Asyut University
Faculty of computers and information

Graduation Project Report
(2021-2022)

OTS
(Optical Touch Screen)

Graduation Project Report

OTS
(Optical Touch Screen)

by

- Fares Ahmed
- Habeba Ayman
- Ibrahim Mohamed
- Reem Ashraf
- Salma Osama
- Yousef Barty
- Yousef Shaban

Supervised by

Prof. Abdel-Rahman Hedar

Contents

Abstract

Head position and eye tracking can be regarded as a pivotal real-time input medium for human-computer communication, which is especially important for people with physical disability. To improve the reliability, mobility, and usability of head and eye tracking technique in user-computer dialogue, a novel head and eye control system with integrating both mouse and keyboard functions. The proposed system focuses on providing a simple and convenient interactive mode by only using user‘s head and eye. Head and eye tracking in real-time was developed in the last few years as a challenging problem for computer vision. Hands-free computing is becoming an important area of research as it addresses the needs of quadriplegics This paper presents a Human-computer interaction (HCI) system that is of great importancefor amputees and those who have issues with using their hands. The proposed system is an interface that reads the head movement and eye blinking, in which the head acts as a computer mouse. This method is based on estimating head-position and direction (The left, right head directions). Eye blinking was determined also by calculating the midpoint of middle landmarks of the eye, then the system employs eyeblinking to write on a virtual keyboard and show some of suggested words or moving the cursor to select objects with head movement. The proposed system uses a simple webcam and its software requirements. Less than one second was spent on writing the letter by the proposed method using eye-controlled keyboard which is less time than other related research and fast movement of cursor in real time using head movement.

1

Introduction

Chapter Summary: This chapter discusses the topic of the research, identifies the beneficiaries and the purpose of the study, and gives an overview of the chapters in a simplified manner.

Introduction

Head position and Eye-tracking are technologies that make it possible for a computer or other device to know where a person is looking. Head and eye tracker can detect the attention and focus of the user. they allow for unique insights into human behavior and facilitates natural user interfaces in a broad range of devices. The ability to control a computer using the head and eyes is also vital for people who are unable to speak or use their hands [1]. The study of eye movement emerged from this technique. Common research on eye movement has traditionally centered largely on the fields of behavioral psychology and medicine. In recent years, scientists have recognized the great potential of machine eye movement and combined eye movement with computer vision technology to detect attention. Technology, eye tracking technology, etc... The research of these vision technologies involves optics, computer science, physiology and other fields, including the detection, localization, tracking and other physiological features of human face, human eye and iris. [2]. There are different types of eye tracking device and the most appropriate one for the user depends on the nature of research.

The main types are:

Screen based: These are stand-alone, remote devices that either come as an individual unit or a smaller panel that can be attached to a laptop or monitor.

Wearable: These include eye-tracking glasses and virtual reality (VR) headsets with integrated eye-tracking.

Webcam: Webcam eye trackers do not have sensors or specialized cameras, they are solely comprised of the webcam device attached or built-in to a computer [3], which used in this study. Webcam eye-tracking requires the built-in or external webcam that is connected to a laptop or computer to collect details on where the user is looking. This approach does not contain rays of infrared light, instead, it relies on the image generated from the webcam. An algorithm is used to get the position of the head and eyes, and then the direction of the eyes is correlated to an image on the Webcam eye-tracking requires the built-in or external webcam that is connected to a laptop or computer to collect details on where the user is looking. This approach does not contain rays of infrared light; instead, it relies on the image generated from the webcam. The Python language is one of the most suitable programming languages for building an application that depends on eye movement as it contains a library called OpenCV and it's defined as an open-source computer vision and machine learning library. It was built with a vision to provide basic infrastructure to the computer vision application [4]. The process of determining eye directions by OpenCV library is expressed in general as shown in figure1.

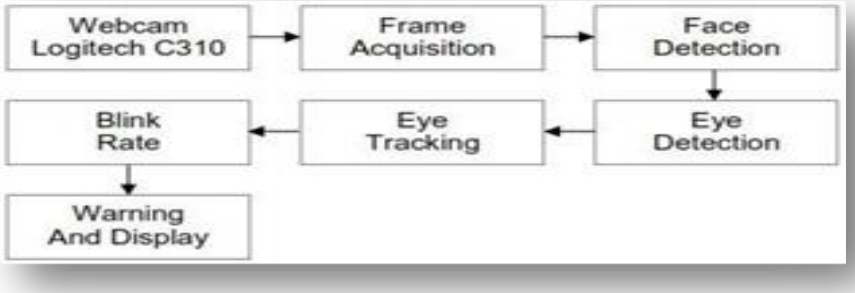


Figure 1.0: Main steps for eye tracking [8]

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, track camera movements, follow eye movements, recognize scenery, etc.; besides, the applications that are designed using this library are suitable for running on computers and smartphones [5]. The objective of this research is trying to help people who have issues with using their hands in writing-based eye blinking or mouse-control using head movement using simple webcam. This proposed system does not contain any electronic tools and based on programming only so its cost is very low.

1.1. Project Literature Survey

The technology, which allows extensive use of the principle of eye- tracking, includes sectors in the automotive industry, medical science, exhaustion simulation, automobile simulators, cognitive tests, computer vision, behavior recognition, etc. Over a span of time the importance of eye recognition and monitoring in industrial applications grew. This importance of eye- tracking applications lead to more effective and durable designs that are required in many modern appliances. An extensive review of the literature relating to the eye-tracking system has been carried out in the field of healthcare applications. In [6], a proposed eye tracking system is presented, which based on extracting frame from captured video then the image is converted from the RGB format to a single dimension grayscale. Gamma correction is applied to the image to correct its brightness and contrast. The eye region is detected by using an algorithm is called the Viola-Jones algorithm the result of the experiment is that the elapsed time of gaze detection is 33.82 second. The authors in [7] proposed a system starts with a webcam to capture the eye images and send a signal to an electronic circuit is called Raspberry Pi for digital image processing which is based on OpenCV library to detect the 2D motion direction of the eyeball. Then the author uses the motion of the eyeball as the cursor control on the Raspberry Pi screen to control some devices such as wheelchair control, electrical device control, and sending a message to the smartphone. The component of the system is the eyeglass connected with webcam and the webcam is connected to a Raspberry Pi microprocessor. The result of the experiment is that the elapsed time of gaze detection is 5.2 s, but the cost was very high. A proposed algorithm, introduced in [8], which performs mouse functions by providing a hands-free interaction between computers and humans. It offers an alternative to the conventional machine with a mouse. By using various expressions of a face using computer vision and matching it with the expression already stored and performing actions as per the step. This algorithm can help visually disabled persons use their face and eye movement to execute the functions of a mouse. It makes it possible for them to left- click, right-click, scroll up and down, move up, down, left, right the cursor. Webcam, NumPy, Dlib, and several other specific libraries have a very simple need for the framework.

The authors in [9] proposed a nurse call system driven by image processing of user's eye movements. The system composed of a computer, a camera, two light emitting diode (LED) lamps and a speaker is capable of non-contact/non- constraint operation. According to the predetermined pattern, a user shifts his or her eyes, then the system extracts from the image certain patterns of eye movements. After detecting those

patterns, the system makes a high sound as same as the role of call button at bedside in hospital. A user can always understand the current condition of the system operation as the flashing pattern of two LED lamps during system execution. Using the data reported from normal healthy subjects, the development and evaluation of the algorithm for detecting eye motion patterns from the image was carried out. The Developed prototype system was used in a subject with movement disorders caused from nerve degeneration. In [10], a proposed system is presented to assist people with mobility impairments by restoring their ability to move effectively and effortlessly without having to depend on others who use an electric, eye-controlled wheelchair. The system was images of eyes of users, which were processed to evaluate gaze orientation, and the wheelchair was moved accordingly. To achieve such a feat, a robotic with the aid of a series of proximity sensors, wheelchair control will override any decision taken by the gaze assessor and stop its movement immediately, if the measured distance is less than a well-defined safety margin. The algorithm was working with 90% accuracy. The work in [11] aims to develop an EOG-based virtual keyboard, which uses a threshold-based method for detecting eye movements from EOG signals in real time, then categorized as saccades or flashes using a new series of sequences. An average blink marking accuracy of 99.92% and 100.00%, was obtained by the two categorized respectively, indicating

that the two eyes' movements can be reliably detected and distinguished in real time using the proposed algorithm. When these technologies were used to interact with the EOG-based asynchronous virtual keyboard, an average typing speed across subjects was achieved from the equivalent of 4.42 characters per minute.

