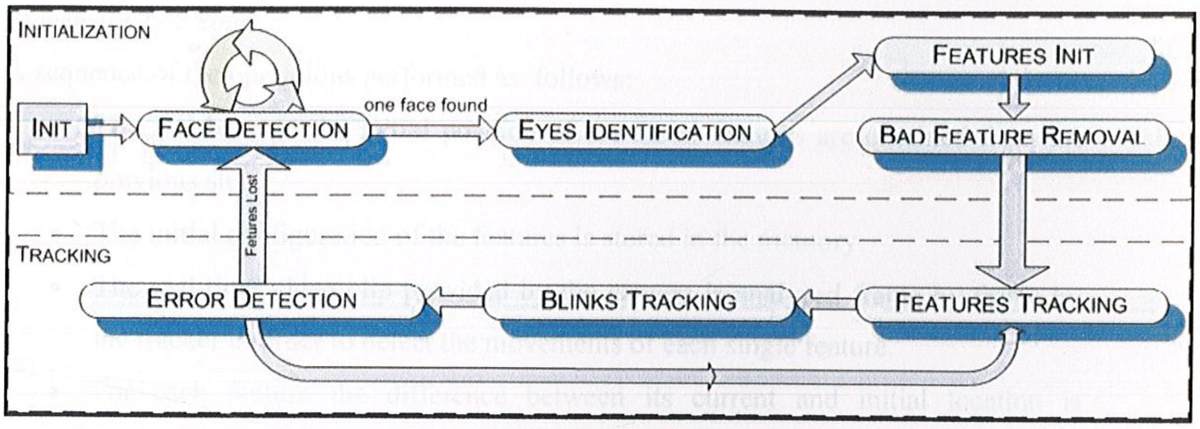


Fig 5: Tracking Layer Functions

Face Detection:

The system needs the ability to understand when it requires initialization i.e. whenever a user appears in front of the camera. The frames provided real-time by the camera are continuously scanned in order to identify the number of faces in visual spectrum, each time a single face is detected its location is calculated and passed to the features identification algorithm.

The network is trained using a set of frontal faces provided with the OpenCV library.

Feature Identification:

The technique created is based on four simple steps:

- Retrieve face location: the location of the face is retrieved thanks to the face detection algorithm from OpenCV.
- Identify eyes location: the location of the eyes is calculated relatively to the location of the face.
- Remove unwanted areas edges of the face
- Identify good features to track

Movement Tracking:

A sequence of the operations performed as follows:

- The number and the initial position of the facial features are obtained from the previous step.
- The initial configuration of the features is stored in the memory.
- The real-time video clip provided by the camera is analyzed frame by frame by the tracker in order to detect the movements of each single feature.
- For each feature the difference between its current and initial location is calculated, then the average of all the differences is calculated. In this way the tracker detects a small movement when the head performs a roll.

Eye Blink Detection:

The technique used to detect the eye blinks is entirely based on the movements of the eyelids which, can be effectively detected using a USB camera (capable of achieving 30 fps).

3.3.2 Client layer

The main aim of Client layer is to make available the GUI that allows the user to manage options & all the functionality for the operation of the system.

