CHAPTER 1
INTRODUCTION

1.1 PROBLEM STATEMENT

There are a number of issues and problems faced by specially abled beings due to which they feel alienated.For example if they want to use a certain vending machine in a public place they might feel a bit inconvenient & dependent on others for simple day to day tasks. So for even simple tasks, if there is nobody around they might feel helpless and there is no choice left for them either to leave that task or wait for someone to help them out.

1.2 MOTIVATION OF THE PROJECT

From what we have recognized whenever a specially abled person is trying to use a vending machine interface he/she faces difficulty in using even the most simplest interfaces for normal people.When we talk about any automated system our first aim is to provide people an easy interface accessible by all which we see lacking in the normal vending machines or any commonly used software.

1.3 PROJECT GOALS AND SCOPE

We aim to provide an easy software interface for specially abled, to allow them to use any vending machine or any other software that has an user interaction with minimal or no help required to perform any task.

Our software provide users with an easy interface, using head movements as cursor movements and eye blinks for product selection and confirmation.The main motive of our software is to elevate the customer experience, to reduce the problems faced by specially abled section of the society, and to improve the efficiency of the operations. We aim to provide specially abled person a friendly software for a hassle free & fast experience and also to all kind of customers .

The latest market intelligence study on the Smart Vending Machines market applies the best of both primary and secondary research techniques to bring to light the growth rate of the Smart Vending Machines market for the forecast period, 2018 - 2025.

A comprehensive study covers hard to find facts about the market landscape as well as its growth prospects in the years to come. Most importantly, the research report includes vital statistics about the major vendors occupying a strong foothold in this industry. Besides this, in order to calculate the market share, the study takes a closer look at the selling price of the product across different regions.

1.4 MACHINE LEARNING AND COMPUTER VISION

Machine learning is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.The name machine learning was coined in 1959 by Arthur Samuel.

Machine learning explores the study and construction of algorithms that can learn from and make predictions on data– such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible; example applications include email filtering, detection of network intruders, and computer vision.

Machine learning is closely related to (and often overlaps with) computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes conflated with data mining,where the latter subfield focuses more on exploratory data analysis and is known as unsupervised learning.

Within the field of data analytics, machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytical models allow researchers, data scientists, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g. in the forms of decisions. Understanding in this context means the transformation of visual images (the input of the retina) into descriptions of the world that can interface with other thought processes and elicit appropriate action. This image understanding can be seen as the disentangling of symbolic information from image data using models constructed with the aid of geometry, physics, statistics, and learning theory.

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

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, and image restoration.

1.5 COMPUTER VISION AND FACIAL LANDMARKS

A landmark is a recognizable natural or man-made feature used for navigation feature that stands out from its near environment and is often visible from long distances. Facial

landmarks is defined as the detection and localization of certain keypoints points on the face which have an impact on subsequent task focused on the face, like animation, face recognition, gaze detection, face tracking, expression recognition, gesture understanding etc. Facial landmark are a prominent feature that can play a discriminative role or can serve as anchor points on a face graph.

Facial landmarks are the nose tip, eyes corners, chin, mouth corners, nostril corners, eyebrow arcs, ear lobes etc. For ease of analysis most landmark detection algorithm prefers an entire facial semantic region, such as the whole region of a mouth, the region of the nose, eyes, eyebrows, cheek or chin. The facial landmarks are classified in two groups, primary and secondary , or fiducial and ancillary. This distinction is based on reliability of image features detection techniques. For example, the corners of the mouth, of the eyes, the nose tips and eyebrows can detected relatively easily by using low level image features, e.g. SIFT, HOG. The directly detected landmarks are referred as fiducial. The fiducial group of landmarks and they play a more prominent role in facial identity and face tracking. The search for secondary landmarks is guided by primary landmarks. The secondary landmarks are chin, cheek contours, eyebrow and lips midpoints, non extremity points, nostrils. It takes more prominent role in facial expression.

Facial landmarks can be used to align facial images to a mean face shape, so that after alignment the location of facial landmarks in all images is approximately the same. Intuitively it makes sense that facial recognition algorithms trained with aligned images would perform much better, and this intuition has been confirmed by many research papers.Once you know a few landmark points, you can also estimate the pose of the head. In other words you can figure out how the head is oriented in space, or where the person is looking. E.g. CLM-Framework described in this post also returns the head pose.

1.51 Implementation Techniques

- **Dlib (C++ / Python)**

  Dlib is a collection of miscellaneous algorithms in Machine Learning, Computer Vision, Image Processing, and Linear Algebra. Most of the library is just header files that you can include in your C++ application.

- **CLM-Framework (C++)**

  CLM-framework, also known as the Cambridge Face Tracker, is a C++ library for facial keypoint detection and head pose estimation.Compiling this library on OSX was bit of a challenge but it was not too bad. The library depends on OpenCV 3 and requires X11.There are two important ways in which Dlib beats CLM-Framework. First, DLib is much faster than CLM-Framework. Second, Dlib's license allows you to use it in commercial applications.

- **Face++ (FacePlusPlus) : Web API**

  One of the best implementations of facial landmark detection is by FacePlusPlus. They won the 300 Faces In-the-Wild Landmark Detection Challenge, 2013. They provide an easy to use API. The problem is that you need to upload an image to their servers

CHAPTER 2

REQUIREMENT ANALYSIS

Requirement analysis in systems engineering and software engineering, encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possible conflicting requirements of the various stakeholders, such as beneficiaries or users. It is an early stage in the more general activity of requirements engineering which encompasses all activities concerned with eliciting, analyzing, documenting, validating and managing software or system requirements.

2.1 FEASIBILITY ANALYSIS

Feasibility study is made to see if the project on completion will serve the purpose of the organization for the amount of work, effort and the time that is spent on it. Feasibility study lets the developer foresee the future of the project and the usefulness. A feasibility study of a system proposal is according to its workability, which is the impact on the organization, ability to meet their user needs and effective use of resources. Thus when a new application is proposed it normally goes through a feasibility study before it is approved for development.