1.1.1. Project Objectives

- Detects users‘ head pose estimation to begin the next process.
- Tracks the movement of users‘ head to record the new of position of users‘ head.
- Writes the letter according to the blink that has been detected.
- Moves to the position the user wants according to the head movement.
- The object selected will do the proposed event after blinking.

1.1.2. Project Aims

The main purpose in our project is to Enable disable people writing or moving mouse cursor using their eyes and head

1- Our project helps Muscular dystrophy people who cannot write text or do anything.

2- A person with disabilities can perform a wide range of tasks through a computer that help them to communicate more effectively and live fuller, more independent lives.

3- We have developed a new methodology to investigate writing that we believe addresses many of shortcomings in people.

1.1.3. Project Phases

Analysis:

- Plan
- Data gathering
- Transform
- Store
- Test
- Publish

Design:

- Requirement Engineering
- ML use-case prioritization
- Data availability check

Implementation:

- Data Engineering

1.1.4 Project Methodology

1. **Face detection methods**

Face detection is the step stone for all facial analytical algorithms, including facial orientation, facial mapping, face recognition, facial authentication, facial expression tracking / recognition, gender recognition.

2. **Feature based methods**

The characteristic-based approach is to identify faces by extracting facial structural features.

3. **Eye Detection approaches**

It is first classifier and then used to distinguish between facial and non-facial regions. There are some detection methods that undergo various stages, such as the positioning of the face and eyes from different locations.

4. **Head Pose Estimation**

The input of head pose is taken from the individual's face. If a person looks at a center mouse pointer, that point would be taken as the input point, and it sets that location as the basis for head pose tracking and it begins moving in the direction of the person's head movement and the cursor stops moving when the person's head stop moving.

5. **Controlling the mouse pointer**

If the head moves in the left direction, the mouse pointer moves in the left direction as well, and the same happens in the right direction as well.

6. **Controlling the mouse click events**

Slightly half-closed state when eye downwards. This phenomenon can be used to guide the step from top to bottom of the mouse pointer. A blink of an eye can be done to the mouse click case. As the mouse pointer begins to shift the eye, pupil's path if a person decides to perform a chosen event, then the person blinks the eye mouse pointer executes the click event.

7. **Writing on Virtual keyboard**

User can write on keyboard when the mouse cursor place on specific letter. And blink his eyes for ten frames the letter selected will be written.

1.2. Problem Definition

Dysgraphia is a deficiency in the ability to write, primarily handwriting, but also coherence. Dysgraphia is a specific learning disability (SLD) as well as a transcription disability, meaning that it is a writing disorder associated with impaired handwriting, orthographic coding, and finger sequencing (the movement of muscles required to write). It often overlaps with other learning disabilities such as speech impairment, attention deficit hyperactivity disorder (ADHD) or developmental coordination disorder (DCD). In the Diagnostic and Statistical Manual of Mental Disorders (DSM-IV), dysgraphia is characterized as a learning disability in the category of written expression when one's writing skills are below those expected given a person's age measured through intelligence and age-appropriate education. The DSM is not clear in whether.

Writing refers only to the motor skills involved in writing, or if it also includes orthographic skills and spelling.

The word dysgraphia comes from the Greek words dyes meaning "impaired" and γραφία graphic meaning "writing by hand". There are at least two stages in the act of writing: the linguistic stage and the motor-expressive-praxis stage. The linguistic stage involves the encoding of auditory and visual information into symbols for letters and written words. This is mediated through the angular gyrus, which provides the linguistic rules which guide writing. The motor stage is where the expression of written words or graphemes is articulated. This stage is mediated by Exner's writing area of the frontal lobe. The condition can cause individuals to struggle with feedback & anticipating and exercising control over rhythm & timing throughout the writing process.

People with dysgraphia can often write on some level and may have trouble with other activities requiring reciprocal movement of their fingers and other fine motor skills, such as tying shoes, fastening buttons or playing certain musical instruments. However, dysgraphia does not affect all fine motor skills. People with dysgraphia often have unusual difficulty with handwriting and spelling which in turn can cause writing fatigue. Unlike people without transcription disabilities, they tend to fail to preserve the size and shape of the letters they produce if they can‘t look at what they are writing. They may lack basic grammar and spelling skills (for example, having difficulties with the letters p, q, b, and d), and often will write the wrong word when trying to formulate their thoughts on paper. The disorder generally emerges when the child is first introduced to writing. There is accumulating evidence that individuals with SLDs and DCD do not outgrow their disorders. Accordingly, it‘s been found that adults, teenagers.

1.3. Overview of the Project Document

- Chapter 1: Introduction
  Discusses the research idea in all aspects, and the purpose of the study, also who will benefit and why?

- Chapter 2: Software Analysis
  Explain more information about system in details

- Chapter 3: System Modeling and Design
  Presents the design of the project diagrams and initial interface for the system.

- Chapter 4: System Implementation
  Presents the Implementation of our versions.

2

Software Analysis

Chapter Summary:
This chapter analyze our system in details.

2.1. What's OpenCV?

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. Mainly used for computer vision, machine learning, and image processing. The library has more than 2500 optimized algorithms and it helps to process images and videos to identify objects, faces, handwriting, track camera movements, stitch images together, find similar images from an image database and lot more. OpenCV has some classifiers to identify the object types. There are few classifiers available such as HAAR cascade classifier. The classifier is an xml file and has lot of definitions/patterns inside. When some object matches with those defined patterns then our code will identify and categorizes that object. Below are some sample patterns.

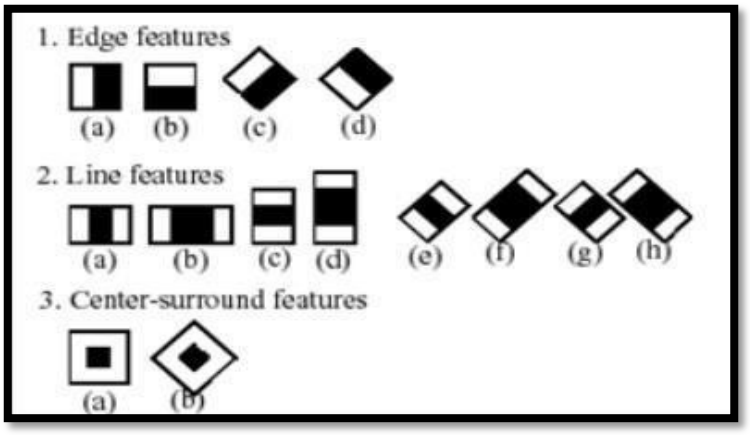


Figure 2.1: some sample patterns of HAAR cascade [10]

To make it simpler let's consider the Car image from the above pic and try to apply the defined patterns to see in case any one of the combinations justify the image. Therefore, it does, and we can say it's a car.