3.4 Tracker Function Hierarchy

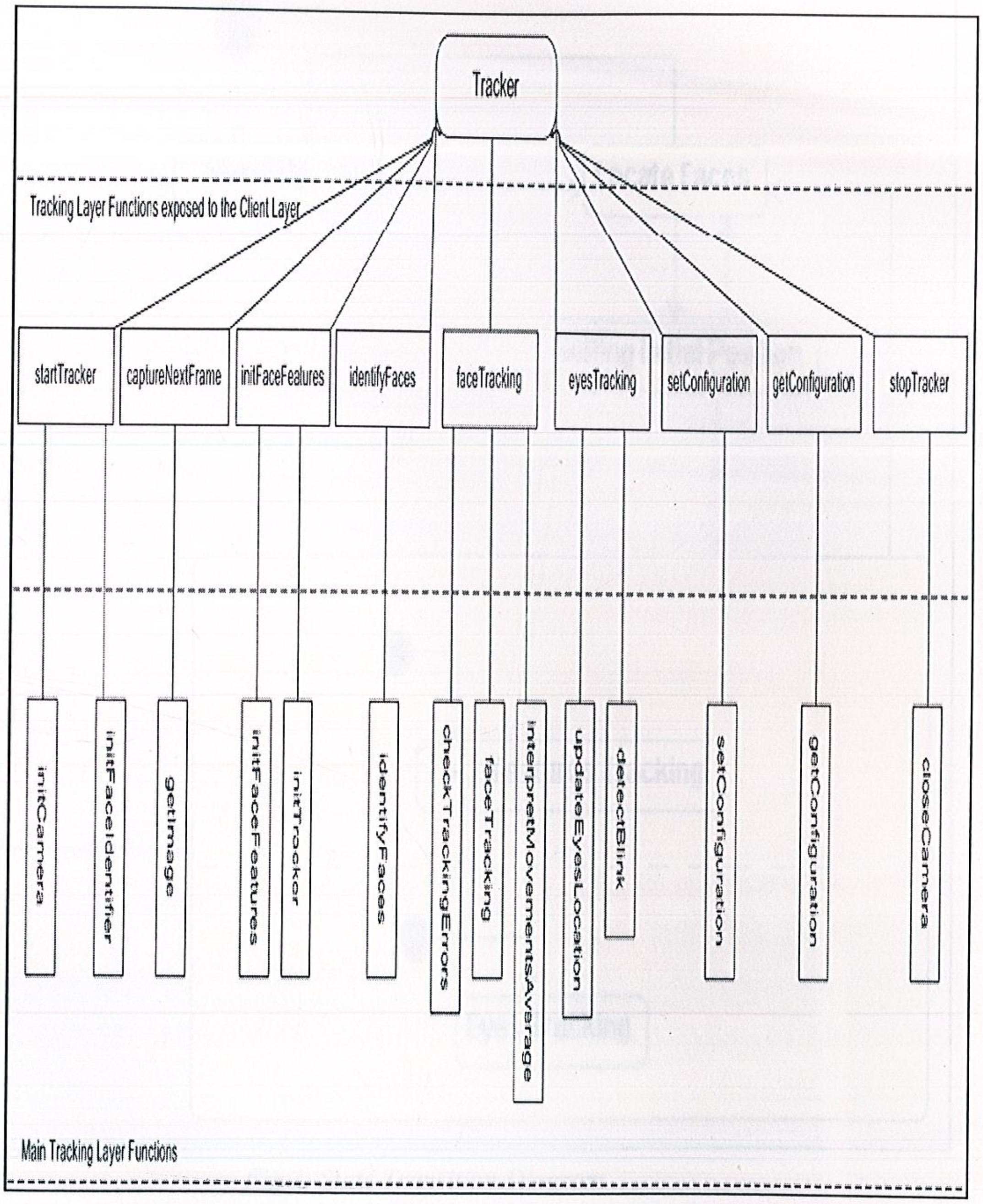


Fig 6: Tracker FunctionHierarchy

3.5State Transition Diagram

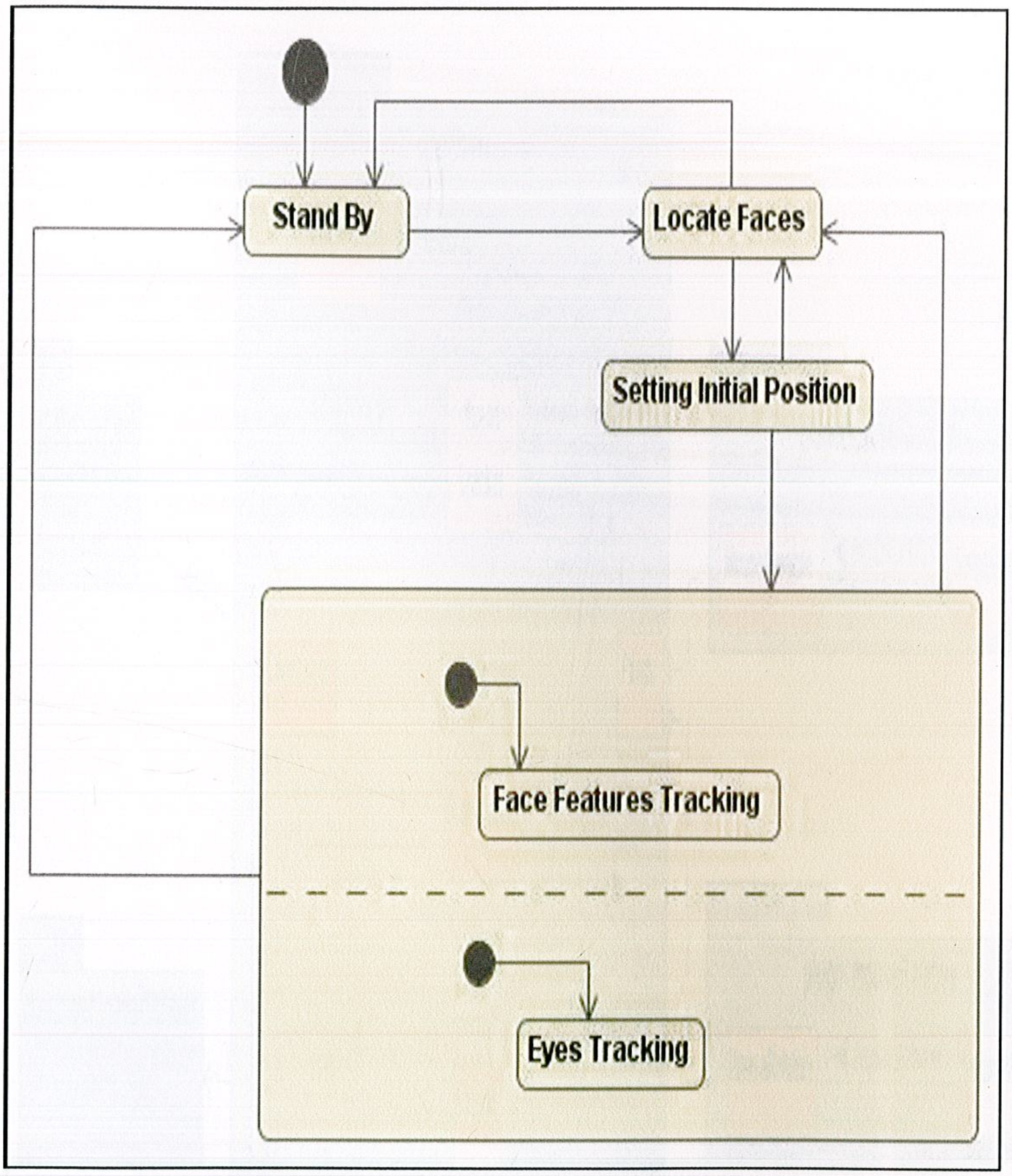


Fig 7: State Transition Diagram

3.6 Class Diagram

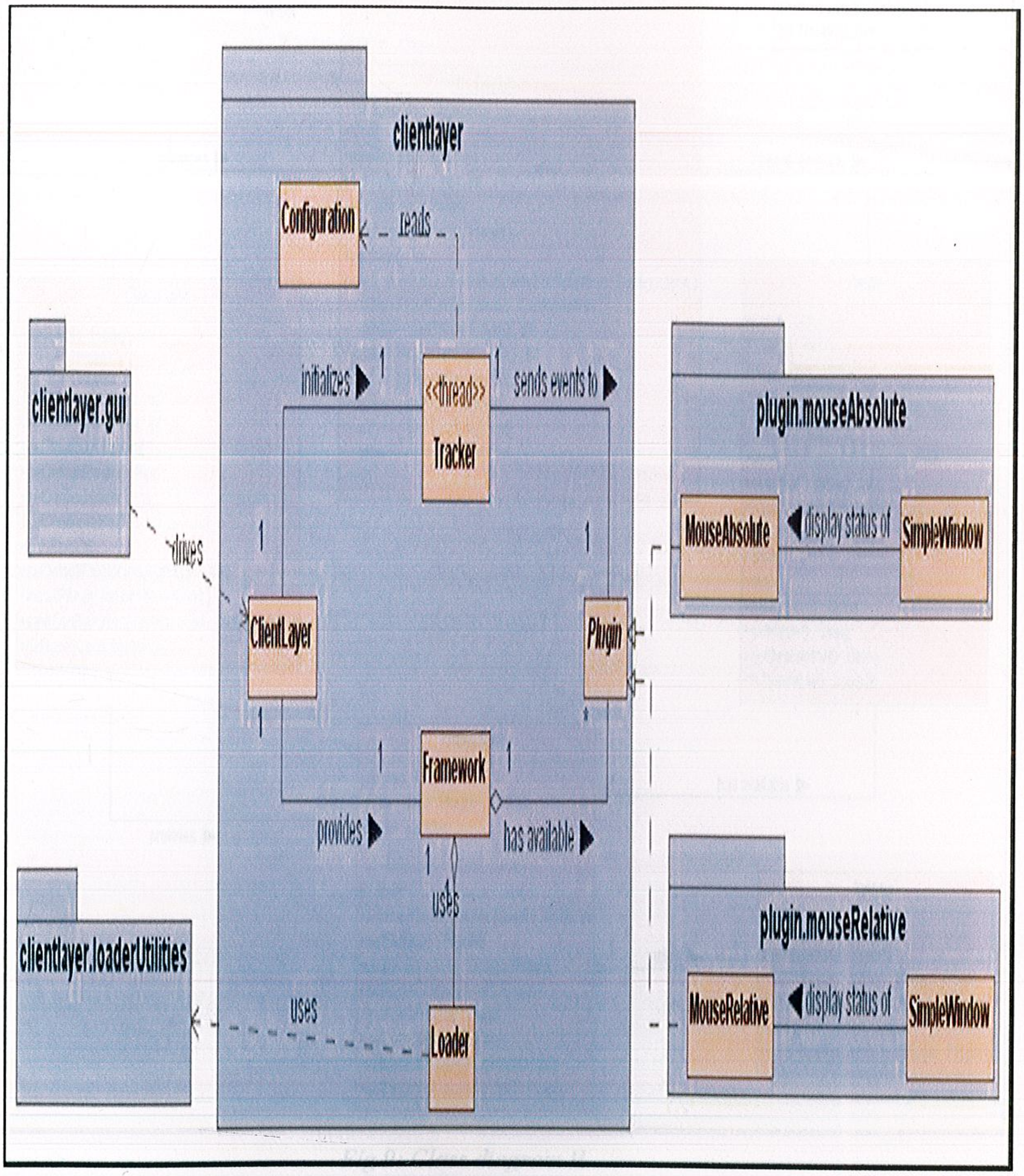


Fig 8: Class Diagram

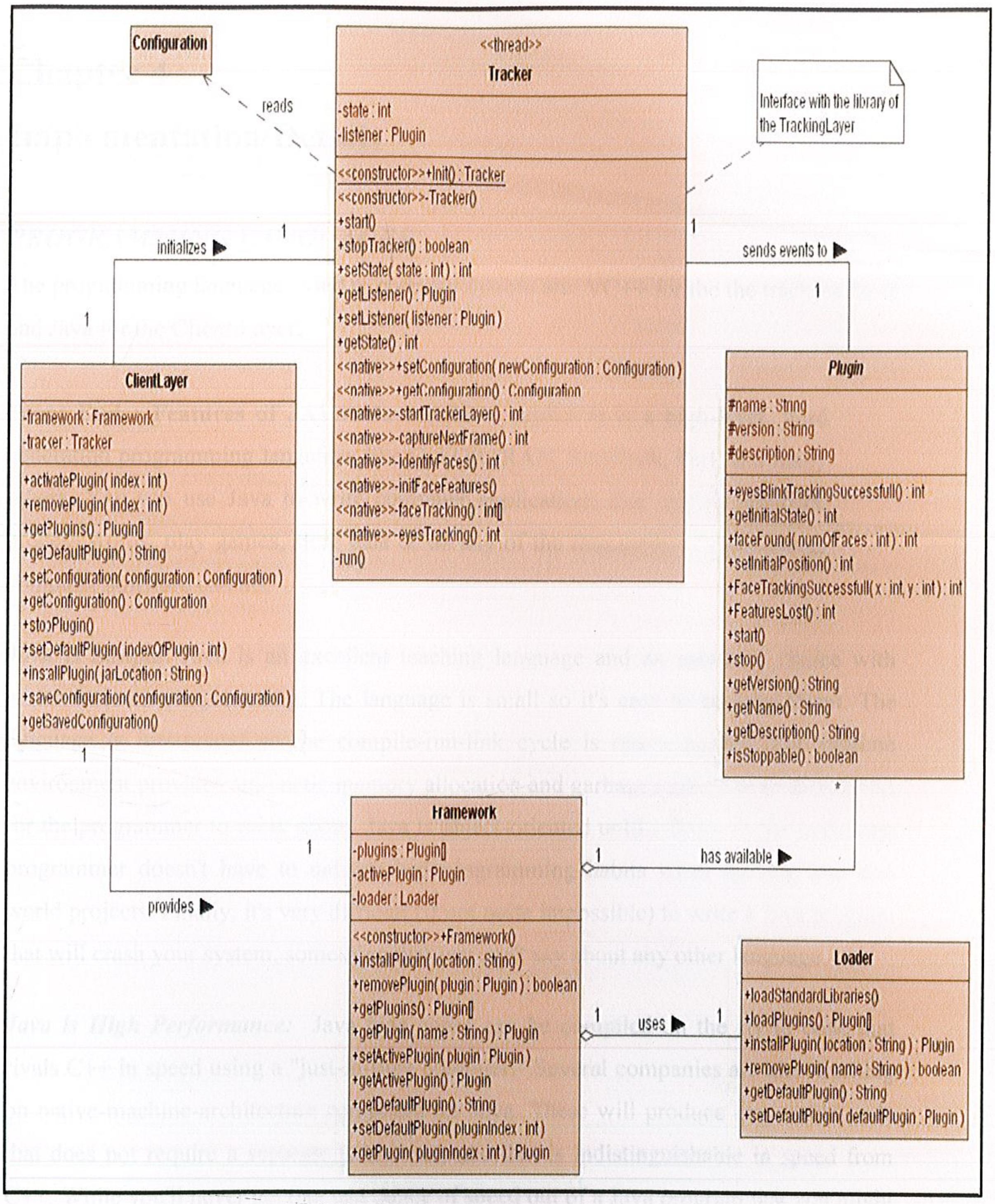

Configuration
reads
<<thread>>
Tracker
-state : int
-listener : Plugin
<<constructor>>+Init() : Tracker
<<constructor>>-Tracker()
+start()
+stopTracker() : boolean
+setState(state : int) : int
+getListener() : Plugin
+setListener(listener : Plugin)
+getState() : int
<<native>>+setConfiguration(newConfiguration : Configuration)
<<native>>+getConfiguration() : Configuration
<<native>>-startTrackerLayer() : int
<<native>>-captureNextFrame() : int
<<native>>-identifyFaces() : int
<<native>>-initFaceFeatures()
<<native>>-faceTracking() : int[]
<<native>>-eyesTracking() : int
-run()
Interface with the library of the TrackingLayer
initializes
1
1
1
sends events to
1
ClientLayer
-framework : Framework
-tracker : Tracker
+activatePlugin(index : int)
+removePlugin(index : int)
+getPlugins() : Plugin[]
+getDefaultPlugin() : String
+setConfiguration(configuration : Configuration)
+getConfiguration() : Configuration
+stopPlugin()
+setDefaultPlugin(indexOfPlugin : int)
+installPlugin(jarLocation : String)
+saveConfiguration(configuration : Configuration)
+getSavedConfiguration()
Plugin
#name : String
#version : String
#description : String
+eyesBlinkTrackingSuccessfull() : int
+getInitialState() : int
+faceFound(numOfFaces : int) : int
+setInitialPosition() : int
+FaceTrackingSuccessfull(x : int, y : int) : int
+FeaturesLost() : int
+start()
+stop()
+getVersion() : String
+getName() : String
+getDescription() : String
+isStoppable() : boolean
1
*
Framework
-plugins : Plugin[]
-activePlugin : Plugin
-loader : Loader
<<constructor>>+Framework()
+installPlugin(location : String)
+removePlugin(plugin : Plugin) : boolean
+getPlugins() : Plugin[]
+getPlugin(name : String) : Plugin
+setActivePlugin(plugin : Plugin)
+getActivePlugin() : Plugin
+getDefaultPlugin() : String
+setDefaultPlugin(pluginIndex : int)
+getPlugin(pluginIndex : int) : Plugin
1
provides
1
has available
1
uses
1
Loader
+loadStandardLibraries()
+loadPlugins() : Plugin[]
+installPlugin(location : String) : Plugin
+removePlugin(name : String) : boolean
+getDefaultPlugin() : String
+setDefaultPlugin(defaultPlugin : Plugin)

Fig 9: Class diagram II

Chapter 4
Implementation Details

PROGRAMMING LANGUAGES

The programming languages used to code the system are: VC++ for the the tracking layer and Java for the Client Layer.

Front End : Features of JAVA: Java (with a capital J) is a high-level, third generation programming language, like C, FORTRAN, Smalltalk, Perl, and many others. You can use Java to write computer applications that crunch numbers, process words, play games, store data or do any of the thousands of other things computer software can do.

Java is Simple: Java is an excellent teaching language and an excellent choice with which to learn programming. The language is small so it's easy to become fluent. The language is interpreted so the compile-run-link cycle is much shorter. The runtime environment provides automatic memory allocation and garbage collection so there's less for the programmer to think about. Java is object-oriented unlike Basic so the beginning programmer doesn't have to unlearn bad programming habits when moving into real world projects. Finally, it's very difficult (if not quite impossible) to write a Java program that will crash your system, something that you can't say about any other language.

Java is High Performance: Java byte codes can be compiled on the fly to code that rivals C++ in speed using a "just-in-time compiler." Several companies are also working on native-machine-architecture compilers for Java. These will produce executable code that does not require a separate interpreter, and that is indistinguishable in speed from C++. While you'll never get that last ounce of speed out of a Java program that you might be able to wring from C or FORTRAN, the results will be suitable for all but the most demanding applications.

Java database Connectivity: A great promise of Java has been the ability to build platform-independent client/server database applications. In Java 1.1 this has come to fruition with Java Data Base Connectivity (JDBC). One of the major problems with databases has been the feature wars between the database companies. There is a "standard" database language, Structured Query Language (SQL-92), but usually you must know which database vendor you're working with despite the standard. JDBC is designed to be platform-independent, so you don't need to worry about the database you're using while you're programming. However, it's still possible to make vendor-specific calls from JDBC so you aren't restricted from doing what you must.

Java is Platform Independent: Java was designed to not only be cross-platform in source form like C, but also in compiled binary form. Since this is frankly impossible across processor architectures Java is compiled to an intermediate form called byte-code. A Java program never really executes natively on the host machine. Rather a special native program called the Java interpreter reads the byte code and executes the corresponding native machine instructions. Thus to port Java programs to a new platform all that is needed is to port the interpreter and some of the library routines. Even the compiler is written in Java. The byte codes are precisely defined, and remain the same on all platforms.

Why Java ?

Here we list the basic features that make Java a powerful and popular programming language:

Platform Independence : The *Write-Once-Run-Anywhere* ideal has not been achieved (tuning for different platforms usually required), but closer than with other languages.

- Object Oriented
- Object oriented throughout - no coding outside of class definitions, including main().
- An extensive class library available in the core language packages.

Compiler/Interpreter Combo

- Code is compiled to byte codes that are interpreted by a Java virtual machines (JVM) .
- This provides portability to any machine for which a virtual machine has been written.
- The two steps of compilation and interpretation allow for extensive code checking and improved security.

Robust

- Exception handling built-in, strong type checking (that is, all data must be declared an explicit type), local variables must be initialized.

Several dangerous features of C & C++ eliminated:

- No memory pointers
- No preprocessor
- Array index limit checking

Features such as eliminating memory pointers and by checking array limits greatly help to remove program bugs. The garbage collector relieves programmers of the big job of memory management. These and the other features can lead to a big speedup in program development compared to C/C++ programming.

The Java Native Interface: The Java Native Interface (JNI) is a programming framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly.

JNI allows one to write native methods to handle situations when an application cannot be written entirely in the Java programming language, e.g. when the standard Java class library does not support the platform-specific features or program library. It is also used to modify an existing application—written in another programming language—

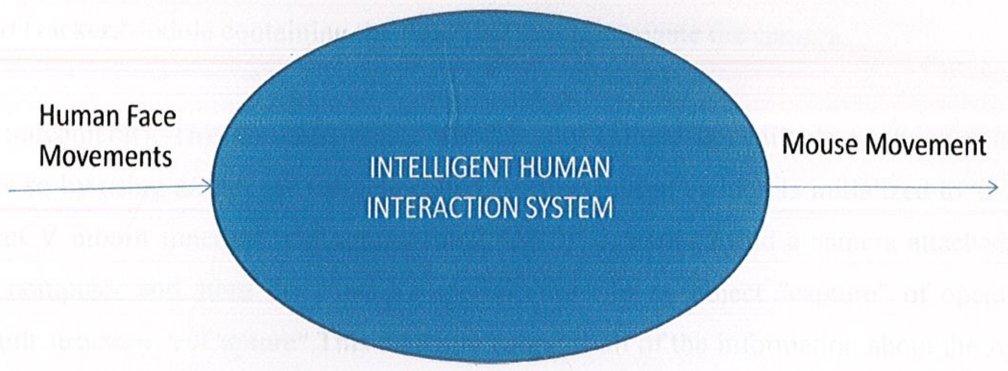
to be accessible to Java applications. Many of the standard library classes depend on JNI to provide functionality to the developer and the user, e.g. file I/O and sound capabilities. Including performance- and platform-sensitive API implementations in the standard library allows all Java applications to access this functionality in a safe and platform-independent manner. Before resorting to using JNI, developers should make sure the functionality is not already provided in the standard libraries.

The JNI framework lets a native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code.

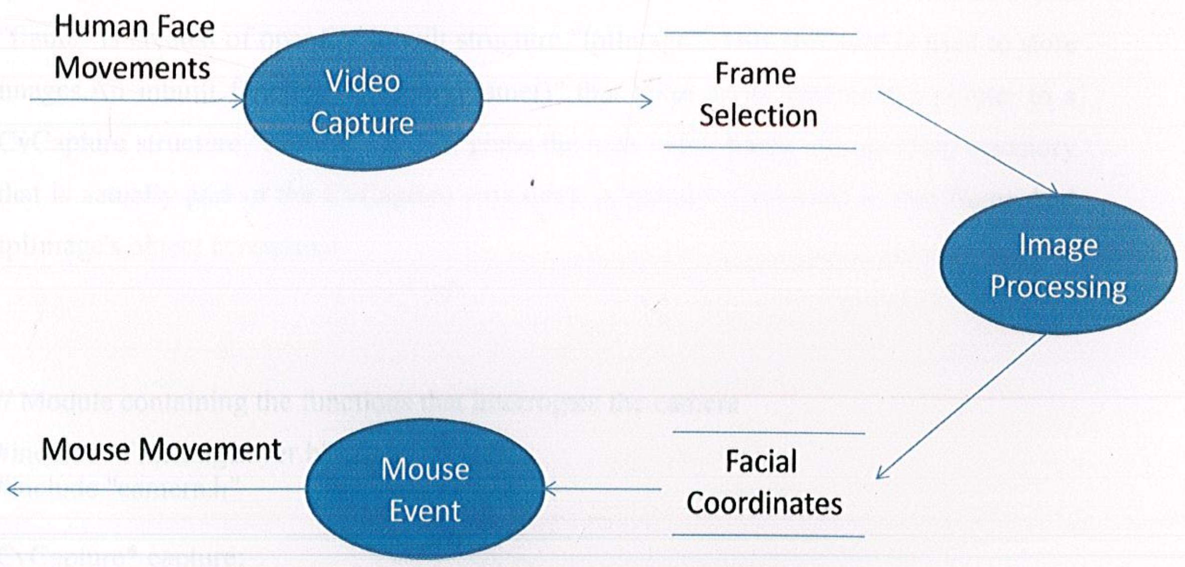
JNI is sometimes referred to as the "escape hatch" for Java developers because it allows them to add functionality to their Java application that the standard Java APIs cannot otherwise provide. It can be used to interface with code written in other languages, such as C and C++. It is also used for time-critical calculations or operations like solving complicated mathematical equations, since native code can be faster than JVM code.

4.1Data Flow Diagram

Level 0 DFD



Level 1 DFD



4.2 Camera Functions

Camera.cpp :-

StartTracker:Module containing the functions that interrogate the camera.

int initcamera():-This function detect whether any camera is available to the system.It does so by using an integer variable called "successfullInit" which is initialized to '0'.An openCV inbuilt function "cvCaptureFromCAM(0)" is used to find a camera attached to the computer and store the camera's specifications in an object "capture" of openCV inbuilt structure "cvCapture".This structure contains all of the information about the AVI file being read, including state information. When created in this way, the CvCapture structure is initialized to the beginning of the AVI.

If the camera is found the "successfullinit" variable is changed to '1' else it returns '0'.

IplImage* initFaceIdentifier():-This function is used to initialize the system.An object "frame" is created of openCV inbuilt structure "IplImage" .This structure is used to store images.An inbuilt function "cvQueryFrame()" that takes as its argument a pointer to a CvCapture structure "capture". It then grabs the next video frame into memory (memory that is actually part of the CvCapture structure). A pointer is returned to that frame.And IplImage's object is returned.

```
// Module containing the functions that interrogate the camera
#include "TrackingLayer.h"
#include "camera.h"

CvCapture* capture;
/**< stores the connection with the camera */
/**
* Initialize the camera */
int initCamera(){
        int successfullInit = 0;
        //use the first camera found
        capture = cvCaptureFromCAM(0);

        if(capture)
                successfullInit = 1;

        return successfullInit;
}

/**
* Get the current image from the camera
* \n The camera needs to be initialized from \ref initCamera()
* \return the pointer to the image retrieved
*/
IplImage* getImage( ) {
        IplImage* frame = 0;
   frame = cvQueryFrame( capture );

        return frame;
}

// Terminates the camera

void closeCamera(){
        cvReleaseCapture( &capture ); //Release the captured memory
}
```

// Header of \ref camera.cpp

```
int initCamera();
IplImage* getImage();
void closeCamera();
```

4.3 Configurator

It containing the functions to set the configuration of the object identified by the camera. It is used to identify features of the object captured like the max feat , EyeErosion, Eyeframegap, Eyemax relations.

This C++ program includes “TrackingLayer.h” and "configurator.h" header files.