The document provide the feasibility of the project that is being designed and lists various areas that were considered very carefully during the feasibility study of this project such as Technical, Economic and Operational feasibilities. The following are its features:

2.11 TECHNICAL FEASIBILITY

The system must be evaluated from the technical point of view first. The assessment of this feasibility must be based on an outline design of the system requirement in the terms of input, output, programs and procedures. Having identified an outline system, the investigation must go on to suggest the type of equipment, required method developing the system, of running the system once it has been designed. Technical issues raised during the investigation are :

Does the existing technology sufficient for the suggested one?

Can the system expand if developed?

The project should be developed such that the necessary functions and performance are achieved within the constraints. The project is developed within latest technology.

Though the technology may become obsolete after some period of time, due to the fact that newer version of same software supports older versions, the system may still be used. So there are minimal constraints involved with this project. The system has been developed using python the project is technically feasible for development.

2.12 ECONOMIC FEASIBILITY

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require.

The following are some of the important financial questions asked during preliminary investigation:

- The cost to conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the form of reduced costs or fewer costly errors.

  Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already

available, it give an indication of the system is economically possible for development.

2.13 OPERATIONAL FEASIBILITY

Operational feasibility is a measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. Our proposed system overcomes all the problems related to present complex system and satisfies all the scope as defined.

2.2 FUNCTIONAL REQUIREMENT ANALYSIS

The Functional Requirements Definition documents and tracks the necessary information required to effectively define the process and this document is created during the Planning Phase of the project. From the functional perspective the project is Measurable, Realistic and Complete. Our project doesn‟t require any special training to use it and can be used by any common man. Considering the problems with the current system, understanding the user's needs and expectations the requirements for the proposed system is collected and found to be complete. Also the system does not require any additional features that may cause delay in the release of the project. This makes our project functionally measurable, realistic and complete.

2.3 NON-FUNCTIONAL REQUIREMENTS

2.31 QUALITY OF SERVICE

Quality Of Service (QOS) is a major issue in desktop applications. In the proposed system, there is constant transfer of data between the peripheral (here camera) and the system. The simplicity of the algorithm enhances the response time, thus bettering the QOS of the system.

2.32 AVAILABILITY

The user does not need to register to the system before using it. It can be used by any Human due to the feature classification method used in the system. It is easy to use and available to all.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATION

3.1 Python and associated packages

Python was the language of choice for this project. This was an easy decision for the multiple reasons.

- 1. Python as a language has an enormous community behind it. Any problems that might be encountered can be easily solved with a trip to Stack Overflow. Python is among the most popular languages on the site which makes it very likely there will be a direct answer to any query .

- 2. Python has an abundance of powerful tools ready for scientific computing. Packages such as Numpy, Pandas, and SciPy are freely available, performant, and well documented. Packages such as these can dramatically reduce, and simplify the code needed to write a given program. This makes iteration quick.

- 3. Python as a language is forgiving and allows for programs that look like pseudo code. This is useful when pseudo code given in academic papers needs to be implemented and tested. Using Python, this step is usually reasonably trivial.

However, Python is not without its flaws. The language is dynamically typed and packages are notorious for Duck Typing. This can be frustrating when a package method returns something that, for example, looks like an array rather than being an actual array. Coupled with the fact that standard Python documentation does not explicitly state the return type of a method, this can lead to a lot of trial and error testing that would not

otherwise happen in a strongly typed language. This is an issue that makes learning to use a new Python package or library more difficult than it otherwise could be

3.2 IMPORTED MODULES

3.21 OpenCV

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image

database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV's deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

3.22 Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code

- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure (which, despite the name, is a dynamic array), these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays.

3.23 Dlib

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

To follow or participate in the development of dlib subscribe to dlib on github. Also be sure to read the how to contribute page if you intend to submit code to the project.

To quickly get started using dlib, follow these instructions to build dlib.

Major Features

- **Documentation**
    - Unlike a lot of open source projects, this one provides complete and precise documentation for every class and function. There are also debugging modes that check the documented preconditions for functions. When this is enabled it will catch the vast majority of bugs caused by calling functions incorrectly or using objects in an incorrect manner.
    - Lots of example programs are provided.
    - I consider the documentation to be the most important part of the library. So if you find anything that isn't documented, isn't clear, or has out of date documentation, tell me and I will fix it.
- **High Quality Portable Code**
    - Good unit test coverage. The ratio of unit test lines of code to library lines of code is about 1 to 4.
    - The library is tested regularly on MS Windows, Linux, and Mac OS X systems. However, it should work on any POSIX system and has been used on Solaris, HPUX, and the BSDs.
    - No other packages are required to use the library. Only APIs that are provided by an out of the box OS are needed.
- **Machine Learning Algorithms**
    - Deep Learning.
    - Conventional SMO based Support Vector Machines for classification and regression.
    - Reduced-rank methods for large-scale classification and regression.
    - Relevance vector machines for classification and regression.
    - General purpose multiclass classification tools.

    - Structural SVM tools for object detection in images as well as more powerful (but slower) deep learning tools for object detection.

- Structural SVM tools for labeling nodes in graphs.
- A large-scale SVM-Rank implementation.
- An online kernel RLS regression algorithm.
- An online SVM classification algorithm.

- **Numerical Algorithms**
  - A fast matrix object implemented using the expression templates technique and capable of using BLAS and LAPACK libraries when available.
  - Numerous linear algebra and mathematical operations are defined for the matrix object such as the singular value decomposition, transpose, trig functions, etc.
  - A big integer object.
  - A random number object.
- **Image Processing**
  - Routines for reading and writing common image formats.
  - Automatic color space conversion between various pixel types.
  - Implementations of the SURF, HOG, and FHOG feature extraction algorithms.
  - Tools for detecting objects in images including frontal face detection and object pose estimation.
  - High quality face recognition.

3.24 Imutils

This package includes a series of OpenCV + convenience functions that perform basics tasks such as translation, rotation, resizing, and skeletonization.

Installation

This package assumes that you already have NumPy and OpenCV installed (along with matplotlib, if you intend on using the opencv2 matplotlib function).

To install the the imutils library, just issue the following command:

I just open sourced my personal imutils package: A series of OpenCV convenience functions.Python

"*$ pip install imutils"*

Translation

Translation is the shifting of an image in either the x or y direction. To translate an image in OpenCV you need to supply the (x, y)-shift, denoted as (tx, ty) to construct the translation matrix M:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

And from there, you would need to apply the cv2.warpAffine function.