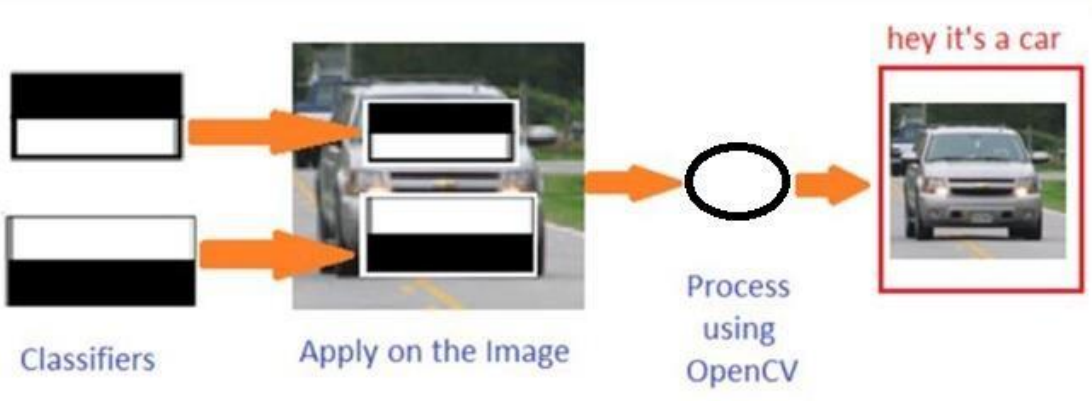


Figure 2.2: car image using haar cascade models [7]

2.1.1. Detect things using Haar Cascade

When we say Haar cascades, we are talking about cascade classifiers based on Haar features. To understand what this means, we need to take a step back and understand why we need in the first place. Back in 2001, Paul Viola and Michael Jones came up with a very effective object detection method in their seminal paper. It has become one of the major landmarks in the field of machine learning. Let's say we want to detect an object like, say, a pineapple. To solve this, we need to build a machine learning system that will learn what a pineapple looks like. It should be able to tell us if an unknown image contains a pineapple or not. To achieve something like this, we need to train our system. In the realm of machine learning, we have a lot of methods available to train a system. It's a lot like training a dog, except that it won't fetch the ball for you! To train our system, we take a lot of pineapple and non-pineapple images, and then feed them into the system. Here, pineapple images are called positive images and the non-pineapple images are called negative images. As far as the training is concerned, there are a lot of routes available. But all the traditional techniques are computationally intensive and result in complex models. We cannot use these models to build a real time system. Hence, we need to keep the classifier simple. But if we keep the classifier simple, it will not be accurate. The tradeoff between speed and accuracy is common in machine learning. We overcome this problem by building a set of simple classifiers and then cascading them together to form a unified classifier that's robust. To make sure that the overall classifier works well, we need to get creative in the cascading step. This is one of the main reasons why the Viola- Jones method is so effective. Coming to the topic of face detection, let's see how to train a system to detect faces. If we want to build a machine learning system, we first need to extract features from all the images. In our case, the machine learning algorithms will use these features to learn what a face looks like. We use Haar features to build our feature vectors. Haar features are simple summations and differences of patches across the image. We do this at multiple image sizes to make sure our system is scale invariant. Once we extract these features, we pass it through a cascade of classifiers. We just check all the different rectangular sub-regions and keep discarding the ones that don't have faces in them. This way, we arrive at the final answer quickly to see if a given rectangle contains a face or not.

2.1.1.1. Face Detection

We need a classifier model that can be used to detect the faces in an image. OpenCV provides an xml file that can be used for this purpose. We use the function Cascade Classifier to load the xml file. Once we start capturing the input frames from the webcam, we convert it to grayscale and use the function detect Multi Scale to get the bounding boxes for all the faces in the current image. The second argument in this function specifies the jump in the scaling factor. As in, if we don't find an image in the current scale, the next size to check will be, in our case, 1.1 times bigger than the current size. The last parameter is a threshold that specifies the number of adjacent rectangles needed to keep the current rectangle. It can be used to increase the robustness of the face detector.

2.1.1.2. Eye Detection

Now that we understand how to detect faces, we can generalize the concept to detect other body parts too. It's important to understand that Viola-Jones framework can be applied to any object. The accuracy and robustness will depend on the uniqueness of the object. For example, a human face has unique characteristics, so it's easy to train our system to be robust. On the other hand, an object like towel is too generic, and there are no distinguishing characteristics as such; so it's more difficult to build a robust towel detector.

2.1.2. Introduction to Head Pose Estimation

Head pose estimation means detecting the position of a human head in the image. Particularly, it means detecting the head‘s Euler angles – yaw, pitch and roll. They define the object‘s rotation in a 3D environment. If you make the right prediction about these three, you‘ll find out which direction the human head will be facing.

2.1.3 Introduction to Dlib

The computer engineer researching how they identify the face of a human in an image. For this, we need to identify first were.

2.1.3.1. Dlib’s 68 face features

The below image is an example of a Dlib's 68 points model. There we can see those points from1 to 68. But we don't need all 68 feature points, then for that, we will show, how we can customize those points according to our requirements.

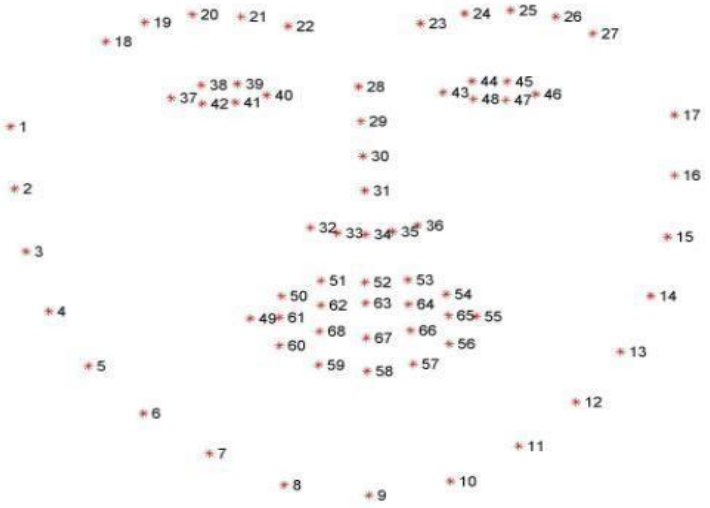


Figure 2.4: points using Dlip in face detection

2.1.3.2. Face Detection

It is a technology of recognizing human faces from any image or video. Mostly, OpenCV and dlib are used to detect face by using various methods. The detector used here is made up of classic Histogram of Oriented Gradients (HOG) feature along with a linear classifier. Facial landmarks detector is implemented inside dlib to detect facial features like eyes, ear, nose etc.

2.1.3.3. Eye Detection

After detecting the face, eye region is detected with the help of facial landmark features. Using the face landmarks dataset, we can point out 68 landmarks on the face. Each landmark is assigned with an index. Using these indices, the desired region of the face is detected,

Point index for two eyes:

- **left eye:** - (37, 38, 39, 40, 41, 42)
- **right eye:** - (43, 44, 45, 46, 47,48)

After extracting eye region, it is processed for detecting eye blinks. The eye region detection is done at the initial stage of the system

2.1.3.4 Eye Blink Detection

With the exact eye region, we can detect the blinks with the help of two lines. One drawn horizontally and other drawn vertically splitting the eyes. Temporary closure of eyes along with the movement of eyelids is known as blink. It is a rapid natural process. We have to find out what happens when eye is blinked. We can conclude that the eye is closed/blinked when:

- **Eyeball is not visible**
- **Eye lid is closed**
- **upper and lower eyelids are connected together**

If these actions are occurred for a period of (approximately 0.3 seconds to 0.4 seconds) time, we can assume it as a blink if it is longer than that then it can be taken as closed eyes. For an opened eye, both vertical and horizontal lines are almost identical while for a closed eye, vertical line becomes very smaller or almost vanished. Keeping the horizontal line as point of reference, a ratio is computed with respect to vertical line. We have to set a threshold value here and if the ratio is greater than the value then we can assume the eye is closed otherwise open. Each eye is represented

by 6 (x, y)-coordinates, starting at the left-corner of the eye (as if you were looking at the person), and then working clockwise around the remainder of the region Based on this image, we should take away on key point:

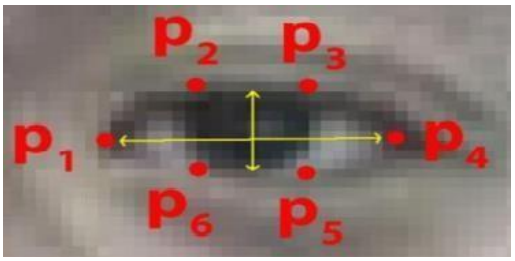


Figure 2.5: detect blink eyes [3]

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 relationcalled the eye aspect ratio