Configurator.cpp

```
/*
*  Module containing the functions to set the configuration

*/

#include "TrackingLayer.h"
#include "configurator.h"

//declaration of the default configuration variables
int DEBUG = 0;
int FACE_TRIM = 25;
int MAX_FEAT = 20;
double FEAT_QUALITY = 0.15;
int MIN_FEAT_DISTANCE = 5;
int WIN_SIZE_W = 10;
int WIN_SIZE_H = 10;
int TRACKER_ERROR = 320;
int EYES_EROSION = 1;
int EYES_TRESHOLD = 20;
int EYES_MAX_ANGLE = 10;
int EYES_FRAMES_GAP = 1;
int EYES_PIX_RATIO = 20;
int EYES_MAX_RELATIONS = 30;
int ACTION_NUM_MOVEMENTS = 2;
int ACTION_FRAMES = 15;

jclass getConfClass(JNIEnv *env){
        jclass confClazz = env->FindClass("clientlayer/Configuration");
        assert(confClazz != NULL); //be sure that the class has been found
        return confClazz;
}
void setConfiguration(JNIEnv *env, jobject configuration){
        jclass configurationClass = getConfClass(env);
        jfieldID tempField = 0;
        jint tempResultInt = 0;
        jdouble tempResultDouble = 0;
        jobject tempResultString = 0;

        //Set DEBUG
        tempField = env->GetFieldID(configurationClass, "debug", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        DEBUG = tempResultInt;

        //Set FACE_TRIM
        tempField = env->GetFieldID(configurationClass, "faceTrim", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        FACE_TRIM = tempResultInt;

        //Set MAX_FEAT
        tempField = env->GetFieldID(configurationClass, "maxFeat", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        MAX_FEAT = tempResultInt;

        //Set FEAT_QUALITY
        tempField = env->GetFieldID(configurationClass, "featQuality", "D");
        tempResultDouble = env->GetDoubleField(configuration, tempField);
        FEAT_QUALITY = tempResultDouble;

        //Set MIN_FEAT_DISTANCE
        tempField = env->GetFieldID(configurationClass, "minFeatDistance", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        MIN_FEAT_DISTANCE = tempResultInt;

        //Set WIN_SIZE_W
        tempField = env->GetFieldID(configurationClass, "winSizeW", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        WIN_SIZE_W = tempResultInt;

        //Set WIN_SIZE_H
```

```
tempField = env->GetFieldID(configurationClass, "winSizeH", "I");
tempResultInt = env->GetIntField(configuration, tempField);
WIN_SIZE_H = tempResultInt;

//Set TRACKER_ERROR
tempField = env->GetFieldID(configurationClass, "trackerError", "I");
tempResultInt = env->GetIntField(configuration, tempField);
TRACKER_ERROR = tempResultInt;

//Set EYES_EROSION
tempField = env->GetFieldID(configurationClass, "eyesErosion", "I");
tempResultInt = env->GetIntField(configuration, tempField);
EYES_EROSION = tempResultInt;

//Set EYES_TRESHOLD
tempField = env->GetFieldID(configurationClass, "eyesThreshold", "I");
tempResultInt = env->GetIntField(configuration, tempField);
EYES_TRESHOLD = tempResultInt;

//Set EYES_MAX_ANGLE
tempField = env->GetFieldID(configurationClass, "eyesMaxAngle", "I");
tempResultInt = env->GetIntField(configuration, tempField);
EYES_MAX_ANGLE = tempResultInt;

//Set EYES_FRAMES_GAP
tempField = env->GetFieldID(configurationClass, "eyesFramesGap", "I");
tempResultInt = env->GetIntField(configuration, tempField);
EYES_FRAMES_GAP = tempResultInt;

//Set EYES_PIX_RATIO
tempField = env->GetFieldID(configurationClass, "eyesPixRatio","I");
tempResultInt = env->GetIntField(configuration, tempField);
EYES_PIX_RATIO = tempResultInt;

//Set EYES_MAX_RELATIONS
tempField = env->GetFieldID(configurationClass, "eyesMaxRelation", "I");
tempResultInt = env->GetIntField(configuration, tempField);
EYES_MAX_RELATIONS = tempResultInt;

//Set ACTION_NUM_MOVEMENTS
```

```
        tempField = env->GetFieldID(configurationClass, "actionNumMovements", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        ACTION_NUM_MOVEMENTS = tempResultInt;

        //Set ACTION_FRAMES
        tempField = env->GetFieldID(configurationClass, "actionFrames", "I");
        tempResultInt = env->GetIntField(configuration, tempField);
        ACTION_FRAMES = tempResultInt;
}

jobject getConfiguration(JNIEnv *env){
        jclass configurationClass = getConfClass(env);
        //allocate a configuration object without calling its constructor;
        jobject currentConf = env->AllocObject(configurationClass);
        assert(currentConf!=NULL); //make sure that the object can be created
        jfieldID tempField = 0;

        //Get DEBUG
        tempField = env->GetFieldID(configurationClass, "debug", "I");
        env->SetIntField(currentConf, tempField, DEBUG);

        //Get FACE_TRIM
        tempField = env->GetFieldID(configurationClass, "faceTrim", "I");
        env->SetIntField(currentConf, tempField, FACE_TRIM);

        //Get MAX_FEAT
        tempField = env->GetFieldID(configurationClass, "maxFeat", "I");
        env->SetIntField(currentConf, tempField, MAX_FEAT);

        //Get FEAT_QUALITY
        tempField = env->GetFieldID(configurationClass, "featQuality", "D");
        env->SetDoubleField(currentConf, tempField, FEAT_QUALITY);

        //Get MIN_FEAT_DISTANCE
        tempField = env->GetFieldID(configurationClass, "minFeatDistance", "I");
        env->SetIntField(currentConf, tempField, MIN_FEAT_DISTANCE);

        //Get WIN_SIZE_W
        tempField = env->GetFieldID(configurationClass, "winSizeW", "I");
        env->SetIntField(currentConf, tempField, WIN_SIZE_W);
```

```
//Get WIN_SIZE_H
tempField = env->GetFieldID(configurationClass, "winSizeH", "I");
env->SetIntField(currentConf, tempField, WIN_SIZE_H);

//Get TRACKER_ERROR
tempField = env->GetFieldID(configurationClass, "trackerError", "I");
env->SetIntField(currentConf, tempField, TRACKER_ERROR);

//Get EYES_EROSION
tempField = env->GetFieldID(configurationClass, "eyesErosion", "I");
env->SetIntField(currentConf, tempField, EYES_EROSION);

//Get EYES_TRESHOLD
tempField = env->GetFieldID(configurationClass, "eyesThreshold", "I");
env->SetIntField(currentConf, tempField, EYES_TRESHOLD);

//Get EYES_MAX_ANGLE
tempField = env->GetFieldID(configurationClass, "eyesMaxAngle", "I");
env->SetIntField(currentConf, tempField, EYES_MAX_ANGLE);

//Get EYES_FRAMES_GAP
tempField = env->GetFieldID(configurationClass, "eyesFramesGap", "I");
env->SetIntField(currentConf, tempField, EYES_FRAMES_GAP);

//Get EYES_PIX_RATIO
tempField = env->GetFieldID(configurationClass, "eyesPixRatio", "I");
env->SetIntField(currentConf, tempField, EYES_PIX_RATIO);

//Get EYES_MAX_RELATIONS
tempField = env->GetFieldID(configurationClass, "eyesMaxRelation", "I");
env->SetIntField(currentConf, tempField, EYES_MAX_RELATIONS);

//Get ACTION_NUM_MOVEMENTS
tempField = env->GetFieldID(configurationClass, "actionNumMovements", "I");
env->SetIntField(currentConf, tempField, ACTION_NUM_MOVEMENTS);
//Get ACTION_FRAMES
tempField = env->GetFieldID(configurationClass, "actionFrames", "I");
env->SetIntField(currentConf, tempField, ACTION_FRAMES);
return currentConf;}
```

Configurator.h

```
/*
*  Header of \ref configurator.cpp

*/

#include <jni.h>

/**
* Set the new configuration passed as a parameter
* @param obj the new configuration
*/
void setConfiguration(JNIEnv *env, jobject obj);
/**
* Create a configuration object with the current configuration
* @return current configuration
*/
jobject getConfiguration(JNIEnv *env);
```

4.4 Configuration.h

It is a header file which contains the configuration values for the TrackingLayer. It defines various default configuration variables used in Configuration.cpp.

This program calls **java function 'configuration'** .once the link is set then java function is used to configure the tracking layer.

Configuration.h

```
/*
*  Contains the configuration values for the TrackingLayer
*/
#ifndef CONF_SETTINGS
#define CONF_SETTINGS

extern int DEBUG;
/**< If it is set to 1 it will display a window with the result of the algorithms */

extern int FACE_TRIM;
/**< How much trim the edge of the face in percentage */

extern int MAX_FEAT;
/**< Max number of face features used to track the face movements */

extern double FEAT_QUALITY;
/**< Multiplier for the maxmin eigenvalue; specifies minimal accepted quality of image feature. */

extern int MIN_FEAT_DISTANCE;
/**< Define the minimum distance in pixel between the features found */

extern int WIN_SIZE_W;
/**< Width of the window size for the tracker and the feature refinement */

extern int WIN_SIZE_H;
/**< Height of the window size for the tracker and the feature refinement */

extern int TRACKER_ERROR;
/**<  the accuracy of the error detection in pixel (the smaller the value the more accurate it is) */
```

```
extern int EYES_EROSION;
/**< Number of iteration of erosion of the image of the eyes */

extern int EYES_TRESHOLD;
/**< It describe the threshold level of the eyes. The smaller the value more value are displayed */

extern int EYES_MAX_ANGLE;
/**< Maximum angle allowed to the axis that link the two eyes. It is used for the initial identification*/

extern int EYES_FRAMES_GAP;
/**< Frames compared to retrieve an eye blink action */

extern int EYES_PIX_RATIO;
/**< The minimum number of pixel necessary in the eyes area to be considered as a possible blink
* (expressed as a ratio with the total eye window area, i.e. the higher = the less pixel required) */

extern int EYES_MAX_RELATIONS;
/**< The maximum number of relations allowed between eyes */

extern int ACTION_NUM_MOVEMENTS;
/**< The number of consecutive eyes movements required required to trigger an action */

extern int ACTION_FRAMES;
/**< The maximum frame gap allowed between two consecutive blinks */

#endif /* CONF_SETTINGS */


#define CASCADE_LOCATION "res\\haarcascade_frontalface.xml"
/**< Location of the Trained Cascade to use */
#define FEAT_REFINEMENT 1
/**< Perform the feature refinement */
#define FEAT_REFINEMENT_CRITERIA
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03)
/**< Criteria for termination of the iterative process of corner refinement */
#define TRACKER_PYR_NUM 3
/**< Maximal pyramid level number for the tracker */
#define TRACKER_CRITERIA
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03)
```

```
/**< Criteria for termination of the iterative process of tracking */
#define TRACKER_FLAGS 0
/**< Tracking flags */
#define WINDOWS_NAME "Debug"
/**< Title of the debug window */
//#define SHOW_POINTS
/**< if defined shows the points tracked in a window */
//#define SHOW_MOVEMENTS
/**< if defined shows the hypotetic movements of the cursor */
```

4.5 EyesTracking

<u>eyesTracking.cpp:-</u>

It contains the functions to track the eyes position and the eye blinks.

It includes ‘Trackinglayer.h’ and ‘eyesTracking.h’ header files.

It uses various variables and buffers to store initial and final eye coordinates.

Eg: CvRect *initialLeftEye , *initialRightEye.

The difference of coordinates and angle is calculated between right eye and left eye.

The function used to do so are:

void eyesTrackerInit(CvRect *leftEyeIn, CvRect *rightEyeIn, IplImage* frame)

void setEyesDebugImg(IplImage* image)

int compareRect(CvRect *tempBox1, CvRect *tempBox2)

CvPoint getRectCenter(CvRect *tempBox1)

void removeUnfeasibleEyes(list < CvRect* > &lines)

void difference(IplImage* img1, IplImage* img2, IplImage* resultImage)

The integer variable “minimumWhitePixel” is used for the minimum number of pixel necessary in the eyes area to be considered as a possible blink in both eyes..

inline int checkEyeFeasibility(CvRect* eye) – Check if the eyes are still in the view of the camera. It return 0 if the eye is feasible, 1 otherwise.

Program then updates mouse coordinates keeping in consideration old mouse coordinates using function 'int updateEyesLocation' .This also checks if eyes are in the view of camera by using function:

int updateEyesLocation(int newX, int newY, int oldX, int oldY, IplImage *outputImage)

All this operations are done using image buffers IplIimage.

Function 'eyesTrackerInit' is one of the imp functions in this program it does following action in a sequence:

1. Camera resolution: It checks camera height and width.
2. eyes difference: It finds difference between initial image and final image using CvCreateImage.
3. eyes images buffers: It stores various eye details in variables or buffers.
4. Function 'detect Blink' is used to detect if there were any 'repeated blinking'.

<u>**'eyesTracking.h'**</u> is a header file which does

1. Update the location of the eyes.
2. Identify if a blink has occurred that may trigger an action.
3. Set the output image used for the debug process.