Instead of manually constructing the translation matrix M and calling cv2.warpAffine , you can simply make a call to the translate function of imutils .

Example: Python

1. translate the image x=25 pixels to the right and y=75 pixels up
2. translated = imutils.translate(workspace, 25, -75)

Resizing

Resizing an image in OpenCV is accomplished by calling the cv2.resize function. However, special care needs to be taken to ensure that the aspect ratio is maintained. Thisresize function of imutils maintains the aspect ratio and provides the keyword arguments width and height so the image can be resized to the intended width/height while (1) maintaining aspect ratio and (2) ensuring the dimensions of the image do not have to be explicitly computed by the developer.

Another optional keyword argument, inter , can be used to specify interpolation method as well.

Example:

I just open sourced my personal imutils package: A series of OpenCV convenience functions.

"Python

1. *# loop over varying widths to resize the image to*
2. *for width in (400, 300, 200, 100):*
3. *# resize the image and display it*
4. *resized = imutils.resize(workspace, width=width)*
5. *cv2.imshow("Width=%dpx" % (width), resized) "*

3.25 PyAutoGui

The purpose of PyAutoGUI is to provide a cross-platform Python module for GUI automation for human beings. The API is designed to be as simple as possible with sensible defaults.

For example, here is the complete code to move the mouse to the middle of the screen on Windows, OS X, and Linux:

```
>>> import pyautogui
>>> screenWidth, screenHeight = pyautogui.size()
```

>>> pyautogui.moveTo(screenWidth / 2, screenHeight / 2)
And that is all.

PyAutoGUI can simulate moving the mouse, clicking the mouse, dragging with the mouse, pressing keys, pressing and holding keys, and pressing keyboard hotkey combinations.

Dependencies

On Windows, PyAutoGUI has no dependencies (other than Pillow and some other modules, which are installed by pip along with PyAutoGUI). It does not need the pywin32 module installed since it uses Python's own ctypes module.

On OS X, PyAutoGUI requires PyObjC installed for the AppKit and Quartz modules. The module names on PyPI to install are pyobjc-core and pyobjc (in that order).On Linux, PyAutoGUI requires python-xlib (for Python 2) or python3-Xlib (for Python 3) module installed.

CHAPTER 4

UML DIAGRAMS

4.1 Use Case Diagram

Use case diagrams are behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system. The User looks into the camera. The camera captures frames of the User and according to User head movements determines the motion of the pointer to the left, right, up and down. Similarly, User also judges whether to select mode 1 or mode 2 according to the mouth open and close motion

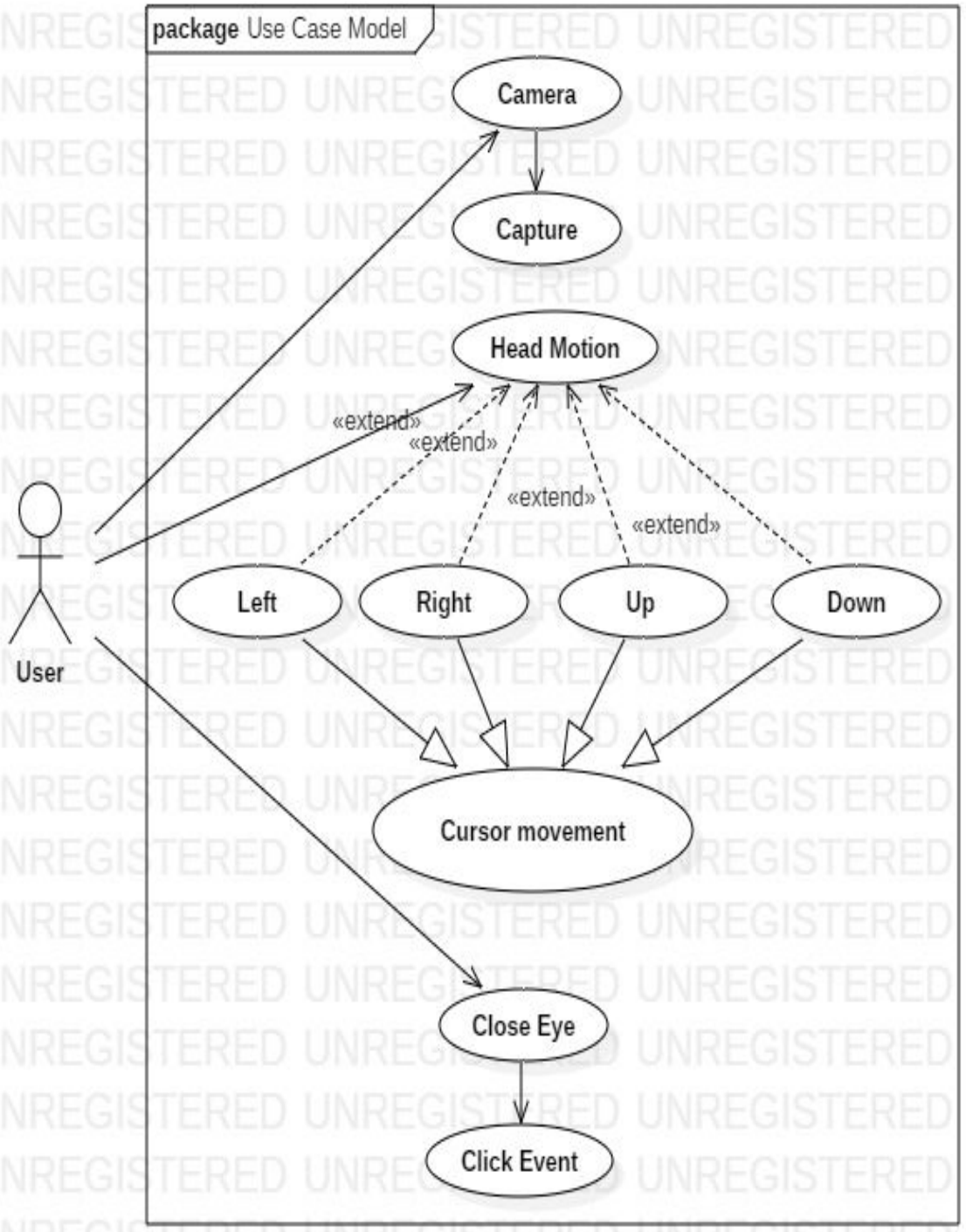

package Use Case Model
Camera
Capture
Head Motion
«extend»
«extend»
«extend»
«extend»
Left
Right
Up
Down
User
Cursor movement
Close Eye
Click Event

Fig. 4.11

4.2 Activity Diagram

Activity diagrams are graphical representations of work flows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by step workflows of components in a system. An activity diagram shows the overall flow of control.

Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:

- rounded rectangles represent activities
- diamonds represent decisions
- bars represent the start (split) or end (join) of concurrent activities
- a black circle represents the start (initial state) of the workflow
- An encircled black circle represents the end (final state)

Arrows run from the start towards the end and represent the order in which activities happen. The join and split symbols in activity diagrams only resolve this for simple cases the meaning of the model is not clear when they are arbitrarily combined with decisions.

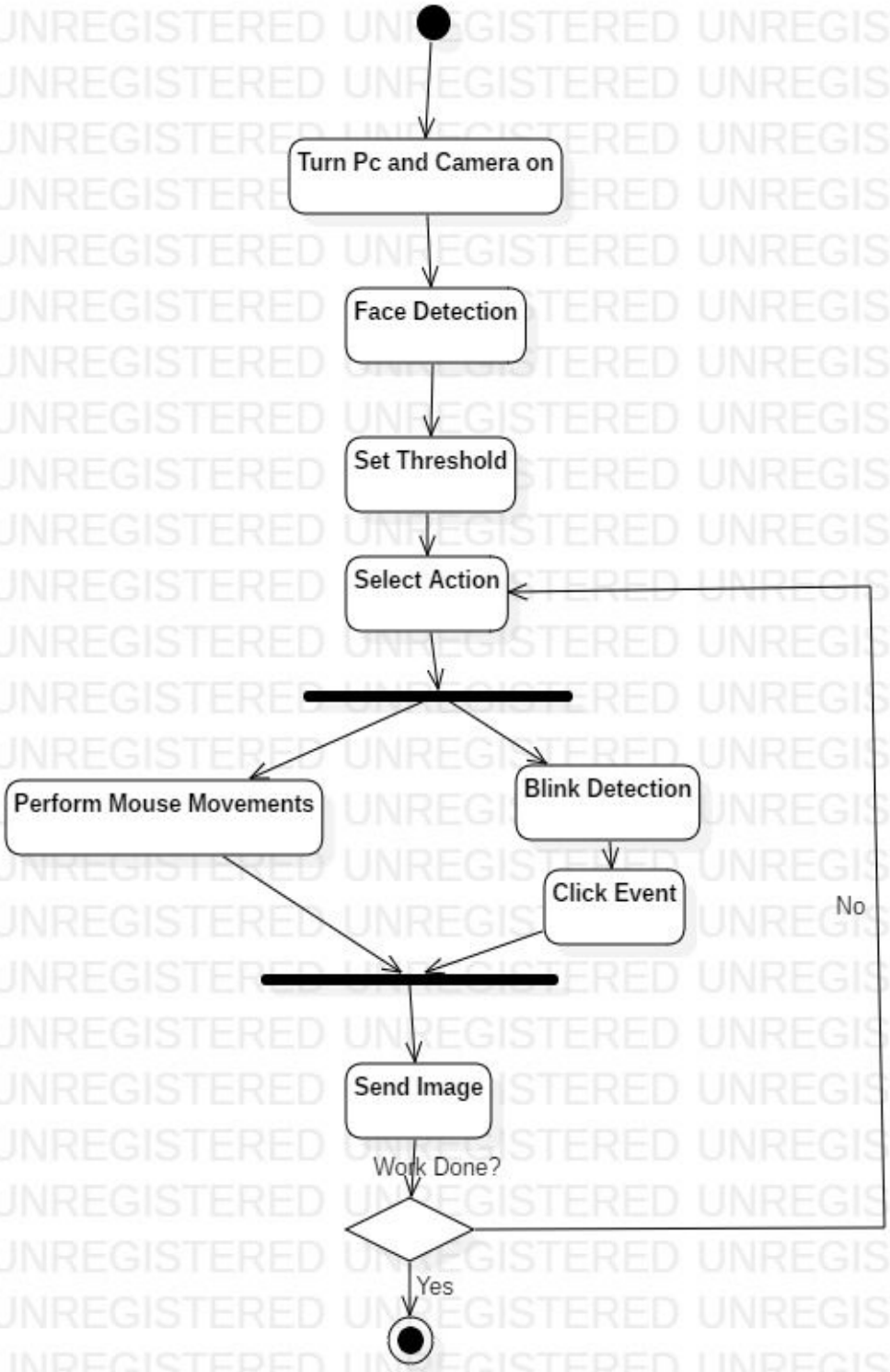

Turn Pc and Camera on
Face Detection
Set Threshold
Select Action
Perform Mouse Movements
Blink Detection
Click Event
No
Send Image
Work Done?
Yes

Fig. 4.21

4.3 Sequence Diagram

A sequence diagram in a Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart.