(EAR): $$EAR = \frac{\| p_2 - p_6 \| + \| p_3 - p_5 \|}{2 * \| p_1 - p_4 \|}$$

Where p1, …, p6 are 2D facial landmark locations. The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighting the denominator appropriately since there is only one set of horizontal points but two sets of vertical points. Well, as we'll find out, the eye aspect ratio is approximately constant while the eye is open but will rapidly fall to zero when a blink is taking place. Using this simple equation, we can avoid image processing techniques and simply rely on the ratio of eye landmark distances to determine if a person is blinking. To make this clearer, consider the following figure from Soukupová and Čech:

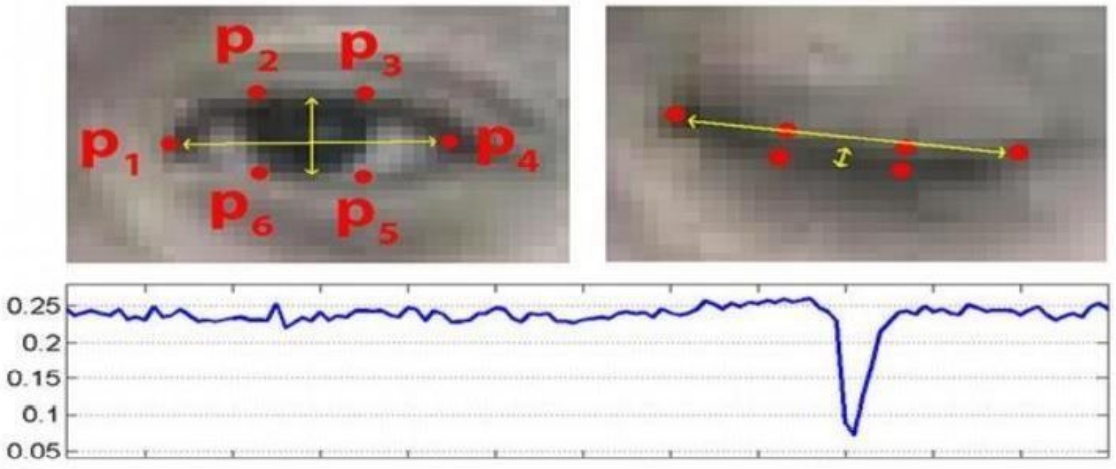


Figure 2.6: to calculate ratio of blank [7]

2.1.4. Introduction to MediaPipe

1. MediaPipe way faster, has more landmarks, it can reach up to 30 fps as well. While dlib reach about 10fps.
2. MediaPipe is easy to install, just with pip command

3. MediaPipe does not require extra things, visual studio, cmake, etc. on windows machine,

Mechanism: Mediapipe Face Mesh is a face geometry solution that estimates 468 3D face landmarks in real-time even on mobile devices. It employs machine learning (ML) to infer the 3D surface geometry, requiring only a single camera input without the need for a dedicated depth sensor. Utilizing lightweight model architectures together with GPU acceleration throughout the pipeline, the solution delivers real-time performance critical for live experiences.

The pipeline consists of two real-time deep neural network models that work together: A detector that operates on the full image and computes face locations and a 3D face landmark model that operates on those locations and predicts the approximate surface geometry via regression. Having the face accurately cropped drastically reduces the need for common data augmentations like affine transformations consisting of rotations, translation, and scale changes. Instead, it allows the network to dedicate most of its capacity towards coordinate prediction accuracy.

The output of the pipeline is a set of 478 3D landmarks, including 468 face landmarks from Mediapipe Face Mesh, with thosearound the eyes further refined (see Fig 2.7)

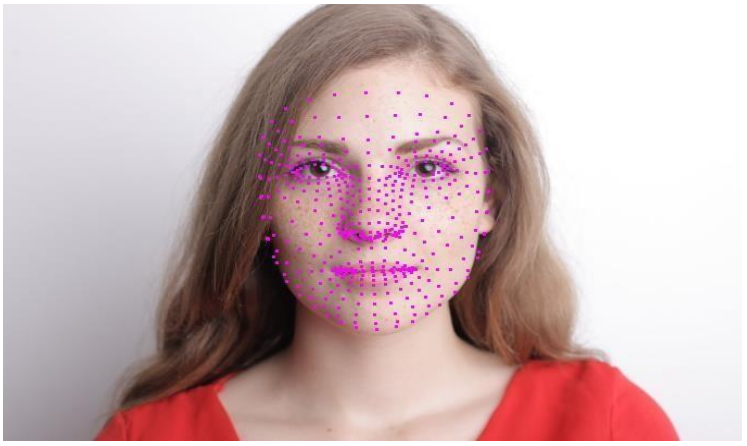


Figure 2.7: 468 face landmarks from Mediapipe Face Mesh [5]

2.1.4.1. Iris Detection

Mediapipe Iris is a ML solution for accurate iris estimation, able to track landmarks involving the iris, pupil and the eye contours using a single RGB camera, in real-time, without the need for specialized hardware. Through use of iris landmarks, the solution is also able to determine the metric distance between the subject and the camera with relative error less than 10%.

Mechanism: The first step in the pipeline leverages Mediapipe Face Mesh, which generates a mesh of the approximate face geometry. From this mesh, we isolate the eye region in the original image for use in the subsequent iris-tracking step.

The iris model takes an image patch of the eye region and estimates both the eye landmarks (along the eyelid) and irislandmarks (along this iris contour).

Figure 2.8: irislandmarks [5]

2.1.4.2. Iris and Depth

MediaPipe Iris can determine the metric distance of a subject to the camera with less than 10% error, without requiringany specialized hardware

2.1.4.3. Advanced information about MediaPipe Iris

A 3-megabyte model to predict 2D eye, eyebrow and iris geometry from monocular video captured by a front-facing camera on a smartphone in real time. This model is intended to be used in combination with Mediapipe Face Mesh.

Model Type/Model Architecture

- Convolutional Neural NetworkModel Architecture
- Convolutional Neural Network: MobileNetV2-like with customized blocks for real-time performance.

Input(s)

- Image of proportionally cropped left eye with eyebrow (or horizontally flipped right eye) with a 25% margin on each side andsize 64x64

Output(s)

- Eye with eyebrow region surface represented as 71 2D landmarks flattened into a 1D tensor: (x1, y1), (x2, y2), x- and y- coordinates follow the image pixel coordinates.
- Iris surface represented as 5 2D landmarks (1 for pupil center and 4 for iris contour) also represented as flattened 1D tensor.

1-Limitations on mediapipe

INPUTS

Videos should be captured in "selfie" mode. As such, it's not suitable for detection when:

- Face is directed away from the camera (more than 60° when eye is no longer visible),
- Face is inclined from the vertical orientation (more than 8°),
- Eye is only partially visible (less than 50%) or no iris (eye is closed),

- Eye is located too far away from the camera (cropped eye can't be rescaled to model input of 64x64 without qualitydegradation).

ENVIRONMENT

When degrading the environment conditions (very dark or noisy camera video, video with a lot of motion or significant eye overlap) one can expect degradation of quality and increase of ‒jittering‖ (although we cover such cases during training with real-world samples and augmentations)

PRESENCE OF ATTRIBUTES

The model is intended to be used primarily in the tracking mode that guarantees certain accuracy of the eye location; scale androtation (see specification in ―Attributes").

2-Factors and Subgroups

INSTRUMENTATION

- All dataset images were captured on a diverse set of smartphone cameras, both front- and back-facing.
- All images were captured in real-world environments with different light, noise and motion conditions via an AR (AugmentedReality) application.

ENVIRONMENTS

Model is trained on images with various lighting, noise, and motion conditions and with diverse augmentations. However, its quality can degrade in extreme conditions (specified in "Limitations Environment"). This may lead to increased ‒jittering‖ (inter-frame prediction noise).

ATTRIBUTES

- Eye region cropped from the captured frame should contain a single left eye with eyebrow placed in the center of the image.
- There should be a margin around the eye region calculated as 25% of eye region size. Margin should be applied to a minimal proportional bounding box enclosing eye and eyebrow.
- Image must be rotated in a way that a horizontal line can connect the two corners of the eye.
- Model is tolerant to certain level of input inaccuracy:
  - 10% shift and scale (taking eye region width/height as 100% for corresponding axis)
  - 8° roll

3-Metrics (Model Performance Measures)

IC MAE, WWD MAE

For quality and fairness evaluation, we use IC MAE for eye (Mean Absolute Error normalized by Interconvert Distance) and WWD MAE for iris (Mean Absolute Error normalized by White-to white diameter).