eyestracking.cpp

```
/*
* Module containing the functions to track the eyes position and the eye blinks
*/
#include "TrackingLayer.h"
#include "eyesTracking.h"

CvMemStorage* contStore = 0;
CvSeq* contourn = 0;
list < CvRect > objectsFound;
list < CvRect* > relationshipsFound;
```

```
#ifdef SHOW_POINTS
	IplImage* imageTempBlink = 0;
	IplImage* imageTempEyes = 0;
#endif

//difference variables (and buffers)
IplImage* img1BW = 0;
IplImage* img2BW = 0;
IplImage* imgRes = 0;
IplImage* imgResEroded = 0;

//eyes images buffers
IplImage* imgOldLeftEye = 0;
IplImage* imgActualLeftEye = 0;
IplImage* imgOldRightEye = 0;
IplImage* imgActualRightEye = 0;
IplImage* diffLeftEye = 0;
IplImage* diffRightEye = 0;
IplImage* eyesContournImgBuff = 0; //the image used to find the countourns

CvRect *initialLeftEye = 0, *initialRightEye = 0;
CvRect actualLeftEye, actualRightEye, oldLeftEye, oldRightEye;
//the old...eyes position must correspond to the old and new image parsed
int positionX = 0;
int positionY = 0;

int minimumWhitePixel = 0; //The minimum number of pixel necessary in the eyes area to
be considered as a possible blink in both eyes.

int CAMERA_HEIGHT_E = 0;
/**< It stores the height of the camera resolution */
int CAMERA_WIDTH_E = 0;
/**< It stores the width of the camera resolution */

IplImage *DEBUG_EYES_IMAGE = 0;

void eyesTrackerInit(CvRect *leftEyeIn, CvRect *rightEyeIn, IplImage* frame){
	//init cam resolution
	CAMERA_WIDTH_E = frame->width;
	CAMERA_HEIGHT_E = frame->height;
```

```
//init the countourn detection
contStore = cvCreateMemStorage(0);
contourn = cvCreateSeq(CV_SEQ_ELTYPE_POINT, sizeof(CvSeq), sizeof(CvPoint), contStore);

//init the eyes difference
img1BW = cvCreateImage( cvSize(leftEyeIn->width, leftEyeIn->height), IPL_DEPTH_8U, 1 );
img2BW = cvCreateImage( cvSize(leftEyeIn->width, leftEyeIn->height), IPL_DEPTH_8U, 1 );
imgResEroded = cvCreateImage( cvSize(leftEyeIn->width,leftEyeIn->height), IPL_DEPTH_8U, 1 );
imgRes = cvCreateImage( cvSize(leftEyeIn->width,leftEyeIn->height), img2BW->depth, img2BW-
>nChannels );

//eyes images buffers
imgOldLeftEye = cvCreateImage( cvSize(leftEyeIn->width,leftEyeIn->height),
frame->depth, frame->nChannels );
imgActualLeftEye = cvCreateImage( cvSize(leftEyeIn->width,leftEyeIn->height),
frame->depth,  frame->nChannels );
imgOldRightEye = cvCreateImage( cvSize(rightEyeIn->width,rightEyeIn->height),
frame->depth,  frame->nChannels );
imgActualRightEye = cvCreateImage( cvSize(rightEyeIn->width,rightEyeIn->height),
frame->depth,  frame->nChannels );

diffLeftEye = cvCreateImage( cvSize(leftEyeIn->width,leftEyeIn->height), IPL_DEPTH_8U, 1 );
diffRightEye = cvCreateImage( cvSize(rightEyeIn->width,rightEyeIn->height), IPL_DEPTH_8U, 1 );
eyesContournImgBuff = cvCreateImage( cvSize(rightEyeIn->width,rightEyeIn->height),
IPL_DEPTH_8U, 1 );

//init the eyes
initialLeftEye = leftEyeIn;
initialRightEye = rightEyeIn;

actualLeftEye = cvRect(leftEyeIn->x, leftEyeIn->y, leftEyeIn->width, leftEyeIn->height);
actualRightEye = cvRect(rightEyeIn->x, rightEyeIn->y, rightEyeIn->width, rightEyeIn->height);
oldLeftEye = cvRect(leftEyeIn->x, leftEyeIn->y, leftEyeIn->width, leftEyeIn->height);
oldRightEye = cvRect(rightEyeIn->x, rightEyeIn->y, rightEyeIn->width, rightEyeIn->height);

minimumWhitePixel = ((int)(leftEyeIn->width*leftEyeIn->height / EYES_PIX_RATIO))*2;
}

void setEyesDebugImg(IplImage* image){
        DEBUG_EYES_IMAGE = image;
}

int compareRect(CvRect *tempBox1, CvRect *tempBox2){
        int res = 0;
        assert(tempBox1);
        assert(tempBox2);
        if(tempBox1->height > tempBox2->height){
                res = 1;
        }else if(tempBox1->height < tempBox2->height){
                res = -1;
        }else if(tempBox1->width > tempBox2->width){
                res = 1;
        }else if(tempBox1->width < tempBox2->width){
                res = -1;
        }else if(tempBox1->x > tempBox2->x){
                res = 1;
        }else if(tempBox1->x < tempBox2->x){
                res = -1;
        }else if(tempBox1->y > tempBox2->y){
                res = 1;
        }else if(tempBox1->y < tempBox2->y){
                res = -1;
        }

        return res;
}

CvPoint getRectCenter(CvRect *tempBox1){
        assert(tempBox1);

        return cvPoint( (int)(tempBox1->x + (tempBox1->height / 2)),
                                        (int)(tempBox1->y + (tempBox1->width / 2)) );

}

void drawLines( list < CvRect* > &lines, IplImage* dstImage ){
        CvRect* tempLine;
        CvPoint center1, center2;
        for (list<CvRect*>::iterator it = lines.begin(); it!=lines.end(); ++it) {
                tempLine = *it;
                center1 = getRectCenter( &tempLine[0] );
                center2 = getRectCenter( &tempLine[1] );

                cvLine( dstImage,
                                center1,
                                center2,
                                CV_RGB(255,0,0)
                );
        }
}

void removeUnfeasibleEyes( list < CvRect* > &lines ){
        double pi = 3.1415926535;
        double angle;
        CvRect* tempLine;
        CvPoint center1, center2;

        list<CvRect*>::iterator it = lines.begin();
        while(it!=lines.end()){
                tempLine = *it;
                center1 = getRectCenter( &tempLine[0] );
                center2 = getRectCenter( &tempLine[1] );

                //calculate angle (in degrees)
                angle = atan( (float)
                                abs(center1.y - center2.y) /
                                abs(center1.x - center2.x)
                                )*180/pi;
                if( angle > EYES_MAX_ANGLE ) {
                        it = lines.erase(it);
                }else{
                        ++it;
                }
        }

}

void difference(IplImage* img1, IplImage* img2, IplImage* resultImage){
        cvCvtColor(img1, img1BW, CV_BGR2GRAY);
        cvCvtColor(img2, img2BW, CV_BGR2GRAY);
        cvAbsDiff( img1BW, img2BW,  imgRes );
        //cvSub( img1BW, img2BW,  imgRes );

        //erode image
        cvErode( imgRes, imgResEroded, NULL, EYES_EROSION );

        //treshold image
        cvCmpS( imgResEroded, EYES_TRESHOLD, resultImage, CV_CMP_GT  );
}

//the offset will be used to set the right coordinate on the eye
void getContourns(IplImage* image, CvPoint offset){

        cvCopy(image, eyesContournImgBuff);
   // Find all contours (it modifies the image contents).
   cvFindContours( eyesContournImgBuff, contStore, &contourn,
                                  sizeof(CvContour), CV_RETR_LIST,
CV_CHAIN_APPROX_NONE,
                                  offset );

//if a contourn has been found
        if( contourn ){
                bool getCountourns = true;
                while( getCountourns ) {
                        //create a bounding rectangle
                        CvRect areaIdentified = cvBoundingRect(contourn,0);
                        // put it in a vector list
                        objectsFound.push_back( areaIdentified );

                        //go to the next contourn and break the loop if necessary
                        if( !(contourn = contourn->h_next) ){
                                getCountourns = false;

                        }
                }

        }}
```

```
/**
* Check if the eyes are still in the view of the camera
* return 0 if the eye is feasible, 1 otherwise.
*/
inline int checkEyeFeasibility(CvRect* eye){
        int notFeasible = 0;
        if(eye->x <= 0 ||
        (eye->x+eye->width) >= CAMERA_WIDTH_E){
                notFeasible = 1;
        }
        if(eye->y <= 0 ||
        (eye->y+eye->height) >= CAMERA_HEIGHT_E){
                notFeasible = 1;
        }

        return notFeasible;
}

//updates the eyes position keeping in consideration the previous location
int updateEyesLocation(int newX, int newY,
                                                int oldX, int oldY,
                                                IplImage *outputImage) {
        int errors = 0;
        int failed = 0;
        int shiftNewX = positionX - newX;
        int shiftNewY = positionY - newY;
        int shiftOldX = positionX - oldX;
        int shiftOldY = positionY - oldY;

  //shift the eyes
        actualLeftEye.x = initialLeftEye->x - shiftNewX;
        actualLeftEye.y = initialLeftEye->y - shiftNewY;
        actualRightEye.x = initialRightEye->x - shiftNewX;
        actualRightEye.y = initialRightEye->y - shiftNewY;

        oldLeftEye.x = initialLeftEye->x - shiftOldX;
        oldLeftEye.y = initialLeftEye->y - shiftOldY;
        oldRightEye.x = initialRightEye->x - shiftOldX;
        oldRightEye.y = initialRightEye->y - shiftOldY;
```

```
            //check errors
            errors += checkEyeFeasibility(&actualLeftEye);
            errors += checkEyeFeasibility(&actualRightEye);
            errors += checkEyeFeasibility(&oldLeftEye);
            errors += checkEyeFeasibility(&oldRightEye);

            if(errors > 0){
                        failed = 1;
            }

            if(DEBUG){
                        //cvCopy(outputImage, imageTempEyes);
                        //draw the shifted eyes
            cvRectangle(DEBUG_EYES_IMAGE, cvPoint(actualLeftEye.x , actualLeftEye.y),
                                    cvPoint(actualLeftEye.x+actualLeftEye.width,
                                                actualLeftEye.y+actualLeftEye.height),
CV_RGB(0,0,255), 2, 8, 0 );
cvRectangle(DEBUG_EYES_IMAGE, cvPoint(actualRightEye.x , actualRightEye.y),
                        cvPoint(actualRightEye.x+actualRightEye.width,
                                    actualRightEye.y+actualRightEye.height),
                                    CV_RGB(0,0,255), 2, 8, 0 );
            }

            return failed;

}

int detectBlink(IplImage *image1, IplImage *image2, IplImage *outputImage) {
            static int blinkDetectionCounter = 1;
            static int actionGapCounter = 0;

            //difference of the image considering the left eye only
            cvResetImageROI(image1);
            cvResetImageROI(image2);

            cvSetImageROI( image1, oldLeftEye );

            cvCopy(image1, imgOldLeftEye);
            cvSetImageROI( image2, actualLeftEye );
            cvCopy(image2, imgActualLeftEye);
```

```
        difference(imgOldLeftEye, imgActualLeftEye, diffLeftEye);

        //difference of the image considering the right eye only
        cvResetImageROI(image1);
        cvResetImageROI(image2);

        cvSetImageROI( image1, oldRightEye );

        cvCopy(image1, imgOldRightEye);
        cvSetImageROI( image2, actualRightEye );
        cvCopy(image2, imgActualRightEye);
        difference(imgOldRightEye, imgActualRightEye, diffRightEye);

        //if there are enough white pixel
if((cvCountNonZero(diffRightEye) + cvCountNonZero(diffLeftEye)) > minimumWhitePixel)
{

                //get contourns for the left eye
                getContourns(diffLeftEye, cvPoint(actualLeftEye.x, actualLeftEye.y));
                //get contourns for the right eye
                getContourns(diffRightEye, cvPoint(actualRightEye.x, actualRightEye.y));

        CvRect *tempBox1, *tempBox2;
        //create all the possible pair of features
for (list<CvRect>::iterator it1 = objectsFound.begin(); it1!=objectsFound.end(); ++it1) {
for (list<CvRect>::iterator it2 = objectsFound.begin(); it2!=objectsFound.end(); ++it2) {
                CvRect* tempLine = (CvRect*) malloc(2*sizeof(CvRect));
                        tempBox1 = &(*it1);
                                tempBox2 = &(*it2);
                                if( compareRect(tempBox1, tempBox2) != 0 ){
                                //if the two elements are different
                                        tempLine[0] = *it1;
                                        tempLine[1] = *it2;
                                        relationshipsFound.push_back( tempLine );
                                }
                        }
                }
        }

        //set the action and blink as not detected
        int actionDetected = 0;
        int blinkDetected = 0;

//show eyes found if any relationship is detected
(discarding whenever there are too many of them)
if(relationshipsFound.size() > 0 && ((int)relationshipsFound.size()) <=
EYES_MAX_RELATIONS){

                //blink detected!!!!!!!!!!!!!!!!!!!
                blinkDetected = 1;

                if(DEBUG){
                        drawLines( relationshipsFound, DEBUG_EYES_IMAGE );
                }

        }


        if(blinkDetected == 0){
                ++actionGapCounter;
                if(actionGapCounter > ACTION_FRAMES){
                //if no blinks has been detected and the action gap has been exceded
                //reinitialize the whole array and restart the counter
                        blinkDetectionCounter = 1;
                        actionGapCounter = 0;
                }
        }else if(blinkDetectionCounter < ACTION_NUM_MOVEMENTS){
        //if a blink has been detected and it is not the triggering one
                ++blinkDetectionCounter;
                actionGapCounter = 0;
        }else{
        //if a blink has been detected and it is the triggering one
                blinkDetectionCounter = 1;
                actionDetected = 1;
                actionGapCounter = 0;
        }

        for (list<CvRect*>::iterator it = relationshipsFound.begin();
it!=relationshipsFound.end(); ++it) {
                CvRect* tempRect = *it;
                free(tempRect);
        }
        relationshipsFound.clear();

        cvResetImageROI(image1);
        cvResetImageROI(image2);
        objectsFound.clear();
        cvClearMemStorage(contStore);

        return actionDetected;
}
```

Eyestracking.h

```
/*
*  Header of \ref eyesTracking.cpp
*/
#include <list>
#include <math.h>
#include <string.h>

using namespace std;

void eyesTrackerInit(CvRect *leftEyeIn, CvRect *rightEyeIn, IplImage *frame);
/**
* Update the location of the eyes
* @return 0 if it is valid, 1 otherwise
*/
int updateEyesLocation(int newX, int newY, int oldX, int oldY, IplImage *outputImage);
/**
* Identify if a blink has occurred that may trigger an action.
* This Function needs to be called constantly
*/
int detectBlink(IplImage *image1, IplImage *image2, IplImage *outputImage
```

4.6 Face features Initializing

Face feature initialization :

1. It contains the functions to initialize the face area
2. It identifies the best facial features to use with the tracking,
3. Generates rough location of eyes.
4. Trim edges of faces to avoid bad features.

It does so by use of function Facefeatureinit() which finds the location of face by using a boolean variableinitsuccessul which is set to 1 if features are identified. the coordinates of face are stored in variable facelocation of cvpoint type.

after generating the coordinates of face it tries to find out the rough location of eyes by storing coordinates into variable lefteye and righteye.

After that it trims the edge of he face to avoid the bad features by using opencv function 'pixelstotrim'.it also tries to remove bad eye fearures so as to have maximum of only wanted features and remove unwanted ones.

After that it tries to find out best facial features to track by converting the saved image into gray scale and perform subpixel refinement on the search by using opencv function 'cvFindCornerSubPix'