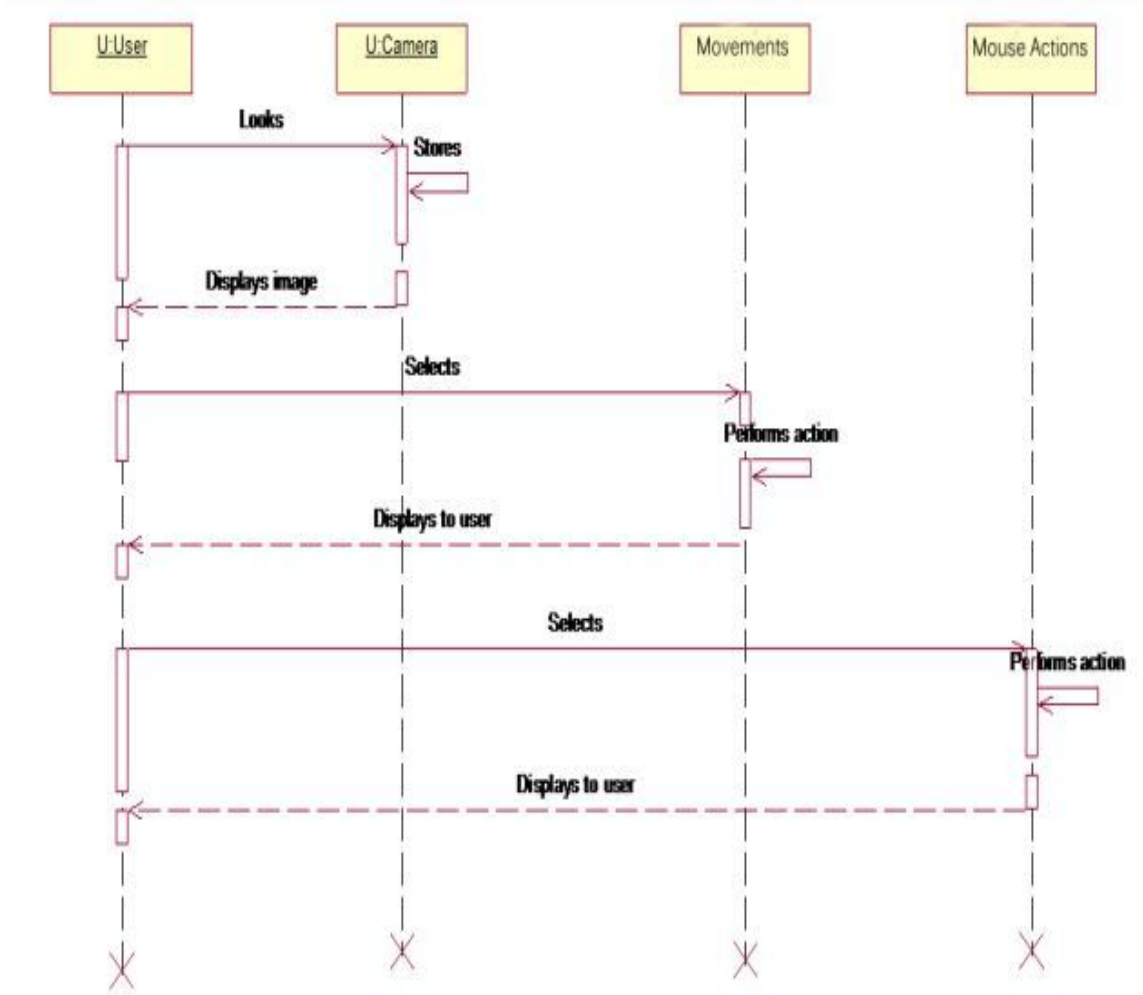


Fig. 4.31

CHAPTER 5

IMPLEMENTATION

5.1 IMPLEMENTATION DETAILS WITH CODE

Facial landmarks are used to localize and represent salient regions of the face, such as:

- Eyes
- Eyebrows
- Nose
- Mouth
- Jawline

Facial landmarks have been successfully applied to face alignment, head pose estimation, face swapping, blink detection and much more.

Detecting facial landmarks is therefore a two step process:

- Step #1: Localize the face in the image.
- Step #2: Detect the key facial structures on the face ROI.

Detecting facial landmarks is therefore a two step process:

- Step #1: Localize the face in the image.
- Step #2: Detect the key facial structures on the face ROI.

Face detection (Step #1) can be achieved in a number of ways.

We could use OpenCV's built-in Haar cascades.

We might apply a pre-trained HOG + Linear SVM object detector specifically for the task of face detection.

```
from scipy.spatial import distance as dist
import cv2
import imutils
from imutils import face_utils
import numpy as np
import dlib

```

Fig 5.11

Or we might even use deep learning-based algorithms for face localization.

In either case, the actual algorithm used to detect the face in the image doesn't matter. Instead, what's important is that through some method we obtain the face bounding box (i.e., the (x, y)-coordinates of the face in the image).

Given the face region we can then apply Step #2: detecting key facial structures in the face region.

There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions:

- Mouth
- Right eyebrow
- Left eyebrow
- Right eye
- Left eye
- Nose
- Jaw

The facial landmark detector included in the dlib library is an implementation of the One Millisecond Face Alignment with an Ensemble of Regression Trees paper by Kazemi and Sullivan (2014).

This method starts by using:

1. A training set of labeled facial landmarks on an image. These images are manually labeled, specifying specific (x, y)-coordinates of regions surrounding each facial structure.
2. Priors, of more specifically, the probability on distance between pairs of input pixels.

Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the pixel intensities themselves (i.e., no "feature extraction" is taking place).

The end result is a facial landmark detector that can be used to detect facial landmarks in real-time with high quality predictions.

For more information and details on this specific technique, be sure to read the paper by Kazemi and Sullivan linked to above, along with the official dlib announcement.

Understanding dlib's facial landmark detector

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.

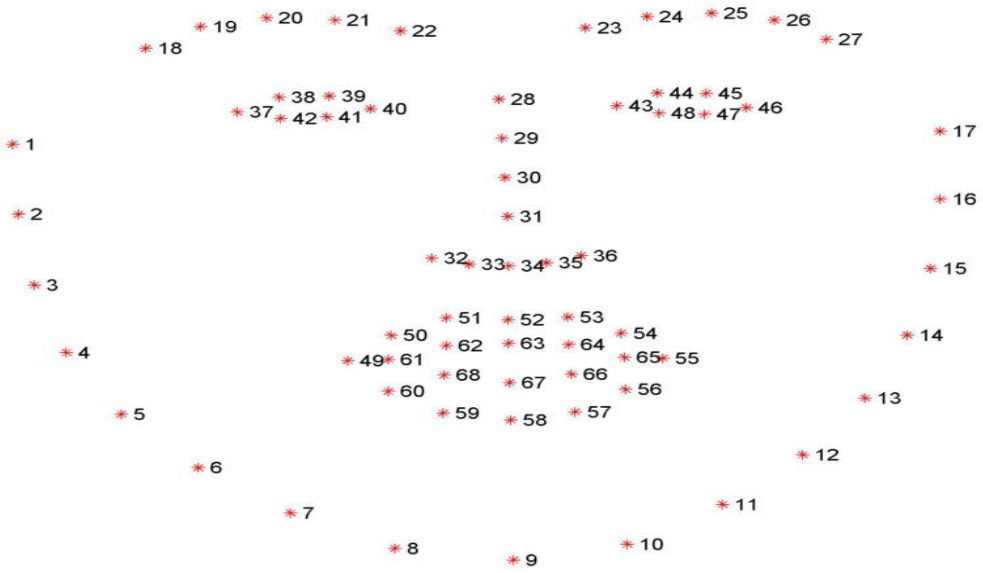


Fig 5.12

The indexes of the 68 coordinates can be visualized on the image below:

These annotations are part of the 68 point iBUG 300-W dataset which the dlib facial landmark predictor was trained on.