MEAN ABSOLUTE ERROR

Mean absolute error is calculated as the pixel distance between ground truths and predicted landmarks. In order to make a fair comparison with human annotators we use 2D coordinates predicted by the model.

NORMALIZATION BY IC

Normalization by intercorner distance is applied to unify the scale of the eyes and is taken as 100%. Unified scale allows to calculate correct average metrics per-region and overall. IC is calculated as the 3D distance between the eye corners taken from the ground truth (3D is used to accommodate for head rotations).

NORMALIZATION BY WWD

Normalization by white-to-white diameter is applied to unify scale of the irises and is taken as %100. WWD is calculated as the 3D distance between the left and right landmarks on iris contour taken from the ground truth (3D is used to accommodate for head rotations).

4-Evaluation Modes

TRACKING MODE

This is the main mode that takes place most of the time. It is based on obtaining a highly accurate eye region crop from the prediction on the previous frame and a translation (crop center) from the face mesh on the current frame (frames 2, 3, ... on theimage below).

REACQUISITION MODE

Takes place when there is no information about the eye region from previous frames. It happens either on the first frame (image below) or on the frames when face tracking is lost. In this case, we use the Face Mesh model prediction to obtain eye region crop and 2D translation.

2.1.4.4. Eye Blink Detection

in order to detect a blink, we have to consider few things, first of all, here is an image, which shows you the differencebetween closed and open Eyes, with landmarks draw.

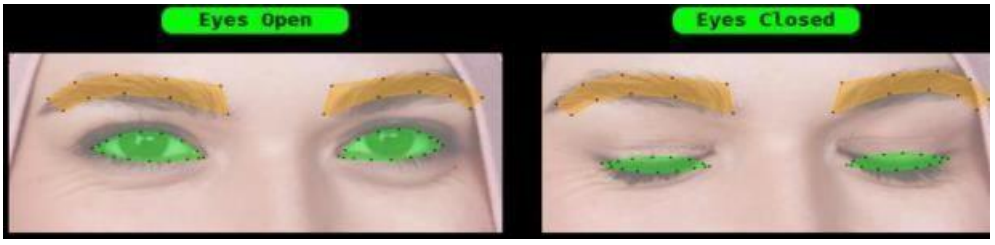


Figure 2.9: eye blink detection using Mediapipe [10]

When focusing on eyes it clearly shows that, when eyes are open, width and height of eyes landmarks at max, but when it closed, their no effect on the Width of eyes, but changes happing in the Height of Eyes, that is over catch, we are going to target that, to detect the blink of an eye. instead of height and width, we will find the Euclidean distance. one cloud as why Euclidean distance, it because when we tilt our head a bit, the width and height of eyes will change accordingly in order to avoid that, must find the distance, form one point the other point first of All, we will select the landmarks, from

Eyes landmarks, let me show you in the image, First

below image, show to two lines draw on eyes using landmarks, indicate the distance between two landmarks, horizontally and vertically, when eyes are open, the vertical distance reaches it maximum values, while horizontal remain constant, on the other hand, Eyes are closed then Vertical distance approaches to it minimum values

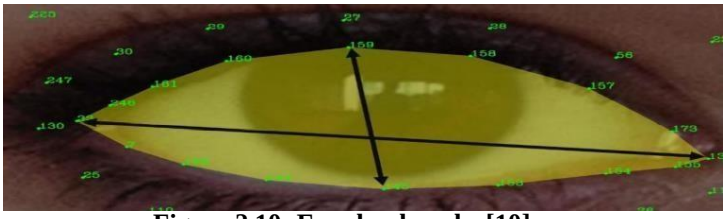


Figure 2.10: Eyes landmarks [10]

2.1.5. Approaches in Eye tracking (WebGazer)

WebGazer, a new approach to browser-based eye tracking for common webcams. WebGazer aims to overcome the accuracy problems that webcams typically face, by adopting user interactions to continuously self-calibrate during regular web browsing. Webcam images during these user interactions can be collected and used as cues for what the user‘s eyes look like. Future observations of the eyes can be matched to similar past instances as WebGazer collects mappings of eye features to on-screen gaze locations, allowing inference of the eye-gaze locations even when not interacting. The base WebGazer technology is compatible with three open-source eye detection libraries for locating the bounding box of the user‘s eyes. The eye detectors that are evaluated in this work are clmtrackr, is-object detect, and tracking.js, but it can be generalized to include others. There are two gaze estimation methods in WebGazer, one which detects the pupil and uses its location to linearly estimate a gaze coordinate on the screen, and a second which treats the eye as a multi-dimensional feature vector and uses regularized linear regression combined with user interactions. WebGazer goes beyond using only clicks as user interaction data, to also applying the cursor movements and the eye gaze-cursor coordination delay as modifiers to the basic gaze estimation model. This is where understanding user behavioral habits is helpful in constructing the model. Which this library we need to do the same as it and develop it more accuracy and make it for writing on keyboard not just cursor like a mouse.

2.2. System Architecture

The proposed system consists of very simple components. A webcam used for face image acquisition. Face and Eye detections were performed, then calculating the gaze ratio for detecting the eye direction.

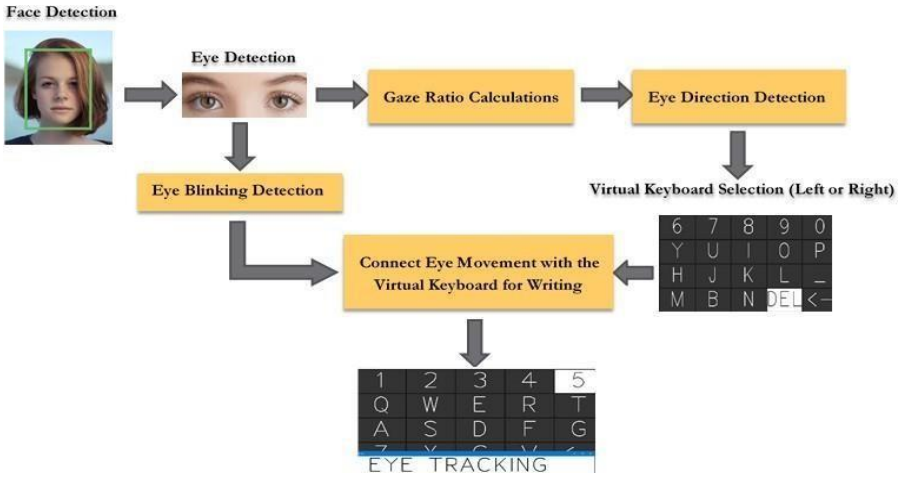


Figure 2.12: Proposed system architecture [10]

Two virtual Keyboard are designed contains all letters and numbers which help users to write letter or number in less than one second and this is a good contribution in the proposed system. One Virtual keyboard was selected depends on the eye direction. Selecting the wanted letter or number depends on detecting the used eye blinking as shown figure 2.12. The user can also choose one of the suggested words to make the writing fast The mouse cursor moves depend on head pose estimation. The selected object will be opened when system detect the blink.

2.2.1. Results of Experiment

Performing eye detection after detecting the face from the captured image. The libraries OpenCV, NumPy, and the dlib are used to detect 68 specific facial landmarks points as shown in figure (2.4). The used algorithms determine the location of the face and the eye according to these landmarks. There is a specific index assigned to each point. Points of the left eye are (36, 37, 38, 39, 40, 41) and of the right eye are (42, 43, 44,45, 46, 47). Eye gaze detection: The main reason for eye-gaze detection is determining the direction of the eye (left right center). It is known that the eye is divided into two parts sclera (white part of the eye) and pupil.

By looking at the image in figure (2.3), it's that the sclera fills the right part of the eye when the eye is looking at the left, The reverse occurs when looking to the right and the white is well balanced between left and right when it looks to the center. Therefore, we focus to detect the white part(sclera). The idea is to split the eye into two parts and to find out which of the two parts there is more sclera visible. If the sclera is more visible on the right part, so the eye is looking at the left and versa. Technically to detect the

sclera they will be converted into grayscale, then finding a threshold value of eye (the value which Separates between sclera and pupil) and counting the white pixels. then get the ratio between left side white pixels and right-side white pixels. This ratio represents where the specific eye is looking. Normally both the eyes look in the same direction, so if we correctly detect the gaze of a single eye, we detect the gaze of both eyes.