faceFeatinit.cpp

```
/*
*  Module containing the functions for initialize the face area
*/

#include "TrackingLayer.h"
#include "faceFeatInit.h"

#define MIN_FEAT_NUM 3

/**
* Identify the best facial features to use with the tracking
* uses:
* @param imgToDetect image to analyse
* @param featuresPoints Pointer to image containing just the face
*/
void identifyBestFacialFeatures( IplImage* imgToDetect, CvPoint2D32f* featuresPoints );

/**
* Adjust the offset of the points
* @param points Pointer to an array of points to be adjusted
* @param numberOfPoints numberof points
* @param offset Pointer to a dtructure containing the offset
*/
void calculatePointsOffset( CvPoint2D32f* points, int numberOfPoints, CvPoint* offset );

/**
* Remove the features located in the eyes area
* @param[out] features face features
* @param[in] eyeLocation location of the eyes
* @param[out] number of features removed
*/
void removeEyesFeatures
(CvPoint2D32f* features, CvRect* eyeLocation, int* featuresRemovedNum);

static CvMemStorage* STORAGE = 0;
/**< Internal memory storage for  Object detection*/
static CvHaarClassifierCascade* CASCADE = 0;
/**< Internal training set storage  */

static int pointsCount;
/**< number of features tracked  */

bool faceFeatureInit
( IplImage* image, CvRect* faceRectangleInitial, CvPoint2D32f* featuresPoints,
 CvRect* leftEye, CvRect* rightEye ){
        bool initSuccessfull = true;
        CvPoint faceLocation = cvPoint(-1, -1);
        IplImage* faceOutput = 0;
        int facesNum = 0;
        CvSeq* faces = 0;
        pointsCount = MAX_FEAT;
        //printf("faceFeatureInit - 1\n");

        CvRect* faceRectangle = (CvRect*) malloc( sizeof(CvRect) );
        memcpy( faceRectangle, faceRectangleInitial, sizeof(CvRect) );

        //printf("faceFeatureInit - 2\n");

        //Check the number of faces found
//      facesNum = identifyFaces( image, &faces );


//      //generate an image of the face
//      if( facesNum == 1 ) {
                //get the eyes rough location
                int eyeDim = (int)(faceRectangle->width/4);
                int eyesY = (int)(faceRectangle->y +  (faceRectangle->height/4));
                int eye1X = (int)(faceRectangle->x +  (faceRectangle->width/5));
                int eye2X = (int)(faceRectangle->x +  (3*(faceRectangle->width/5)));

        leftEye->height = leftEye->width = rightEye->height = rightEye->width = eyeDim;
        leftEye->y = rightEye->y = eyesY;
        leftEye->x = eye1X;
        rightEye->x = eye2X;

                //trim the edge of the face to avoid bad features
                float floatingWidth = (float)faceRectangle->width;
                int pixelToTrim = (int)( floatingWidth / 100 * FACE_TRIM);
                faceRectangle->width = faceRectangle->width - (pixelToTrim*2);
                faceRectangle->x = faceRectangle->x + pixelToTrim;

                faceLocation.x = faceRectangle->x;
                faceLocation.y = faceRectangle->y;

        //printf("faceFeatureInit - 3\n");
        //create an image to store that face
        faceOutput = cvCreateImage( cvSize(faceRectangle->width, faceRectangle->height),
        image->depth, image->nChannels );
                //retrieve that image
                cvGetSubRect( image, (CvMat*) faceOutput, *faceRectangle );
        assert(faceOutput); //check if the image is generated

        //printf("faceFeatureInit - 4\n");
        identifyBestFacialFeatures( faceOutput, featuresPoints );

        //printf("faceFeatureInit - 5\n");
        calculatePointsOffset( featuresPoints, pointsCount, &faceLocation );

        //printf("faceFeatureInit - 6\n");
        //--remove the features in the eyes area and check if there are any feature left
        int featuresRemoved = 0;
        removeEyesFeatures(featuresPoints, leftEye, &featuresRemoved);
        removeEyesFeatures(featuresPoints, rightEye, &featuresRemoved);
        //--

        //printf("faceFeatureInit - 7\n");
        //--check how mant usable features are left
        int usableFeatures = 0;
        for(int i = 0; i<pointsCount; ++i){
                if(featuresPoints[i].x > 0 && featuresPoints[i].y > 0){
                        featuresPoints[usableFeatures].x = featuresPoints[i].x;
                        featuresPoints[usableFeatures].y = featuresPoints[i].y;
                        ++usableFeatures;
                }
        }
        pointsCount = usableFeatures;

        if(usableFeatures < MIN_FEAT_NUM){
    initSuccessfull = false;
                printf("Not enough features found\n");
        }else{
                //printf("faceFeatureInit - 8\n");
                // remove all unwanted features
                for(int i = usableFeatures; i<MAX_FEAT; ++i){
                        featuresPoints[i].x = -999;
                        featuresPoints[i].y = -999;
                }
        }
        //--
/*
```

```
		}else if(facesNum > 1) {
			initSuccessfull = false;
			printf("More than one face was found\n");
		}else {
			initSuccessfull = false;
			printf("No faces were found\n");
		}
*/

/*
		//todo: remove
		for( int i = 0; i < usableFeatures; i++ ) {
		if(featuresPoints[i].x > 0 && featuresPoints[i].y > 0){
		cvCircle( image, cvPointFrom32f(featuresPoints[i]), 1, CV_RGB(0,255,0), -1, 8, 0 );
			}
		}
		cvRectangle( image, cvPoint( faceRectangle->x, faceRectangle->y),
			cvPoint( faceRectangle->x+faceRectangle->width, faceRectangle-
>y+faceRectangle->height), cvScalar(255,0,0), 4);

		cvSaveImage( "faceRectangle.jpg", image );
		exit(0);
*/
		//printf("faceFeatureInit - 9\n");
		free( faceRectangle );

		//printf("faceFeatureInit - 10\n");

		return initSuccessfull;
}

void removeEyesFeatures
(CvPoint2D32f* features, CvRect* eyeLocation, int* featuresRemovedNum){
		int featuresRemoved = 0;
		bool initSuccessfull = true;

		for( int i = 0; i<pointsCount; ++i ) {
			//if the feature is inside the eyeLocation tag it as an unavailable feature
		if( features[i].x >= 0 &&
(features[i].x >= eyeLocation->x && features[i].x <= (eyeLocation->x + eyeLocation->width))
 &&
(features[i].y >= eyeLocation->y && features[i].y <= (eyeLocation->y + eyeLocation->height))
		){
			features[i].x = -1;
			features[i].y = -1;
		}
	}

	featuresRemovedNum += featuresRemoved;

}

void calculatePointsOffset( CvPoint2D32f* points, int numberOfPoints, CvPoint* offset ){
	for( int i = 0; i<numberOfPoints; ++i ) {
		points[i].x = offset->x + points[i].x;
		points[i].y = offset->y + points[i].y;
	}
}

void initFaceAreaIdentifier(){
   CASCADE = (CvHaarClassifierCascade*)cvLoad( CASCADE_LOCATION , 0, 0, 0 );
   STORAGE = cvCreateMemStorage(0);
}

int identifyFaces( IplImage* image, CvSeq** faces ){
	//check if it has been identified
	assert(CASCADE);
	assert(STORAGE);

   cvClearMemStorage( STORAGE );

	//Detect faces in the image
   *faces = cvHaarDetectObjects( image, CASCADE, STORAGE,
				1.1, 2, CV_HAAR_DO_CANNY_PRUNING,
				cvSize(40, 40) );
	return (*faces)->total;

 }
```

```
void identifyBestFacialFeatures( IplImage* imgToDetect, CvPoint2D32f* featuresPoints ){
        IplImage* imgToDetectGray = 0;
        IplImage* imgEig = 0;
        IplImage* imgTemp = 0;

        imgToDetectGray = cvCreateImage(cvGetSize(imgToDetect), IPL_DEPTH_8U, 1);
        imgEig = cvCreateImage(cvGetSize(imgToDetect), IPL_DEPTH_32F, 1);
        imgTemp = cvCreateImage(cvGetSize(imgToDetect),IPL_DEPTH_32F, 1);

        cvCvtColor( imgToDetect, imgToDetectGray, CV_BGR2GRAY ); //convert to grayscale

        //find out the "GoodFeaturesToTrack"
        cvGoodFeaturesToTrack (imgToDetectGray, imgEig, imgTemp, featuresPoints,
        &pointsCount, FEAT_QUALITY, MIN_FEAT_DISTANCE, 0);

        if( FEAT_REFINEMENT ){
                //perform subpixel refinement on the search
                cvFindCornerSubPix( imgToDetectGray, featuresPoints,
                                                        pointsCount, cvSize(WIN_SIZE_W,
WIN_SIZE_H) , cvSize(-1,-1),
                                                        FEAT_REFINEMENT_CRITERIA );
        }

        cvReleaseImage( &imgToDetectGray );
        cvReleaseImage( &imgEig );
        cvReleaseImage( &imgTemp );
}

void stopFaceAreaIdentifier( ){
        cvReleaseMemStorage(&STORAGE);
}
```

faceFeatinit.h

```
/*
*  Header of \ref faceFeatInit.cpp

*/


/**
* Initialize identifier. (It allocates all the memory used)
*/
void initFaceAreaIdentifier( );

/**
* Stop identifier. (It releases all the memory used)
*/
void stopFaceAreaIdentifier( );
/**
* Initialize the face detection
* @param[in] image image to analyse
* @param[out] featuresPoints array of points representing the features
* @param[out] leftEye left eyes area
* @param[out] rightEye right eyes area
* @return true if the initialisation is successfull
*/
bool faceFeatureInit( IplImage* image, CvRect* faceRectangle, CvPoint2D32f* featuresPoints,
CvRect* leftEye, CvRect* rightEye );

/**
* Identify Faces
* uses:
* @param image image to analyse
* @param faces a pointer to the contourn of thefaces identified
* returns the number of faces found
*/
int identifyFaces( IplImage* image, CvSeq** faces );
```

4.7 FaceTracking

It contains the functions for tracking the face. It works similar to eyetracking.cpp but here it uses features of particular face more to track it.

'faceTracking.h' it performs face tracking using the given featuresPoints.

Facetracking.cpp

```
/*
* Module containing the functions for track the face

*/

#include "TrackingLayer.h"
#include "faceTracking.h"

//TODO: to be moved
#define MAX_MOV 2000

CvPoint2D32f *initialPoints;
/**< array of points representing the very initial position */
IplImage *newImageGrey = 0, *oldImageGrey = 0, *newImageBuffer = 0, *oldImageBuffer = 0,
*imageSwap;
CvPoint2D32f* features[2] = {0,0};
CvPoint2D32f* featuresSwap;
char *featureStatus;
float *featureMov;
int currentFeatureTrackedNum = 0;
int trackerFlags = 0;
CvMemStorage* errorTrackingStore = 0;
CvSeq* errorTrackingCount = 0;

IplImage *DEBUG_FACE_IMAGE = 0;
/**< It stores the image used to show the inner working of the algorithms */
int CAMERA_HEIGHT_F = 0;
/**< It stores the height of the camera resolution */
int CAMERA_WIDTH_F = 0;
/**< It stores the width of the camera resolution */
```

```
void trackerInit( IplImage* imageIn, CvPoint2D32f* featurePointsIn ) {
        //init cam resolution
        CAMERA_WIDTH_F = imageIn->width;
        CAMERA_HEIGHT_F = imageIn->height;

        //initialize images buffer
        newImageGrey = cvCreateImage( cvGetSize(imageIn), 8, 1 );
        oldImageGrey = cvCreateImage( cvGetSize(imageIn), 8, 1 );
        newImageBuffer = cvCreateImage( cvGetSize(imageIn), 8, 1 );
        oldImageBuffer = cvCreateImage( cvGetSize(imageIn), 8, 1 );
        //initialize feature buffer
        features[0] = (CvPoint2D32f*)malloc(MAX_FEAT*sizeof(features[0][0]));
        features[1] = (CvPoint2D32f*)malloc(MAX_FEAT*sizeof(features[0][0]));
        featureStatus = (char*)malloc(MAX_FEAT*sizeof(char));
        featureMov =  (float*)malloc(MAX_FEAT*sizeof(char));
        trackerFlags = 0;
        //initialized the points used for the movements
        initialPoints = (CvPoint2D32f*)malloc(MAX_FEAT*sizeof(CvPoint2D32f));

        //initialize feature
        currentFeatureTrackedNum = 0;
        for(int i=0;i<MAX_FEAT;++i){
                if(featurePointsIn[i].x > 0 && featurePointsIn[i].y > 0){
                        features[0][currentFeatureTrackedNum] = featurePointsIn[i];
                        initialPoints[currentFeatureTrackedNum] = featurePointsIn[i];
                        featureStatus[currentFeatureTrackedNum] = 1;
                        featureMov[currentFeatureTrackedNum] = 0;
                        ++currentFeatureTrackedNum;


                        printf("x: %f, y: %f\n", featurePointsIn[i].x, featurePointsIn[i].y); //TODO:
remove
                }
        }

        // init old image
        cvCvtColor( imageIn, oldImageGrey, CV_BGR2GRAY );

        //initialize the error tracking sequences
        errorTrackingStore = cvCreateMemStorage(0);
}


void setFaceDebugImg(IplImage* image){
        DEBUG_FACE_IMAGE = image;
}

void drawPoints( IplImage* imgToDetect, CvPoint2D32f* featuresPoints, int
currentFeatureTrackedNum){
        for( int i = 0; i < currentFeatureTrackedNum; i++ ) {
                if(featuresPoints[i].x > 0 && featuresPoints[i].y > 0){
                        cvCircle( imgToDetect, cvPointFrom32f(featuresPoints[i]), 1, CV_RGB(0,255,0
-1, 8, 0 );
                }
        }
}


CvPoint2D32f* faceTracking( IplImage* imageIn ){

   cvCvtColor( imageIn, newImageGrey, CV_BGR2GRAY );

        /*
        //TODO: remove
        static int aaa2 = 0;
        char strFile1[255];
        char strFile2[255];
        sprintf( strFile1, "oldImageGrey%d-%dx%d.jpg",  aaa2, cvGetImageROI(oldImageGrey).width
cvGetImageROI(oldImageGrey).height );
        sprintf( strFile2, "newImageGrey%d-%dx%d.jpg",  aaa2,
cvGetImageROI(newImageGrey).width, cvGetImageROI(newImageGrey).height );

        cvSaveImage( strFile1, oldImageGrey );
        cvSaveImage( strFile2, newImageGrey );
        aaa2++;
*/
        int featuresNumber = currentFeatureTrackedNum;

        cvCalcOpticalFlowPyrLK( oldImageGrey, newImageGrey, oldImageBuffer, newImageBuffer,
                    features[0], features[1], featuresNumber, cvSize(WIN_SIZE_W, WIN_SIZE_H),
TRACKER_PYR_NUM, featureStatus, featureMov,
                    TRACKER_CRITERIA, trackerFlags );

        trackerFlags |= CV_LKFLOW_PYR_A_READY; //pyramid for the first frame is precalculated
before the call

/*
                    //TODO: remove
                    static FILE* fOut = fopen("points2.txt", "a+");
                    fprintf( fOut, "-----------points %d:\n\n",featuresNumber );
                    for( int i = 0; i<featuresNumber; ++i ) {
                            //if( features[1][i].x >= 0 ){
                                    fprintf( fOut, "x:%f \ty:%f -> x:%f \ty:%f \n", features[0][i].x,
features[0][i].y, features[1][i].x, features[1][i].y );
                                    fprintf( fOut, "featureStatus:%d \tmovement:%f\n", featureStatus[i],
featureMov[i]);
                            //}
                    }
                    fflush(fOut);
*/

        // Check and reset bad features
        CvPoint2D32f* featuresTmp = (CvPoint2D32f*) malloc(sizeof(CvPoint2D32f)*MAX_FEAT);
        currentFeatureTrackedNum = 0;
        for( int i = 0; i<featuresNumber; ++i ) {
                if( features[1][i].x >= 0 && features[1][i].y >= 0
                        && featureStatus[i] > 0 && featureMov[i] <= MAX_MOV ){
                        featuresTmp[currentFeatureTrackedNum].x = features[1][i].x;
                        featuresTmp[currentFeatureTrackedNum].y = features[1][i].y;
                        ++currentFeatureTrackedNum;
                }
        }
        for( int i = 0; i<currentFeatureTrackedNum; ++i ) {
                features[1][i].x = featuresTmp[i].x;
                features[1][i].y = featuresTmp[i].y;
        }
        free(featuresTmp);
        //
        if(DEBUG){
                drawPoints(DEBUG_FACE_IMAGE, features[1], currentFeatureTrackedNum);

                //TODO: remove
                //printf("good features Tracked: %d", currentFeatureTrackedNum);

/*

                static int aaa = 0;
                char strFile[255];
                sprintf( strFile, "errorDetected%d-%d.jpg",  aaa, featuresNumber );
                cvSaveImage( strFile, DEBUG_FACE_IMAGE );
                aaa++;

                //TODO: remove
                static FILE* fOut = fopen("points.txt", "a+");
                fprintf( fOut, "-----------points:\n\n" );
                for( int i = 0; i<featuresNumber; ++i ) {
                        //if( features[1][i].x >= 0 ){
                                fprintf( fOut, "x:%f \ty:%f -> x:%f \ty:%f \n", features[0][i].x,
features[0][i].y, features[1][i].x, features[1][i].y );
                        //}
                }
                fflush(fOut);
*/

        }

        CV_SWAP( oldImageGrey, newImageGrey, imageSwap );
        CV_SWAP( oldImageBuffer, newImageBuffer, imageSwap );
        CV_SWAP( features[0], features[1], featuresSwap );

        //printf("features: %d\n", featuresNumber);


        return features[1];
}

void interpretMovementsAverage(CvPoint2D32f* currentFeaturesPointsIn, float* x, float* y){
        float totalXmovements = 0;
        float totalYmovements = 0;

        if(currentFeatureTrackedNum > 0) {
                for( int i = 0; i < currentFeatureTrackedNum; i++ ) {
                        totalXmovements += currentFeaturesPointsIn[i].x - initialPoints[i].x ;
                        totalYmovements += currentFeaturesPointsIn[i].y - initialPoints[i].y ;
                }

                totalXmovements = (float)(totalXmovements / currentFeatureTrackedNum);
                totalYmovements = (float)(totalYmovements / currentFeatureTrackedNum);
                *x = totalXmovements;
                *y = totalYmovements;
        }
}
int checkTrackingErrors(CvPoint2D32f* featuresPointsIn, CvRect* initialFace,
                                IplImage* tempImage, IplImage* debugImage){
/*
        static FILE* fOut = fopen("points.txt", "a+");
fprintf( fOut, "-----------rect: %d x %d, w: %d, h: %d\n\n", initialFace->x, initialFace->y,
initialFace->width, initialFace->height );
                fprintf( fOut, "-----------points:\n\n" );
                for( int i = 0; i<MAX_FEAT; ++i ) {
        //if( features[1][i].x >= 0 ){    fprintf( fOut, "x:%f \ty:%f\n", featuresPointsIn[i].x,
featuresPointsIn[i].y );
                //}
                }

                fflush(fOut);
*/
        //calculate the rectangle of the points tracked
        errorTrackingCount = cvCreateSeq( CV_SEQ_KIND_GENERIC|CV_32SC2,
sizeof(CvContour),
                        sizeof(CvPoint), errorTrackingStore );

        //CvMat* featTracked = cvCreateMat( currentFeatureTrackedNum, 2, CV_32FC1 );

        CvPoint pt0;
        int highX = -1;
        int highY = -1;
        int lowX = 999;
        int lowY = 999;
        for( int i = 0; i < currentFeatureTrackedNum; i++ ) {
                pt0 = cvPointFrom32f(featuresPointsIn[i]);
                cvSeqPush( errorTrackingCount, &pt0 );
                //determine the location of the face of the user
                if(pt0.x > highX){
                        highX = pt0.x;
                }
                if(pt0.x < lowX){
                        lowX = pt0.x;
                }
                if(pt0.y > highY){
                        highY = pt0.y;
                }
                if(pt0.y < lowY){
                        lowY = pt0.y;
                }

                //printf( "x: %d, y: %d\n", pt0.x, pt0.y );
        }
        //CvBox2D trackedPointsBox = cvMinAreaRect2( errorTrackingCount, 0 ); //original
        //CvRect trackedPointsRect = cvBoundingRect( errorTrackingCount, 0 );

    static CvPoint2D32f center;
        static CvPoint icenter;
    static float radius;
        if(currentFeatureTrackedNum > 1){
                int res = cvMinEnclosingCircle( errorTrackingCount, &center, &radius );

                if( res != 0 ){
                        icenter.x = cvRound(center.x);
                        icenter.y = cvRound(center.y);
cvCircle( debugImage, icenter, cvRound(radius), CV_RGB(255, 255, 0), 1, CV_AA, 0 );
                        printf("cvMinEnclosingCircle found\n");
                }else{
                        printf("cvMinEnclosingCircle not found\n");

                        for( int i = 0; i < currentFeatureTrackedNum; i++ ) {
                                pt0 = cvPointFrom32f(featuresPointsIn[i]);
                        printf( "x: %d, y: %d\n", pt0.x, pt0.y );
                    }

                    cvWaitKey(0);
                }
            }


/*
            CvPoint tl;
            tl.x = trackedPointsBox.center.x - (trackedPointsBox.size.width/2);
            tl.y = trackedPointsBox.center.y - (trackedPointsBox.size.height/2);
            CvPoint br;
            br.x = trackedPointsBox.center.x + (trackedPointsBox.size.width/2);
            br.y = trackedPointsBox.center.y + (trackedPointsBox.size.height/2);
*/
/*
            CvPoint tl;
            CvPoint br;
            tl.x = trackedPointsRect.x;
            tl.y = trackedPointsRect.y;
            br.x = trackedPointsRect.x + trackedPointsRect.width;
            br.y = trackedPointsRect.y + trackedPointsRect.height;

            cvDrawRect( debugImage,  tl, br, cvScalar( 255,0,0 ), 1 );
*/

            //track a possible unfeasible set of features
            int errorTracked = 0;
/*

            int boxWidth, boxHeight;
            if(trackedPointsBox.size.height > trackedPointsBox.size.width){
                        boxWidth = (int)trackedPointsBox.size.width;
                        boxHeight = (int)trackedPointsBox.size.height;
            }else{
                        boxWidth = (int)trackedPointsBox.size.height;
                        boxHeight = (int)trackedPointsBox.size.width;
            }
```

```
        if( boxWidth > (initialFace->width +TRACKER_ERROR) ){
                errorTracked = 1;
        }
        if( boxHeight > (initialFace->height + TRACKER_ERROR) ){
                errorTracked = 1;
        }

        //check if the user is going away
        if(highX >= CAMERA_WIDTH_F || lowX <= 0 ||
                highY >= CAMERA_HEIGHT_F || lowY <= 0){
                errorTracked = 1;
        }
*/
        cvClearMemStorage(errorTrackingStore);

        // check to have at a feature to track
        if(currentFeatureTrackedNum < 1){
                errorTracked = 1;
        }

        return errorTracked;
}

void stopTracker(){
        free(features[0]);
        free(features[1]);
        free(featureMov);
        free(featureStatus);
        cvReleaseMemStorage(&errorTrackingStore);
}
```