It's important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the HELEN dataset.

Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data — this is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

There is a relation between the width and the height of these coordinates.

Based on the work by Soukupová and Čech in their 2016 paper, Real-Time Eye Blink Detection using Facial Landmarks, we can then derive an equation that reflects this relation called the eye aspect ratio (EAR):

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

Fig 5.13

```
def EAR(eye):
    # distance between 2 set if vertical landmarks (x,y)-coordinates
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    # horizontal distance
    C = dist.euclidean(eye[0], eye[3])

    # computing EAR
    ear = (A+B)/(2.0*C)
    return ear
```

Fig 5.14

```
# defining 2 constants first as a Threshold EAR
# other for number of consecutive frames the eye must be beliow the threshold
EAR_thresh = 0.1899
EAR_consec_frames = 2

counter = 0
total = 0

# dlib's face detector
detector = dlib.get_frontal_face_detector()\
    # dlib's facial landmarks detector
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

Fig 5.17

5.2 ALGORITHM USED

First the system captures images by camera then detects the head area in the images. Let the origin coordinates (0, 0) And the horizontal and vertical coordinate are noted x and y respectively. The coordinate values are calculated in pixels. The rectangle which frames the face is the detected head area. We calculate the geometric center of the rectangle, and name it as head central coordinates, i.e. (Sx, Sy) . Then we can analyze the specific head

movement by time series relationship of the central coordinates. The algorithm for detection of head movements is described as below.