Calculation of gaze ratio:
The function divides the eye into two parts, right and left sides. The gaze ratio was calculated by counting the white pixels on each side as the following equation:

$$\text{Gaze Ratio} = \frac{\text{Left Side White Pixels}}{\text{Right Side White Pixel}}$$

The Gaze Ratio was calculated for the right and left eyes, and then the final gaze ratio was calculated for both eyes together by thisequation:

$$\text{Gaze Ratio for both eye} = \frac{\text{Right Eye Gaze Ratio} + \text{Left Eye Gaze Ratio}}{2}$$

determine the direction of the eye using the proposed system, all the volunteers were placed in front of the webcam and move their eyes left, right, and in the center.

It observed that when the eye direction is to the right, the gaze ratio value is less than 1 and when the eye direction is left, the gaze ratio value is greater than 1.7 and when the eye direction is center, the gaze ratio value is in between 1 and 1.7. as shown in figure 2.3.

Virtual Keyboard: Virtual keyboard was designed to display the Keys on the screen and light them up one at a time. Once the key wanted to press is lighted up, we simply would need to close our eyes then the key will be pressed.

Creating a Virtual Keyboard

An empty black window is created and called (keyboard) with the size of 1850 pixels of width and 800 pixels of height thenthe keys are added. Each key is just a rectangle that contains some text. Therefore, the Rectangles are drawn with the number of letters inside the black window. Then the letters are defined inside the rectangles by function contain three parameters the letter index, the letter, and if light up or not in the key. The keyboard is split into two parts (left and right), every part ofthe keyboard can be accessed by looking in a specific direction.
For example, if all the letters that are on the left sides wanted to be accessed, just look to the left and the left keyboard will be activated. As shown in figure 2.13 one of two keyboards contain only half of the letters and numbers and considered as left side keyboard and the second keyboard contains all the remaining letters and numbers which considered right side keyboard. The using of two keyboards make the system easier to use and take a shorter time for writing than using one keyboard. The button (–<- –) means back to main keyboard and the button (–_‖)means space between letters while button (DEL) means delete the letter. The following figure shows the main screen, which

is divided into two parts, left and right. When looking to the right, the right keyboard) is opened while when looking to the left, the left keyboard is opened

Figure 2.13: left and right virtual keyboards [13]

.

Figure 2.14: The main keyboard [13]

Eye blinking means that the eye is closed, the eye is blinking when:

- The eyelid is closed.
- Eyeball can't be seen.
- Bottom and upper eyelashes connect together.

To calculate the blinking value, two vertical lines are drawn at each eye as shown in figure 2.5

The length of the horizontal and vertical lines was used to calculate the blinking ratio as in the following equation:

$$\text{Blinking Ratio} = \frac{\text{Horizontal line length}}{\text{Vertical line length}}$$

The Blinking Ratio was calculated for right and left eyes to get the final Blinking Ratio by this equation:

$$\text{Blinking Ratio for both eyes} = \frac{\text{left blinking ratio + right blinking ratio2}}{}$$

Sounds are added to the proposed system. This sound to tell the user when the letter was pressed because when the user is blinking to press the letter, he can't look at the screen at the same time. besides, two other sounds are added one to tell the user when the left keyboard is activated and one to tell when the right one is activated. The library that uses to load three sounds iscalled piglet

Writing in the output window

As explained before, the gaze ratio used to determine the direction of eye movement for selecting the virtual keyboard. These values were appropriate for all volunteers. In The gaze control keyboard, the person waits until the specific key lights up and then close the eyes for less than one second. then the letter appears on the white window –Board‖ .as shown in figure 2.13.

2.3. User of the system

- User 1: People with dyskinesia CP have problems controlling the movement of their hands, arms, feet, and legs, making it difficult to sit and walk. The movements are uncontrollable and can be slow and writhing or rapid and jerky. Sometimes the face and tongue are affected, and the person has a hard time sucking, swallowing, and talking. A person with dyskinetic CP has muscle tone that can change (varying from too tight to too loose) not only from day to day, but even during a single day.
- User 2: People with Parkinson‘s disease is a brain disorder that leads to shaking, stiffness, and difficulty with walking, balance, and coordination. Parkinson's symptoms usually begin gradually and get worse over time. As the disease progresses, people may have difficulty walking and talking. They may also have mental and behavioral changes, sleep problems, depression, memory difficulties.
- User 3: People can use the system for entertainment experience.

2.4. System Constraint, Cost & Feasibility

In this section, we discussed the cost of implementing the system, the obstacles that will face us, and the feasibility of implementing it.

2.4.1. System Constraint

Constraints imposed on the OTS system was taken into consideration before starting building the systemare:

The system needs a good knowledge of patients who are unable to write using their hands, so a careful study of the eye andthe pupil inside it and the movement of the eye right and left and in the middle is necessary.

2.4.2. System cost

Also, among these constraints are costs. The budget for the resources and quantities must be determined. Inthe following Table 2.1, an estimate of the costs of the system for its implementation is defined.

Camera (if used)	**3000 $ for one time**
System developer	**2000 $ Salary/month**
Maintenance	**8000 $/year**

Table 2.1: System Constraint, Cost & Feasibility

2.4.3. System Feasibility

The feasibility study is performed to determine whether the proposed system is viable; it focuses on

- meet user requirements.
- Building a cost-effective system.
- Creating a technically feasible system.

It is further classified into three aspects:

- Economic Feasibility: building the system does not need a lot of costs because it is a technical system, but it needs to maintain and develop the system. Its benefit to manage disabled people to write on screen using their Eyes.
- Technical Feasibility: The system requires some technical resources such as. The system should be moreaccurate, secure, and reliable.
- Operational feasibility: The proposed system is very easy to use, and the system is easily to understood by simple training and can be operated by users .

2.5. System Requirements

In this section, the Functional and Non-Functional Requirements of the OTS system are defined.

2.5.1. Functional and Data Requirements

Based on user needs, a list of the basic requirements needed by the system is provided in Table 2.2.

Requirements	Description
Interaction:	- Facilitation of human-computer interaction and machine learning.
Communication :	- provides means of communication especially for people with severe disabilities
Writing	- users can use the system keyboard to write
Select	- User can use the mouse to select objects.
Word suggestion	- User can use one of the suggested words.

Table 2.2: Functional Requirements

2.5.2. Non-Functional Requirements

Non-functional requirements are more descriptive than functional for system problems, and are primarilyconcerned with quality as shown in table 2.3

Usability:	• **The user interface of the application should be elegant and user-friendly.** • **The system should include a help list.**
Performance:	• **It allows the user to write easily.** • **The response time depends on the motion of the eye and eye blinking.**
Security:	• **the system doesn't record or preserve users" faces** • **Secure access to the data.**
Efficiency:	• **The system must support many users actively.**
Accessibility:	• **Allow users to access who have the App.**
Portability:	• **The system works with any system :(Mac, Windows, Linux).**
Maintainability:	• **The system must maintain every year**

Table 2.3: Non-Functional Requirements

3

System Modeling & Design

Chapter Summary:
This chapter presents the project graphs and the initial graphical user interface for the system.

3.1. Use Case Analysis

Use case analysis is a procedure used to recognize the prerequisites of a framework, and the data used to both characterize forms utilized and classes represented by a collection of actors and processes. Which will be used in the use case diagram in the improvement or overhaul of a product framework or pro-gram.

3.2. Process Modeling

Process modeling is a strategy intended to comprehend and portray the procedure. It interfaces and improvesthe correspondence between the current and the future condition of a process. A defined order of tasks or activities, with a start, an end, and plainly described inputs and outputs.

3.2.1. Behavioral Diagrams

Behavioral Diagrams delineate the components of a system that are reliant on schedule and that pass on thedynamic conceptions of the system and how they identify with one another.