Facetracking.h

```
/*
*  Header of \ref faceTracking.cpp
*/
/**
* Perform face tracking using the given featuresPoints
* @param image image to analyse
* @param featuresPoints array of points representing the features
*/
void trackerInit( IplImage* image, CvPoint2D32f* featuresPoints );

/**
* Perform face tracking using the internal features buffer
* trackerInit has to be called before this one.
* @return the new features position
* @param image image to analyse
*/
CvPoint2D32f* faceTracking( IplImage* image );

/**
* Convert the feature detection in movements using a simple averaging method
* the faceTracking function must be called before this one
* @param[in] currentFeaturesPointsIn  the current feature position
* @param[out] x the relative movement on the x axis
* @param[out] y the relative movement on the y axis
* @return 0 if the features movements are feasible, -1 if there is the need of reinitialize them
*/
void interpretMovementsAverage(CvPoint2D32f* currentFeaturesPointsIn, float* x, float* y);

/**
```

```
* Check if the tracker is still reliable
* @param[in] featuresPointsIn  the current feature position
* @param[in] initialFace the relative movement on the x axis
* @param[in] tempImage the relative movement on the y axis
* @param[in] debugImage
* @return 1 if an error has been tracked, 0 othewise
*/
int checkTrackingErrors(CvPoint2D32f* featuresPointsIn, CvRect* initialFace,
                                        IplImage* tempImage, IplImage* debugImage);
/**
* Deallocate all the memory used by the tracker
*/
void stopTracker( );
/**
* Set the uotput image used for the debug process
* @param image image used
*/
void setFaceDebugImg(IplImage* image);
```

4.8 Tracking Layer

It is one of the most IMP modules in the project. It contains the functions externally exposed by the library to the client layer. All the header files are included in this module.

It does following things:-

- Start the tracker and initialize the camera return CAMERA_NOT_FOUND if no cameras are found.
- Stop the tracker and dispose all the memory used.
- Retrieve the next frame produced by the camera return FRAME_RETRIEVED if the frame has been retrieved correctly.
- Try to locate human faces in front of the camera return the number of faces identified.
- Initialize the face area and the eyes location.
- Track the face movements and convert them into relative coordinates.
- Track an action given by the blinking motion of the eye lids.

In order to do all this functions it connects with respective **JAVA** programs.

'**TrackingLayer,h**' is header file for TrackingLayer.cpp it contains call for various OpenCV header file like highgui.h, CV.h, Tracker.h, configuration.h.

Trackinglayer.cpp

```
/*
* It contains the functions externally exposed by the library
* to the client layer.

*/
#include "TrackingLayer.h"
#include "configurator.h"
#include "camera.h"
#include "faceFeatInit.h"
#include "faceTracking.h"
#include "eyesTracking.h"

IplImage *CURR_FRAME;
/**< It stores the frame retrieved from the camera */
```

```
IplImage *TEMP_IMAGE;
/**< It stores a copy of the frame retrieved from the camera.
* It is used as the image passed to the various functions */
IplImage *NEW_IMAGE;
/**< It stores a copy of the frame retrieved from the camera at the time X.
* It is used together with OLD_IMAGE to detect a blink */
IplImage *OLD_IMAGE;
/**< It stores a copy of the frame retrieved from the camera at the time X - n.
* It is used together with NEW_IMAGE to detect a blink */
IplImage *DEBUG_IMAGE;
/**< It stores the image used to show the inner working of the algorithms */
CvPoint2D32f *FEATURES_IDENTIFIED;
/**< It stores the face features retrieved */
CvRect LEFT_EYE = cvRect(0,0,0,0);
/**< It stores the current location of the left eye */
CvRect RIGHT_EYE = cvRect(0,0,0,0);
/**< It stores the current location of the right eye */
CvRect *INITIAL_FACE = 0;
/**< It stores the initial face dimension */
int CURRENT_MOV_X = 0;
/**< It stores the current face movements on the X axis */
int CURRENT_MOV_Y = 0;
/**< It stores the current face movements on the Y axis */

/**
* Start the tracker and initialize the camera.
* @return CAMERA_NOT_FOUND if no cameras are found
*/
JNIEXPORT jint JNICALL Java_clientlayer_Tracker_startTracker
 (JNIEnv*, jobject) {
        //printf("Java_clientlayer_Tracker_startTracker\n");

int successfullInit = 0;
successfullInit = initCamera();
if(successfullInit) { //this test will fail if no camera is connected
//init global variabes
CURR_FRAME = getImage();
TEMP_IMAGE = cvCreateImage( cvSize(CURR_FRAME->width,CURR_FRAME->height),
CURR_FRAME->depth, CURR_FRAME->nChannels );
NEW_IMAGE = cvCreateImage( cvSize(CURR_FRAME->width,CURR_FRAME->height),
CURR_FRAME->depth, CURR_FRAME->nChannels );
OLD_IMAGE = cvCreateImage( cvSize(CURR_FRAME->width,CURR_FRAME->height),
CURR_FRAME->depth, CURR_FRAME->nChannels );

                FEATURES_IDENTIFIED = 0;
                //init other modules
                initFaceAreaIdentifier();

        //create the output window and image for the debug
if(DEBUG) {
cvNamedWindow( WINDOWS_NAME, CV_WINDOW_AUTOSIZE );
DEBUG_IMAGE = cvCreateImage( cvSize(CURR_FRAME->width,CURR_FRAME->height),
CURR_FRAME->depth, CURR_FRAME->nChannels );
                }
        }

        return successfullInit;
}
/**
 * Stop the tracker and dispose all the memory used.
 */
JNIEXPORT void JNICALL Java_clientlayer_Tracker_stopTrackerLayer
  (JNIEnv *, jobject){
        //printf("Java_clientlayer_Tracker_stopTrackerLayer\n");

        //stop other modules
        stopTracker();
        stopFaceAreaIdentifier();
        closeCamera();

        //deallocate global variables
        cvReleaseImage( &TEMP_IMAGE );
        cvReleaseImage( &OLD_IMAGE );
        cvReleaseImage( &NEW_IMAGE );
        free(FEATURES_IDENTIFIED);
        FEATURES_IDENTIFIED = 0;

        //deallocate debug resources
        if(DEBUG){
                cvDestroyWindow( WINDOWS_NAME );
        }}
/**
* Retrieve the next frame produced by the camera
* @return FRAME_RETRIEVED if the frame has been retrieved correctly
*/
JNIEXPORT jint JNICALL Java_clientlayer_Tracker_captureNextFrame
  (JNIEnv *, jobject){
//printf("Java_clientlayer_Tracker_captureNextFrame\n");

int error = clientlayer_Tracker_FRAME_RETRIEVED - 1; //set the frame as not retrieved
CURR_FRAME = getImage();

if(CURR_FRAME){
//cvFlip( CURR_FRAME,  TEMP_IMAGE, 0);
//flip around y axis and copy the frame to the temp image
cvCopy( CURR_FRAME,  TEMP_IMAGE);
//flip around y axis and copy the frame to the temp image
error = clientlayer_Tracker_FRAME_RETRIEVED;

if(DEBUG){
                //show the old image created
                cvShowImage(WINDOWS_NAME, DEBUG_IMAGE );
                        //create a copy of the original image for the debug
                        cvCopy(TEMP_IMAGE, DEBUG_IMAGE);
                        cvWaitKey(1);
                }
        }
        return error;
}
/**
* Try to locate human faces in front of the camera
* @return the number of faces identified
*/
JNIEXPORT jint JNICALL Java_clientlayer_Tracker_identifyFaces
  (JNIEnv *, jobject) {
        //printf("Java_clientlayer_Tracker_identifyFaces\n");

        CvSeq* faces = 0;
        int numFaces = 0;
        numFaces = identifyFaces( TEMP_IMAGE, &faces );
        int idxGoodFace = -1;
        int maxSize = 0;
        for( int i=0;i<numFaces;++i ){
                CvRect* tmpFaceRect = (CvRect*)cvGetSeqElem( faces, i );
                if( maxSize < tmpFaceRect->height*tmpFaceRect->width ){
                        maxSize = tmpFaceRect->height*tmpFaceRect->width;
                        idxGoodFace = i;
                }

        }

        if( idxGoodFace >= 0 ){
                INITIAL_FACE = (CvRect*)cvGetSeqElem( faces, idxGoodFace );
                numFaces = 1;
        }
/*
        if(DEBUG){
                //TODO: remove
        cvRectangle( TEMP_IMAGE, cvPoint( INITIAL_FACE->x, INITIAL_FACE->y),
        cvPoint( INITIAL_FACE->x + INITIAL_FACE->width , INITIAL_FACE-
>y+INITIAL_FACE->height),
                                        cvScalar(255,0,0), 2 );

                static int aaa = 0;
                char strFile[255];
                sprintf( strFile, "face%d.jpg",  aaa );
                cvSaveImage( strFile, TEMP_IMAGE );
                aaa++;
        }
*/

        return numFaces;
}
/**
* Initialise the face area and the eyes location.
* This procedure is required by faceTracking and eyesTracking.
*
* @return 0 if it is successfull
*/
```

```
JNIEXPORT jint JNICALL Java_clientlayer_Tracker_initFaceFeatures
  (JNIEnv *, jobject){
        int initNotSuccefull = 0;

        //printf("Java_clientlayer_Tracker_initFaceFeatures\n");

        //allocate the memory required by the features point
//      free((void**)&FEATURES_IDENTIFIED);
        if(FEATURES_IDENTIFIED == 0){
                FEATURES_IDENTIFIED = (CvPoint2D32f*)
malloc(sizeof(CvPoint2D32f)*MAX_FEAT);
        }

        bool initialised = faceFeatureInit( TEMP_IMAGE, INITIAL_FACE,
FEATURES_IDENTIFIED, &LEFT_EYE, &RIGHT_EYE);
        //printf("initialised\n");
        if(initialised){
                //printf("b1\n");
                trackerInit( TEMP_IMAGE, FEATURES_IDENTIFIED );
                //printf("b2\n");
//              eyesTrackerInit(&LEFT_EYE, &RIGHT_EYE, TEMP_IMAGE);

                //printf("initialised\n");
        }else{
                initNotSuccefull = 1;
                //printf("not initialised\n");
        }

        return initNotSuccefull;
}
/**
* Track the face movements and convert them into relative coordinates
* @param the coordinates found, or FACE_FEAT_LOST.
*/
JNIEXPORT void JNICALL Java_clientlayer_Tracker_faceTracking (JNIEnv *env, jobject obj,
jfloatArray coordinates){
        static jfloat coordFloat[2] = {0,0};

        //printf("Java_clientlayer_Tracker_faceTracking\n");
        bool errorDetected = false;
        //set the debug resources
        if(DEBUG){
                setFaceDebugImg( DEBUG_IMAGE );
        }

        //perform the feature tracking
        CvPoint2D32f* newFeatures = faceTracking( TEMP_IMAGE );

//      /* TODO: put back
//check errors due to wrong face configuration
if(checkTrackingErrors
(newFeatures, INITIAL_FACE, TEMP_IMAGE, DEBUG_IMAGE) == 1){
        errorDetected = true;
        printf("checkTrackingErrors: errorDetected\n");
        }
//      */


        if(!errorDetected){
                //calculate the movements
                interpretMovementsAverage( newFeatures, &coordFloat[0], &coordFloat[1] );
                env->SetFloatArrayRegion( coordinates, 0, 2, coordFloat);

                //update current coordinates (in order to help the eye tracking)
                CURRENT_MOV_X = (int)coordFloat[0];
                CURRENT_MOV_Y = (int)coordFloat[1];

                //printf("calculate mov\n");
        }

        if(errorDetected){
                //set the error flag
                coordFloat[0] = clientlayer_Tracker_FACE_FEAT_LOST;
                coordFloat[1] = clientlayer_Tracker_FACE_FEAT_LOST;
                env->SetFloatArrayRegion(coordinates, 0, 2, coordFloat);
        }


}
```