(1) Initialization: User sits up in front of the computer. Let the Head-Trace Mouse run. If the head is detected, the head signals in the first 3 seconds are initialized by statistical methods, and then the head central coordinates (Sx, Sy) of the standard head is calculated.

(2) Set threshold value: Determine the threshold value (Kx, Ky) based on experience.

(3) Judge the head movements: Analyze the images after initialization. The head central coordinates of one image is noted as (Cx, Cy).

(4) Standard head relocation: If the standard head has been detected in several continuous images, the average value of these head central coordinates will be calculated as the new head central coordinates (Sx, Sy) of the standard head.

(5) Go back to step (2)

CHAPTER 6

RESULTS

FACIAL LANDMARKS DETECTION

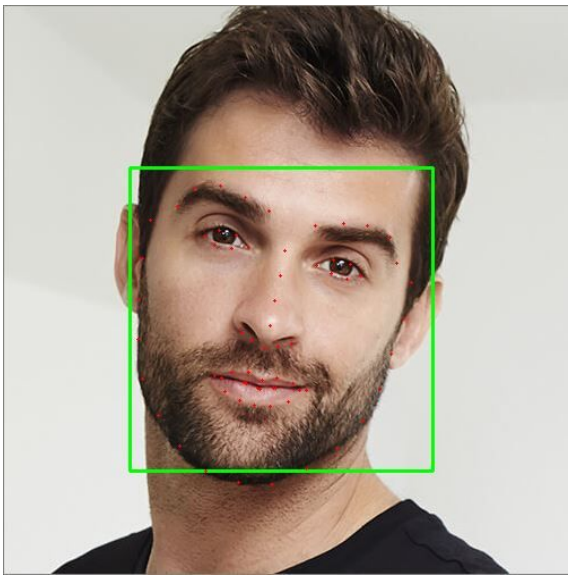


Fig 6.1

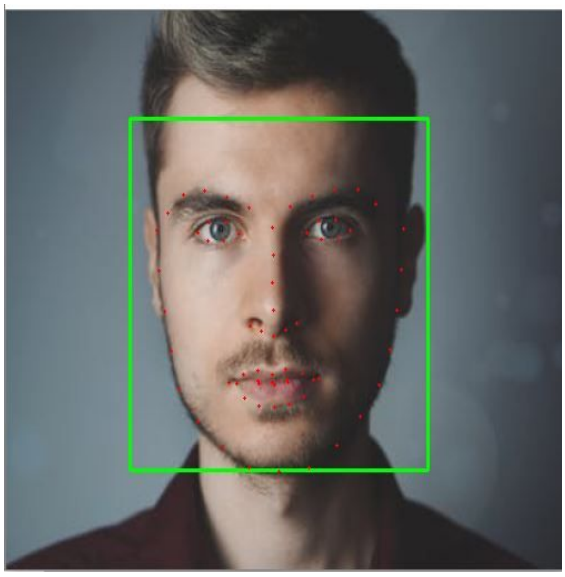


Fig 6.2

Fig 6.3

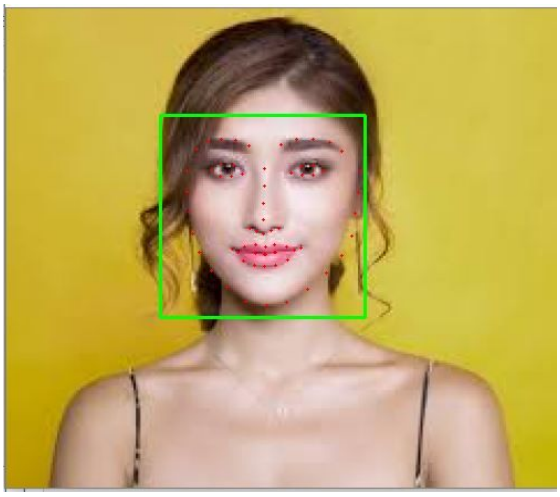


Fig 6.4

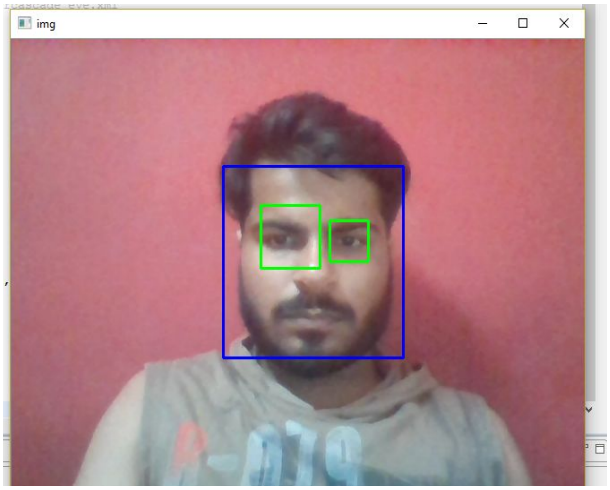


Fig 6.5

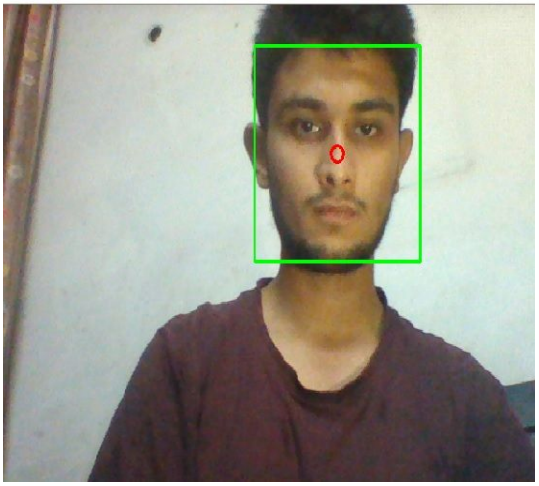
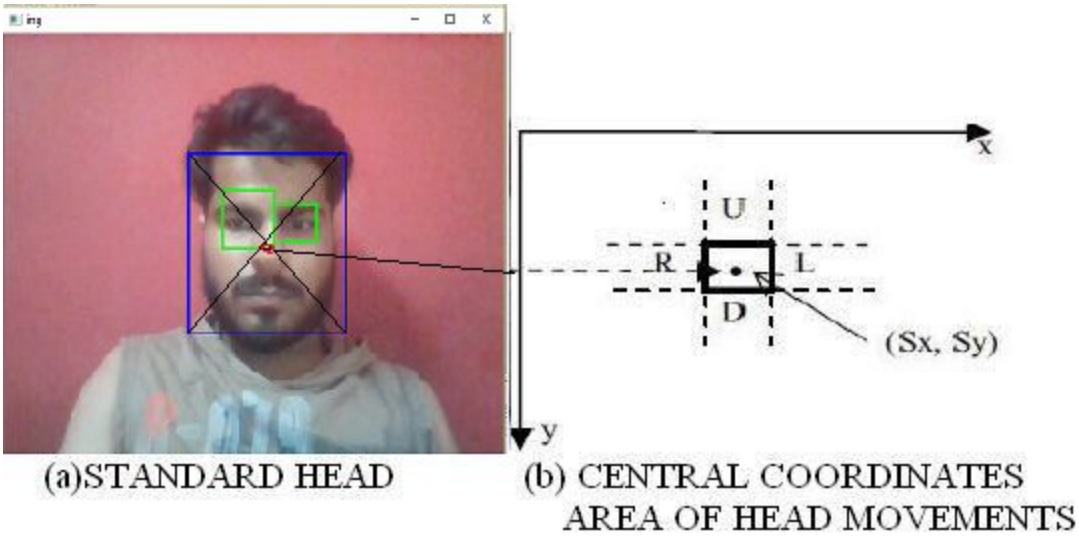


Fig 6.6



(a)STANDARD HEAD (b) CENTRAL COORDINATES AREA OF HEAD MOVEMENTS

Fig 6.7

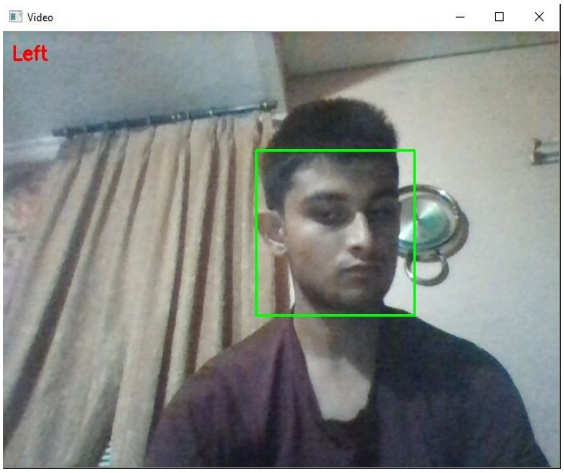


Fig 6.8

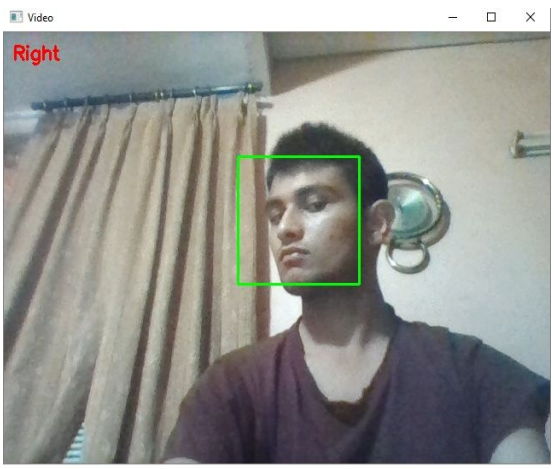


Fig 6.9

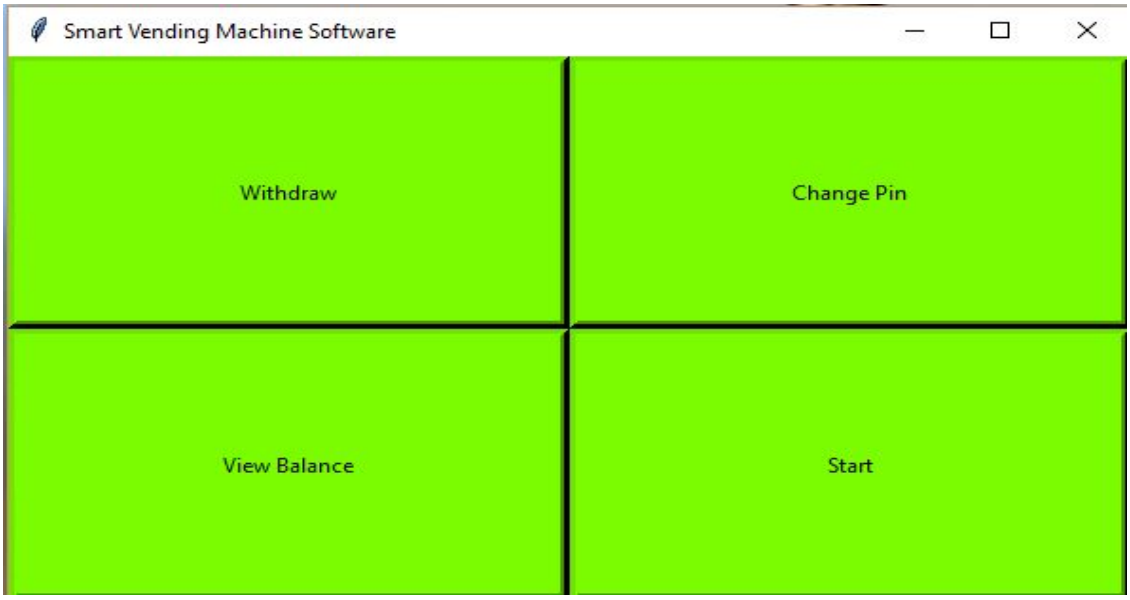


Fig 6.10

Fig 6.11

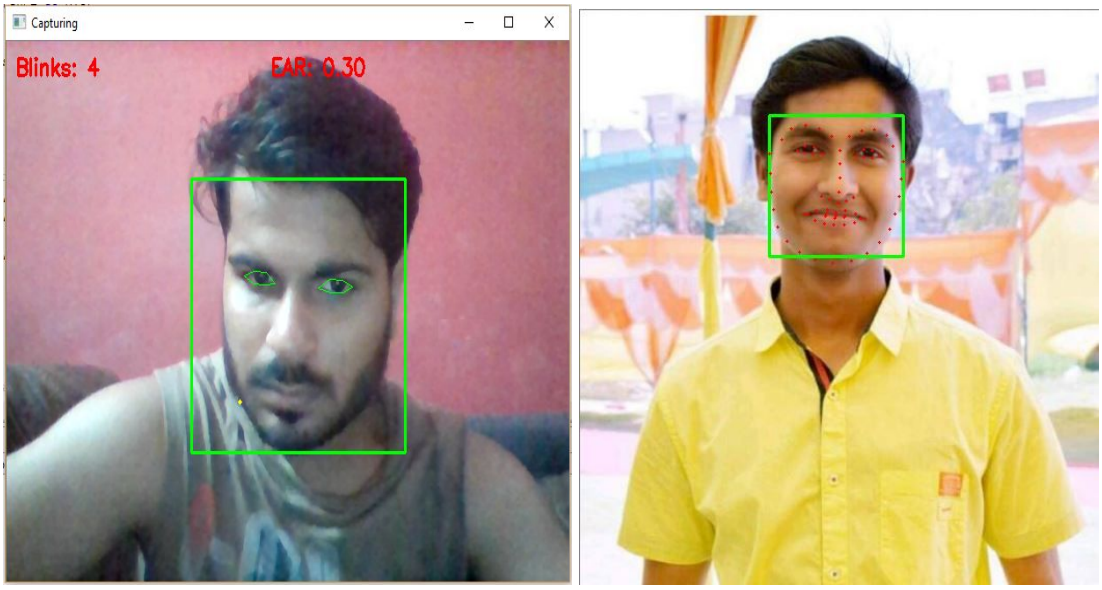


Fig 6.12

CHAPTER 7

CONCLUSION

We have implemented a system to access the mouse pointer on the computer screen using only Head and Eye movements. With the use of a camera and Python technology, the system architecture is prepared. User is able to view head and eye movements captured through the camera which is displayed on the screen, accordingly the user can move the mouse pointer as needed and also perform various mouse actions. The proposed system is feature based thus allowing any user to use the system without prior registration. This system is especially useful for the upper limb disabled.