3.2.1.1. Use Case Diagram

Use Case used to present the core function of the system.In our System, the core functions are:

- User can open the app.
- User can use mouse
- User can write on visual Keyboard.
- User can Close the app.

Open App

In this function, the user opens the app, and the system tries to detect his eyes to begin writing. If thesystem failed in detection, it must notify the user.

Write on visual keyboard

When the user writes, he can choose one of the suggestion words. After the user write what he wants, he can send the text or save the text.

Close the app

After the user use the app, he can close the app.

Use the mouse

User can use the mouse by the movement of his eyes and head position. The selected object will be opened when the system detects the blink.

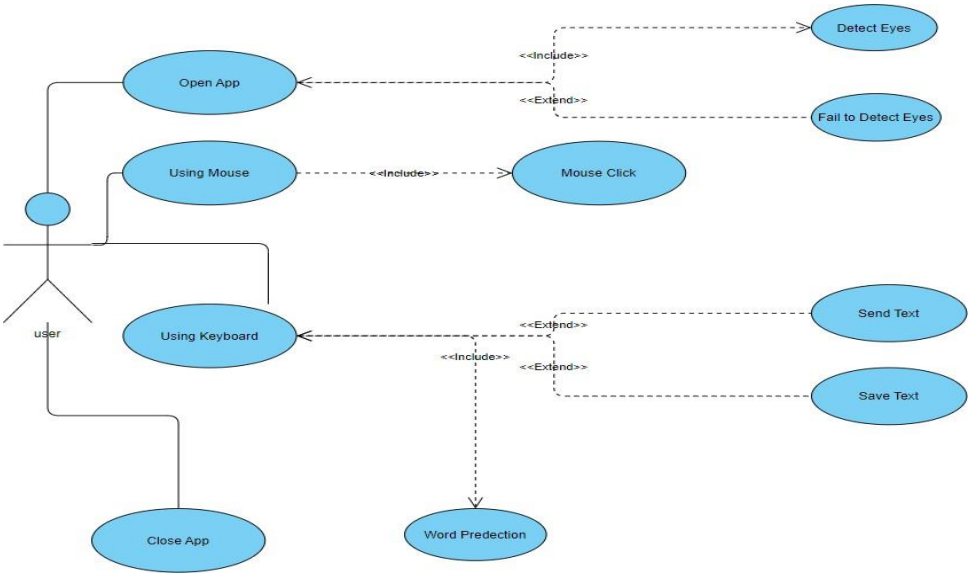


Figure 3.1: Use case diagram

3.2.1.2. Activity Diagram

This activity diagram (Figure 3.2) shows the main actions that have happened in our system when using Mediapipe, and how these actions are related in an overall flow of the system.
All the information and requests also show the system steps from webcam to do the system function.

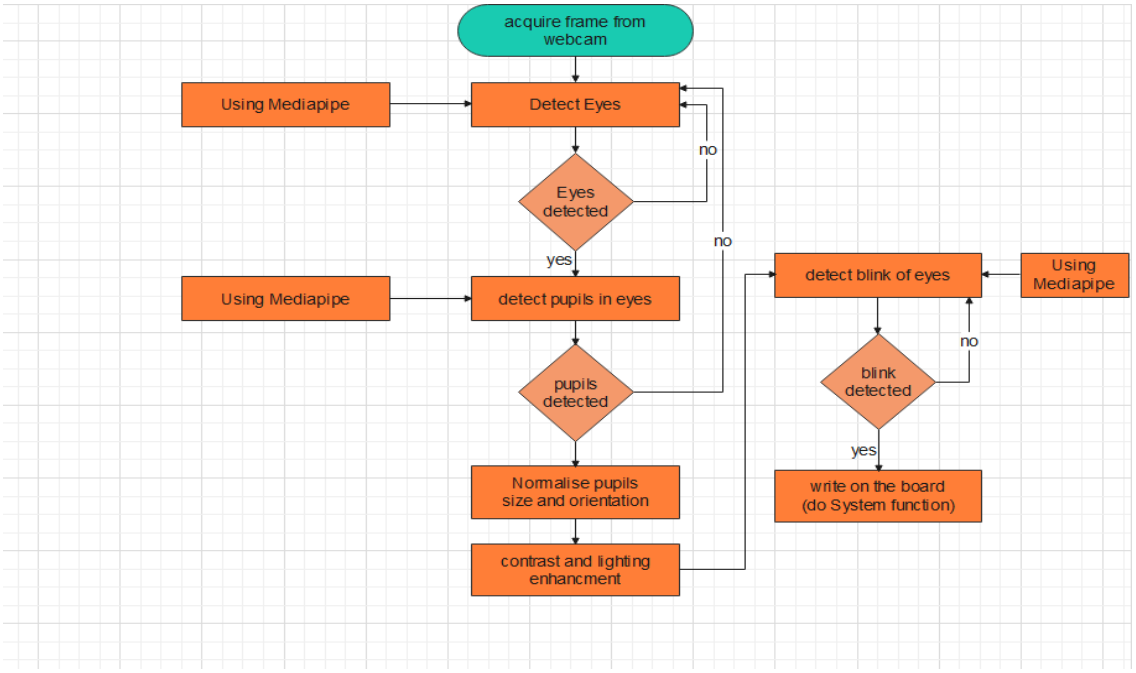


Figure 3.2: Activity Diagram

3.2.2. Sequence Diagram

Sequence diagram, model the communications between objects in a once use case. They outline how the various pieces of a framework communicate with one another to do functions and the request where the cooperation is happen when a specific use case is executed. In Figure 3.3 First, the patient uses the web cam associated with our system, and then the preprocessing takes place to learn more details about the frame, and then the system determines the pupil of the patient by comparing the pupil of the patient with what is in the system database, and then if these operations are completed successfully, the patient can perform the function of the system It is writing through his eyes or moving mouse cursor and select objects.

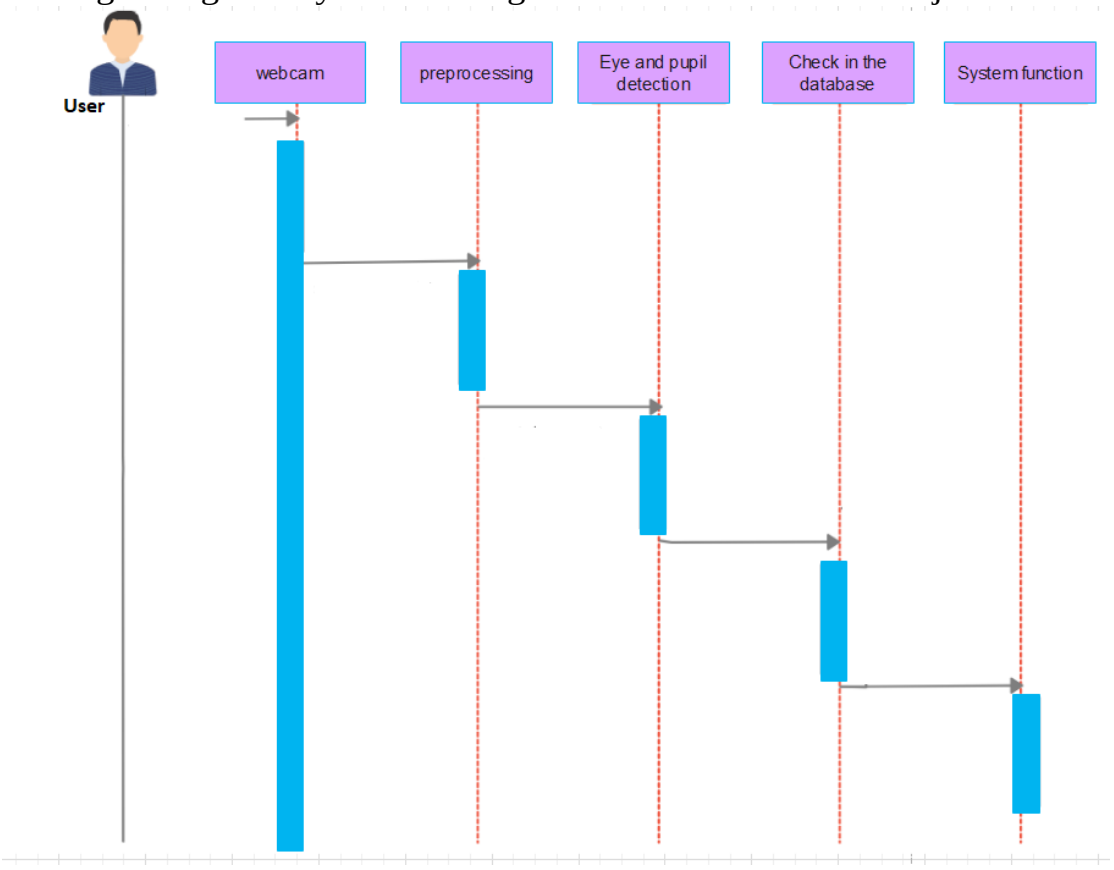


Figure 3.3: Sequence Diagram

3.3. Data Modeling

Data modeling is the way toward making a data model for the data to be put away in a database; it is acon-ceptual representation of Data objects, the relationship between various data objects, and the rules.