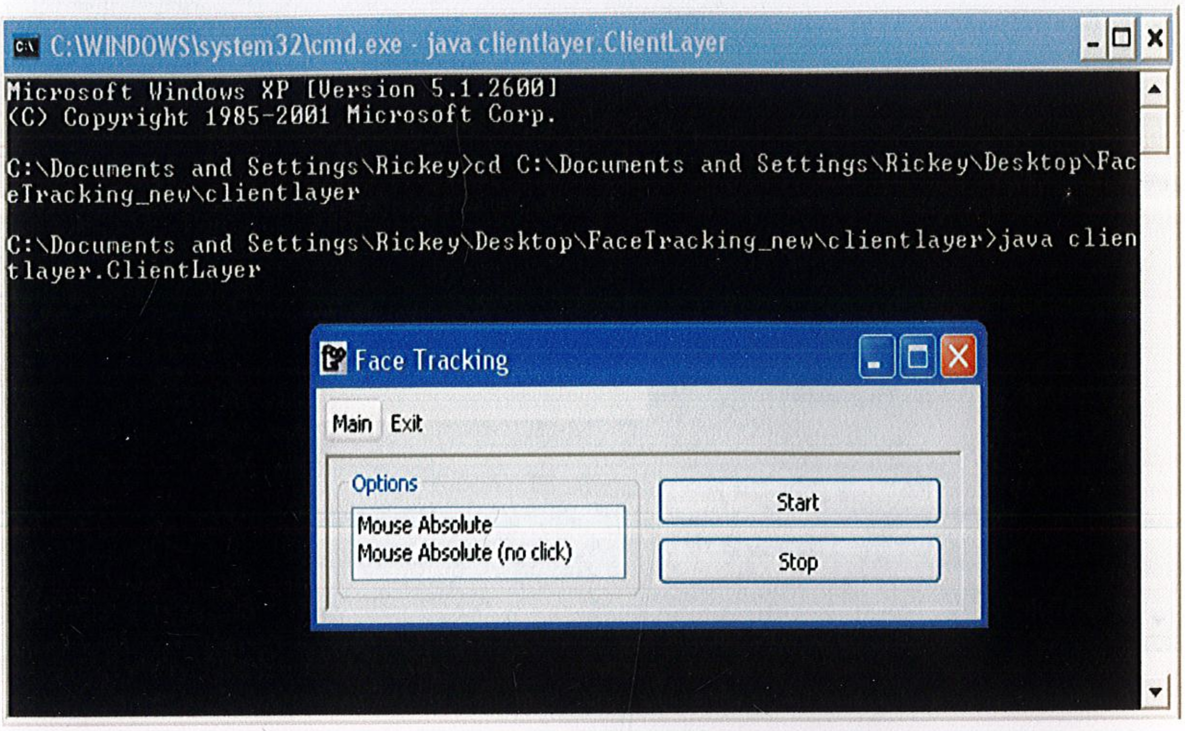
```
/**
* Track an action given by the blinking motion of the eye lids
* @return ACTION_DETECTED, or ACTION_NOT_DETECTED.
*/
JNIEXPORT jint JNICALL Java_clientlayer_Tracker_eyesTracking(JNIEnv *, jobject){
        //printf("Java_clientlayer_Tracker_eyesTracking\n");

        //counter used to store two different frames
        static int framesSpent = 0;
        //flag that detects if it is the first call
        static bool firstRunTracking = true;
        //storage for the old and new movements
        static int oldFaceMovementsX = 0;
        static int oldFaceMovementsY = 0;
        static int newFaceMovementsX = 0;
        static int newFaceMovementsY = 0;
        //return flag
        jint actionDetected = clientlayer_Tracker_ACTION_NOT_DETECTED;

        //set the debug resources
        if(DEBUG){
                setEyesDebugImg(DEBUG_IMAGE);
        }

        //store the image in the buffer to be analysed by the eye tracker
        ++framesSpent;
        if(firstRunTracking){
                cvCopy(TEMP_IMAGE, OLD_IMAGE);
                cvCopy(TEMP_IMAGE, NEW_IMAGE);
                newFaceMovementsX = oldFaceMovementsX = CURRENT_MOV_X;
                newFaceMovementsY = oldFaceMovementsY = CURRENT_MOV_Y;
                firstRunTracking = false;
        }else if(framesSpent >= EYES_FRAMES_GAP){
                //CV_SWAP( oldImage, newImage, tempImage );
                cvCopy(NEW_IMAGE, OLD_IMAGE);
                cvCopy(TEMP_IMAGE, NEW_IMAGE);
                //store the new (and old) face movements
                oldFaceMovementsX = newFaceMovementsX;
                oldFaceMovementsY = newFaceMovementsY;
                newFaceMovementsX = CURRENT_MOV_X;
                newFaceMovementsY = CURRENT_MOV_Y;

                //follow the eyes
                int failed = updateEyesLocation(newFaceMovementsX, newFaceMovementsY,
                                                 oldFaceMovementsX, oldFaceMovementsY,
                                                 TEMP_IMAGE);

                //track blinking
                if(!failed && detectBlink(OLD_IMAGE, NEW_IMAGE, TEMP_IMAGE)){
                        actionDetected = clientlayer_Tracker_ACTION_DETECTED;
                }

                framesSpent = 0;
        }

        return actionDetected;
}
/**
* Set the configuration of the tracker.
* The tracker must be stopped before call this method.
* @param configuration a java object containing the new configuration
*/
JNIEXPORT void JNICALL Java_clientlayer_Tracker_setConfiguration
  (JNIEnv *env, jobject obj, jobject configuration){
        //printf("Java_clientlayer_Tracker_setConfiguration\n");

        setConfiguration(env, configuration);
}
/**
* Get the actual configuration
* @return the actual configuration
*/
JNIEXPORT jobject JNICALL Java_clientlayer_Tracker_getConfiguration
(JNIEnv *env, jobject obj){
        //printf("Java_clientlayer_Tracker_getConfiguration\n");

        return getConfiguration(env);}
```

Trackinglayer.h

```
// Header of \ref TrackingLayer.cpp

#include <stdio.h>
#include <ctype.h>
#include "highgui.h"
#include "cv.h"
#include "Tracker.h"
#include "configuration.h"
```

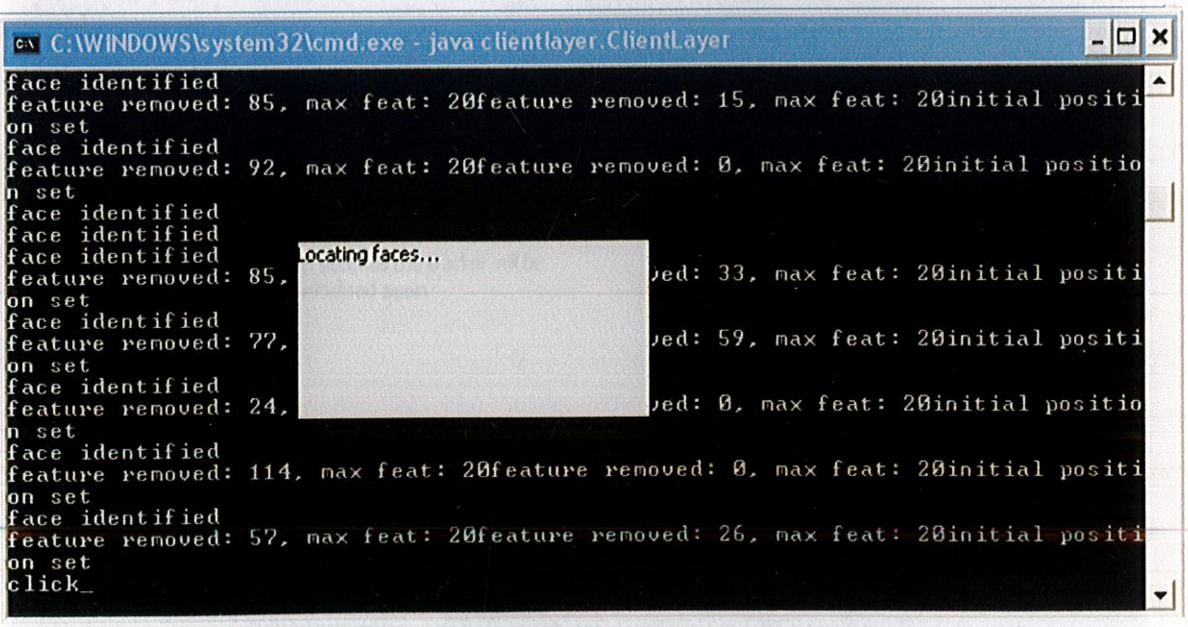
4.9 System SnapShots

1.Initial startup Window


C:\WINDOWS\system32\cmd.exe - java clientlayer.ClientLayer
Face Tracking
Main
Exit
Options
Mouse Absolute
Mouse Absolute (no click)
Start
Stop

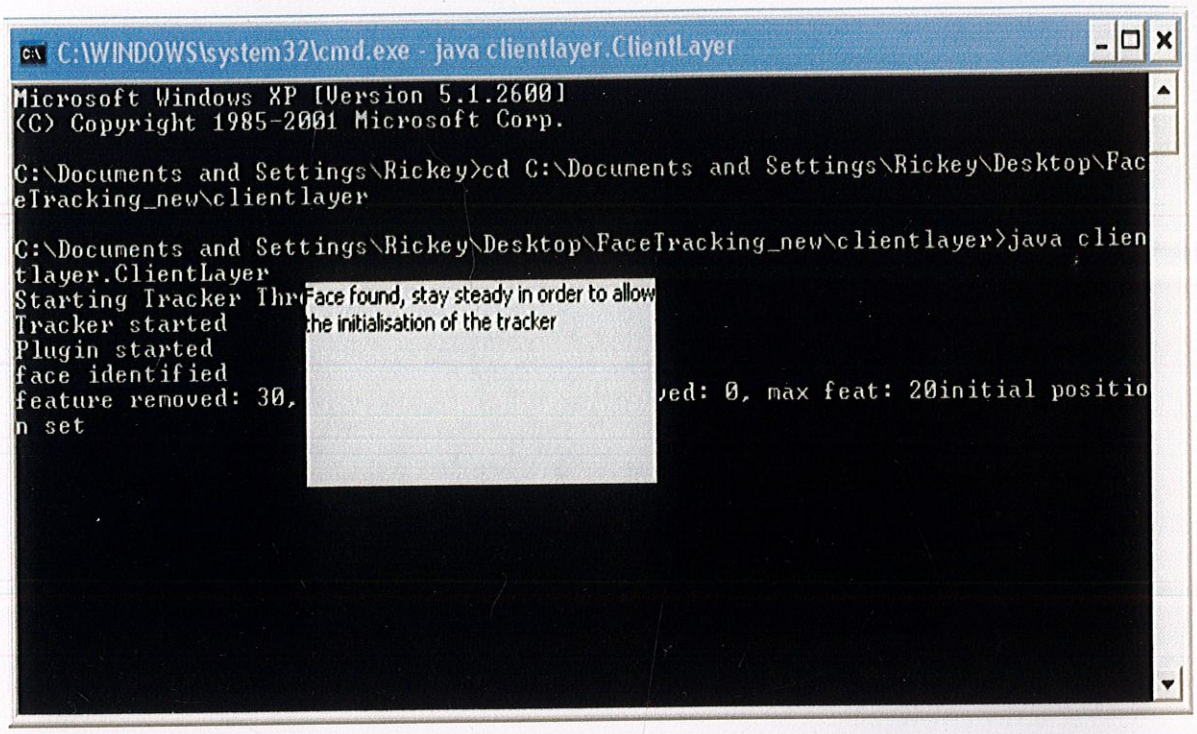
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Rickey>cd C:\Documents and Settings\Rickey\Desktop\Fac
eTracking_new\clientlayer

C:\Documents and Settings\Rickey\Desktop\FaceTracking_new\clientlayer>java clien
tlayer.ClientLayer
```

2.Camera initiated and locating Faces starts



```
face identified
feature removed: 85, max feat: 20feature removed: 15, max feat: 20initial positi
on set
face identified
feature removed: 92, max feat: 20feature removed: 0, max feat: 20initial positio
n set
face identified
face identified
face identified
feature removed: 85,                         ved: 33, max feat: 20initial positi
on set
face identified
feature removed: 77,                         ved: 59, max feat: 20initial positi
on set
face identified
feature removed: 24,                         ved: 0, max feat: 20initial positio
n set
face identified
feature removed: 114, max feat: 20feature removed: 0, max feat: 20initial positi
on set
face identified
feature removed: 57, max feat: 20feature removed: 26, max feat: 20initial positi
on set
click_
```

3.Face Located: Tracker being Initialized

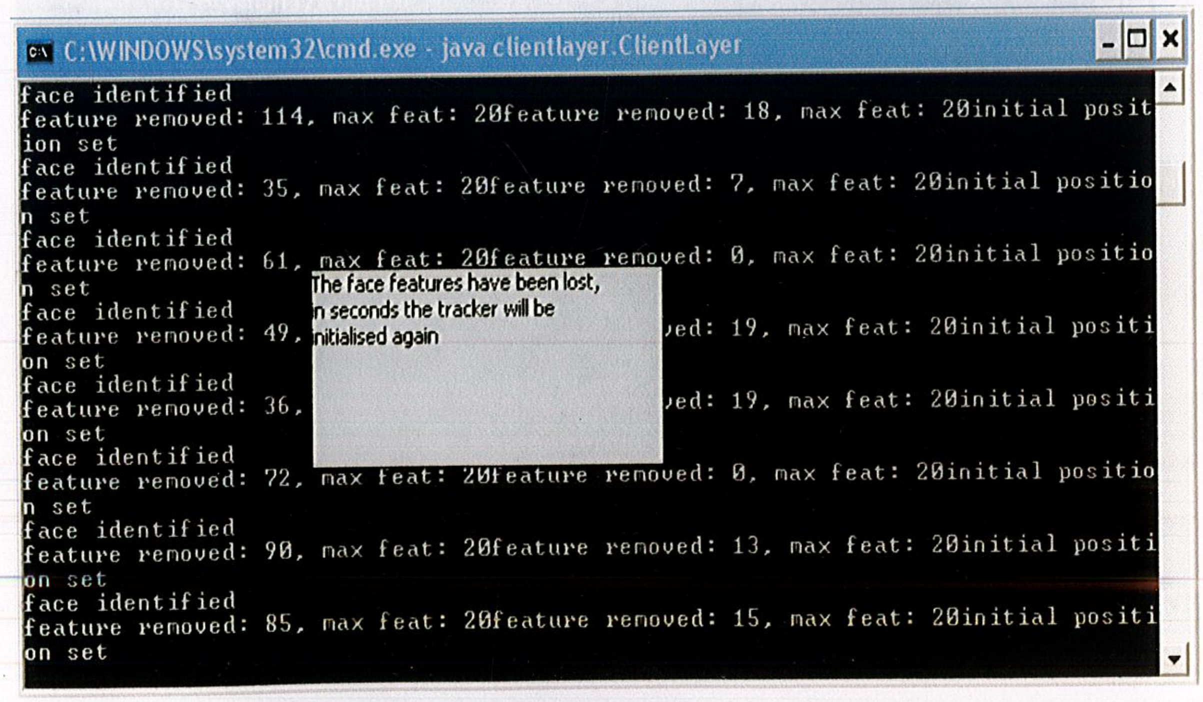


C:\WINDOWS\system32\cmd.exe - java clientlayer.ClientLayer

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Rickey>cd C:\Documents and Settings\Rickey\Desktop\Fac
eTracking_new\clientlayer

C:\Documents and Settings\Rickey\Desktop\FaceTracking_new\clientlayer>java clien
tlayer.ClientLayer
Starting Tracker Thr
Tracker started
Plugin started
face identified
feature removed: 30,                                 ved: 0, max feat: 20initial positio
n set
```