Currently, we are extending our implementation to support keyboard press technology for the ease of the User to use the Keyboard hands free along with the already existing mouse movements provided by the system. This would then enable the User to access the computer owing to only facial features and movements without the use of traditional mouse and keyboard i.e Hands free system.

CHAPTER 8

REFERENCES

BIBLIOGRAPHY:

- P. Viola and M. Jones, “Computer Vision and Pattern Recognition” 2001

- T. Hutchinson, K. P. White Jr., W. N. Martin, K. C. Reichert, and L. A.Frey,“Human-computer interaction using eye-gaze input” 1989.

IEEE PAPERS

[1] D. G. Evans, R. Drew, and P. Blenkhorn, “Controlling mouse pointer position using an infrared head-operated joystick,”2000.

[2] M. Betke, J. Gips, and P. Fleming, “The Camera Mouse: Visual Tracking of Body

Features to Provide Computer Access for People With Severe Disabilities,” On Neural Systems And Rehabilitation Engineering, March 2002.

[3] M. Nabati and A. Behrad, “Camera Mouse Implementation Using 3D Head Pose

Estimation by Monocular Video Camera and 2D to 3D Point and Line Correspondences,” 2010 5th International Symposium on Telecommunications (IST'2010), 2010.

[4] Y. L. Chen, F. T. Tang, W. H. Chang, M. K. Wong, Y. Y. Shih, and T.S. Kuo, “The new design of an infrared controlled human-computer interface for the disabled,” Dec. 1999

WEBSITES

1) https://pypi.org/project/pyobjc-framework-Quartz/
2) https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/
3) https://docs.python.org/2/library/tkinter.html
4) https://docs.python.org/3/
5) https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html
6) https://github.com/manikantanallagatla/Head-Motion-Detection-and-Tracking/tree/master/head-motion-tracking-master