Face found, stay steady in order to allow the initialisation of the tracker

4.Relocation of face features in case its lost

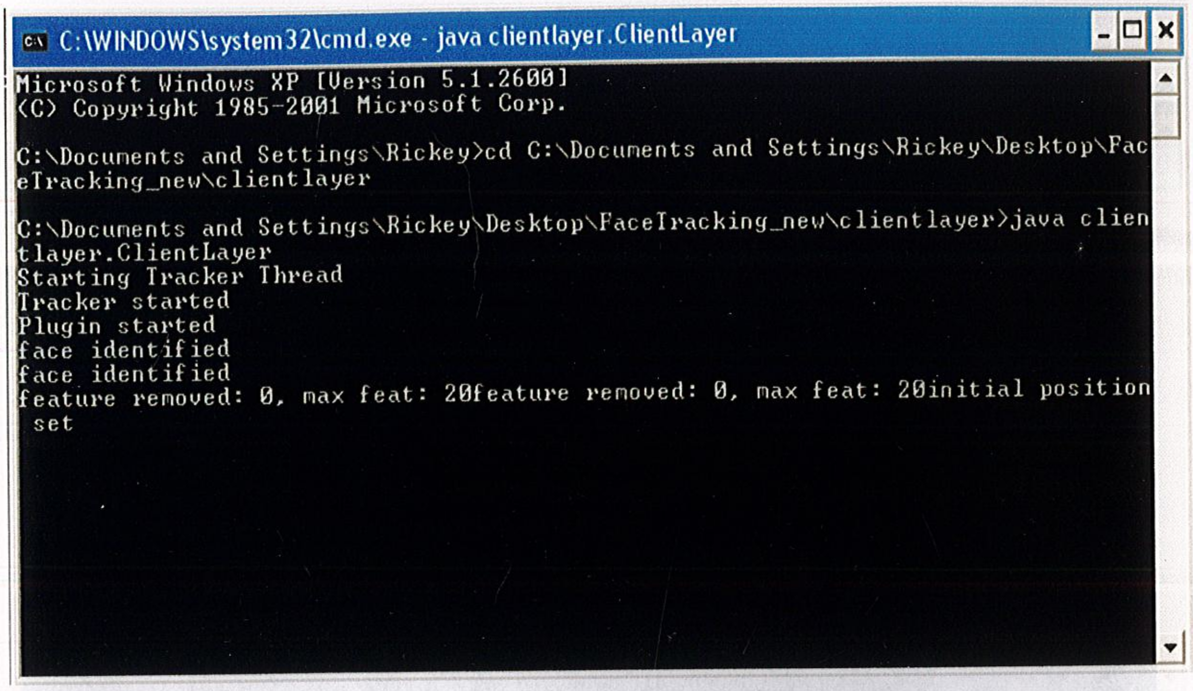


C:\WINDOWS\system32\cmd.exe - java clientlayer.ClientLayer

```
face identified
feature removed: 114, max feat: 20feature removed: 18, max feat: 20initial posit
ion set
face identified
feature removed: 35, max feat: 20feature removed: 7, max feat: 20initial positio
n set
face identified
feature removed: 61, max feat: 20feature removed: 0, max feat: 20initial positio
n set
face identified
feature removed: 49,                                ved: 19, max feat: 20initial positi
on set
face identified
feature removed: 36,                                ved: 19, max feat: 20initial positi
on set
face identified
feature removed: 72, max feat: 20feature removed: 0, max feat: 20initial positio
n set
face identified
feature removed: 90, max feat: 20feature removed: 13, max feat: 20initial positi
on set
face identified
feature removed: 85, max feat: 20feature removed: 15, max feat: 20initial positi
on set
```

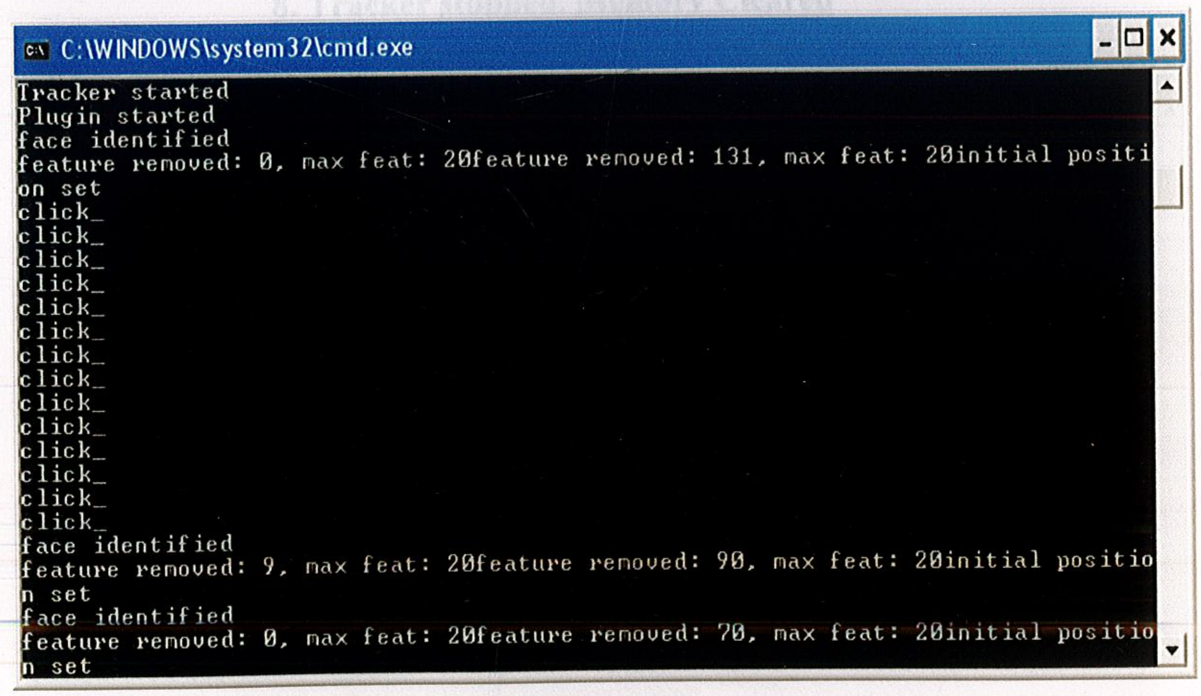
The face features have been lost, in seconds the tracker will be initialised again

5.Face features retracked


C:\WINDOWS\system32\cmd.exe - java clientlayer.ClientLayer

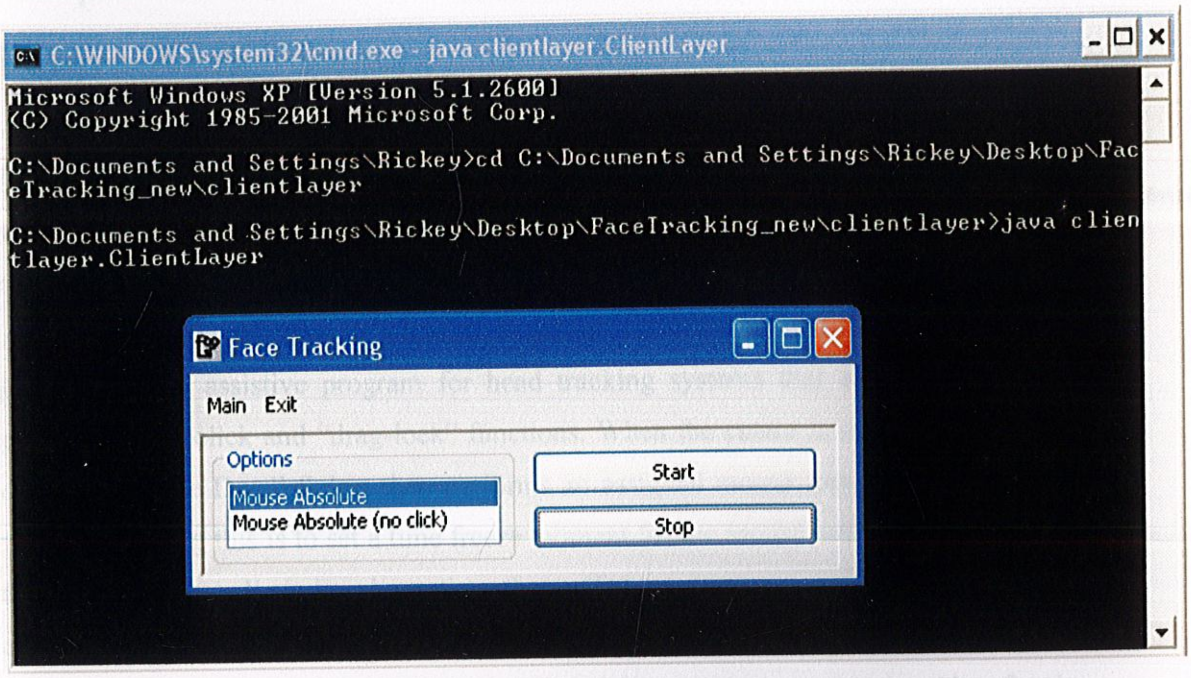
```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Rickey>cd C:\Documents and Settings\Rickey\Desktop\Fac
eTracking_new\clientlayer

C:\Documents and Settings\Rickey\Desktop\FaceTracking_new\clientlayer>java clien
tlayer.ClientLayer
Starting Tracker Thread
Tracker started
Plugin started
face identified
face identified
feature removed: 0, max feat: 20feature removed: 0, max feat: 20initial position
 set
```

6.Click Initiated


C:\WINDOWS\system32\cmd.exe

```
Tracker started
Plugin started
face identified
feature removed: 0, max feat: 20feature removed: 131, max feat: 20initial positi
on set
click_
click_
click_
click_
click_
click_
click_
click_
click_
click_
click_
click_
click_
click_
face identified
feature removed: 9, max feat: 20feature removed: 90, max feat: 20initial positio
n set
face identified
feature removed: 0, max feat: 20feature removed: 70, max feat: 20initial positio
n set
```

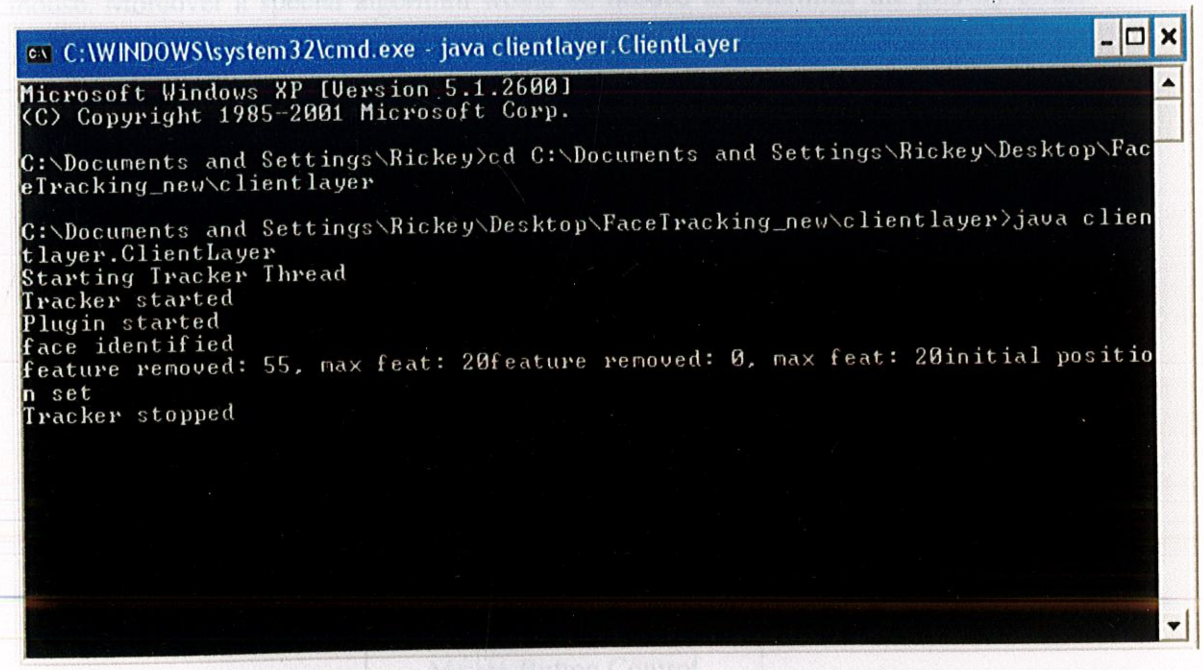
7.Tracker Stopped



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Rickey>cd C:\Documents and Settings\Rickey\Desktop\Fac
eTracking_new\clientlayer

C:\Documents and Settings\Rickey\Desktop\FaceTracking_new\clientlayer>java clien
tlayer.ClientLayer
```

C:\WINDOWS\system32\cmd.exe - java clientlayer.ClientLayer
Face Tracking
Main
Exit
Options
Mouse Absolute
Mouse Absolute (no click)
Start
Stop

8. Tracker stopped, Memory Cleared



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Rickey>cd C:\Documents and Settings\Rickey\Desktop\Fac
eTracking_new\clientlayer

C:\Documents and Settings\Rickey\Desktop\FaceTracking_new\clientlayer>java clien
tlayer.ClientLayer
Starting Tracker Thread
Tracker started
Plugin started
face identified
feature removed: 55, max feat: 20feature removed: 0, max feat: 20initial positio
n set
Tracker stopped
```

Chapter 5
Future Work

As a part of the future work we intend to make the clicking by this system more efficient. Mouse clicking is proposed to be performed on the basis of facial expression or some voice. Nikolaus Bee et al investigated the usability of an eye controlled writing interface that matches the nature of human eye gaze, which always moves and is not immediately able to trigger the selection of a button. A similar interface (termed Dwell Select in this proposal) is an assistive program for head tracking systems that enables the user to perform mouse click and "drag-lock" functions. When the cursor is stationary for a pre-defined time the Dwell Select then performs an assigned mouse function. One possible way of obtaining this is to set a time freeze moment for the mouse pointer, which implies that we set a time limit in whose time frame if there is no movement in the position coordinates of the pointer, open operation is performed for the icon in the vicinity of the pointer. But this feature would involve a lot of complexity since the algorithm for the time frame calculation would have to run iteratively after the change in the position of the mouse. Moreover a special algorithm would be needed to determine the priority of the icon to be opened in case the pointer is not on the correct location.

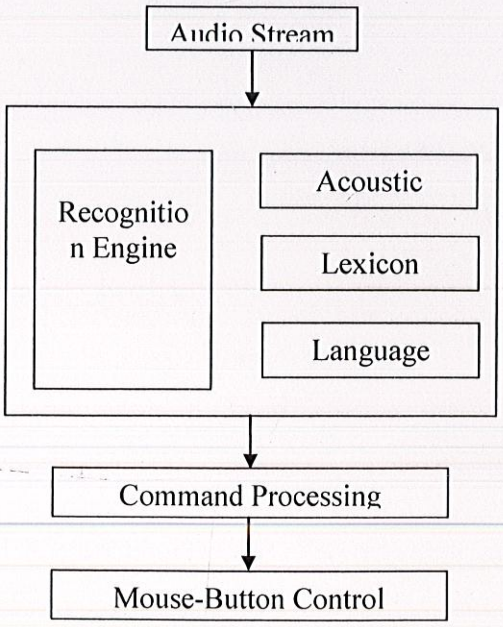


Figure 10: The proposed speech-recognition components of our head-tracking system.

Moreover since the system currently employs a single came, we intend to improve the tracking efficiency by introducing more cameras and entering the 3D domain, so that the features locked cannot be lost even if the face is out of frame from the 1st camera.

Bibliography

[1] Bruce D. Lucas, Takeo Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", In the Proceedings of Imaging Understanding Workshop, Washington, DC, pp. 121-130 (1981).

[2] J. Wolberg, "Data Analysis Using the Least-Squares Method: How to Extract the Most Information from Experiments", Springer-Verlag New York, Dec. 2005.

[3] Jurriaan D. Mulder, Jack Jansen, and Arjen van Rhijn , "An Affordable Optical Head Tracking System for Desktop ",VR/AR Systems Center for Mathematics and Computer Science, CWI Amsterdam, the Netherlands fmullie,jack,arjenvrg@cwi.nl

[4] Nikolaus Bee, Elisabeth Andre, "Writing with Your Eye: A Dwell Time Free Writing System Adapted to the Nature of Human Eye Gaze", *Perception in Multimodal Dialogue Systems*, pp.111-122, Lecture Notes in Computer Science, Springer Berlin 1981.

[5] Frank Loewenich, Frederic Maire Motion, "Tracking and Speech Recognition for Hands-Free Mouse-Pointer Manipulation", Queensland University of Technology, Australia.

[6]S. De Backer, P. Scheunders, "A Competitive Elliptical Clustering Algorithm", *Vision Lab,* Department of Physics, University of Antwerp, Groenenborgerlaan 171, 2020 Antwerpen, Belgium.