3.3.1. Entity Relationship Diagram

First, the patient should tell us if he/she has any pathological case in his/her eye like Cataract, Corneal Opacity, Binocular Vision, or Monocular Vision, and error of refraction, to deal with this is diagnosis. The second step, get the id of the patient, age, and image that shows his/her face and the direction of his eyes. In the third step, we fetch his/her image to dataset/classifier Fourth step; we create a model to detect the face of the patient. In the fifth step, we extract the feature of his/her face, but we will focus on his Eye, Gaza, and Pupil. In the sixth step, when we extract the feature of eye, gaze & pupil then we classify it based on the dataset according to the location of the eye on the screen and its direction. In the seventh step, we map our information (location, direction) on the screen or on the keyboard to know where the patient is looking for. Eighth step, the letter that we predict that the patient is looking for it; it will show on the board.

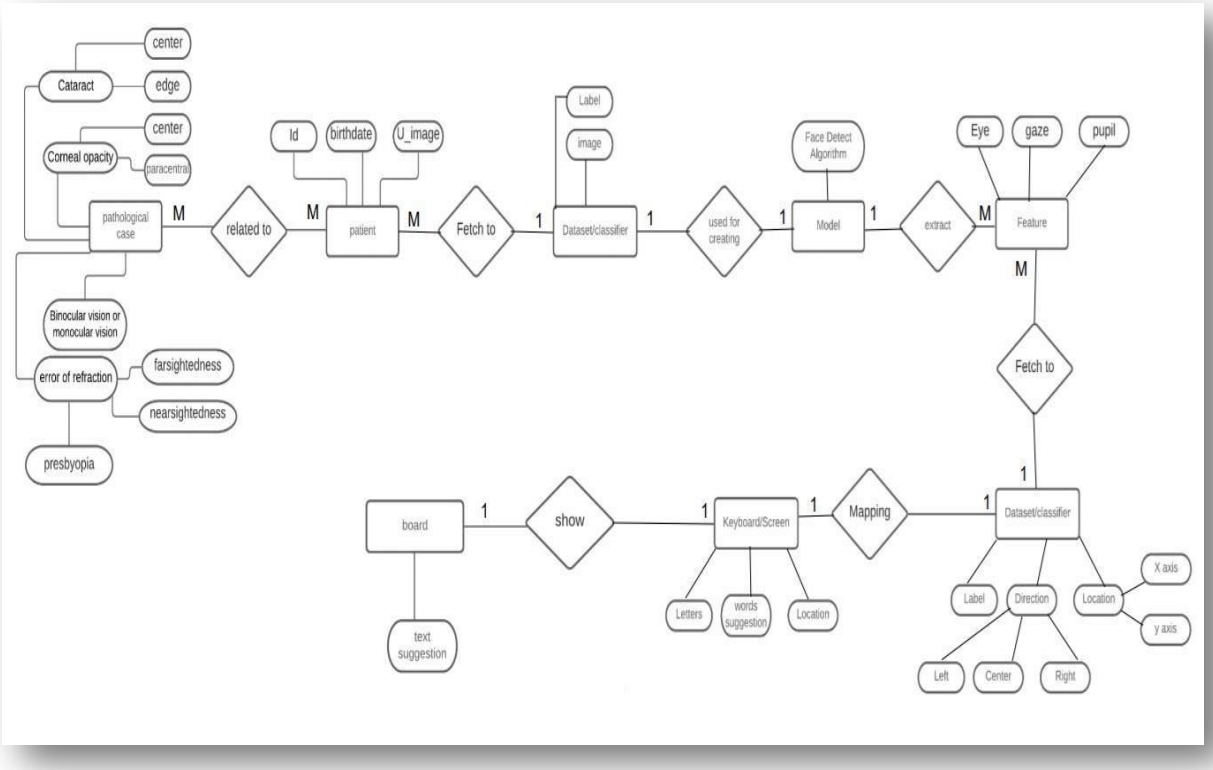


Figure 3.5: Entity Relationship Diagram

3.3.2. Data flow diagram level 0

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

DFD describe our system, Once the patient enters the system his/her webcam camera start live video. from each frame coming from live video, we can detect the patients face by Mediapipe algorithm. If this process fails, we ask the patient to set in front of the camera.

Else, detecting the head pose estimation of patient by using also Mediapipe algorithm. Then extract the facial landmarks such as eyes,we can be easy to detect the eye blinking

In case the process cannot detecting the head movement, the patient still in this process until detecting his/her head movement. Else, the patient should choose from two options by using his/her head movement. The first option is the process of expressing his/her feelings, the patient chooses from feelings database. The second option is the process of writing in keyboard using her/his pupil, patient choose from left.

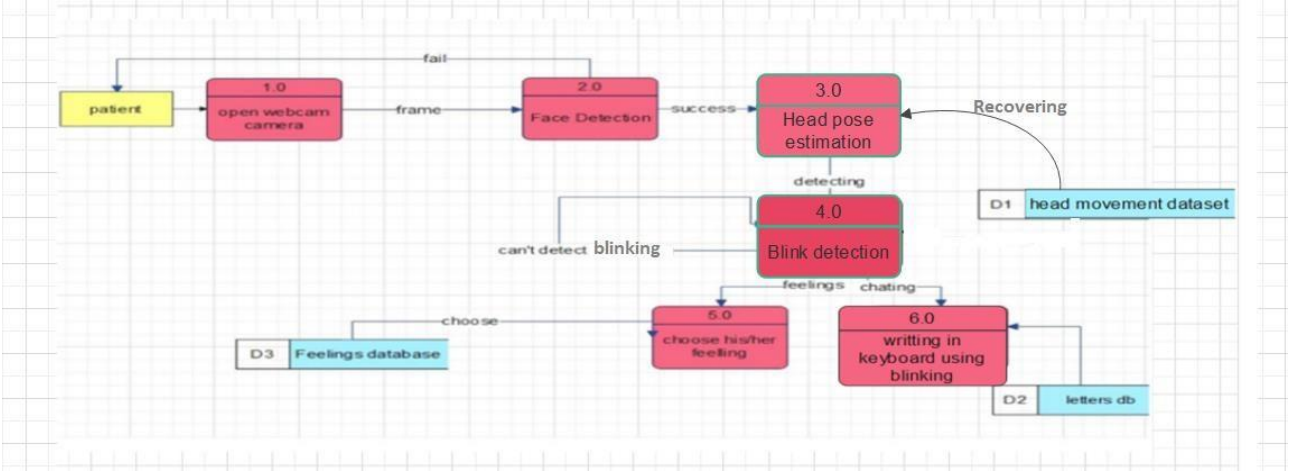


Figure 3.6: data flow diagram (DFD) level 0

3.3.3. Data flow diagram (Context Diagram) level 1

Context diagrams show the interactions between a system and other actors (external factors) with which the system is designed to interface. The patient is the only external factors in our system. The interactions between the patient and the Head pose estimation system include face recognition system which it detect patient‘s face, Eye recognition system which it detect their eyes, detecting head movement of the patient, recording the head movement, check if there is error of detecting or not, The patient can chatting and expressing of his/her feelings .

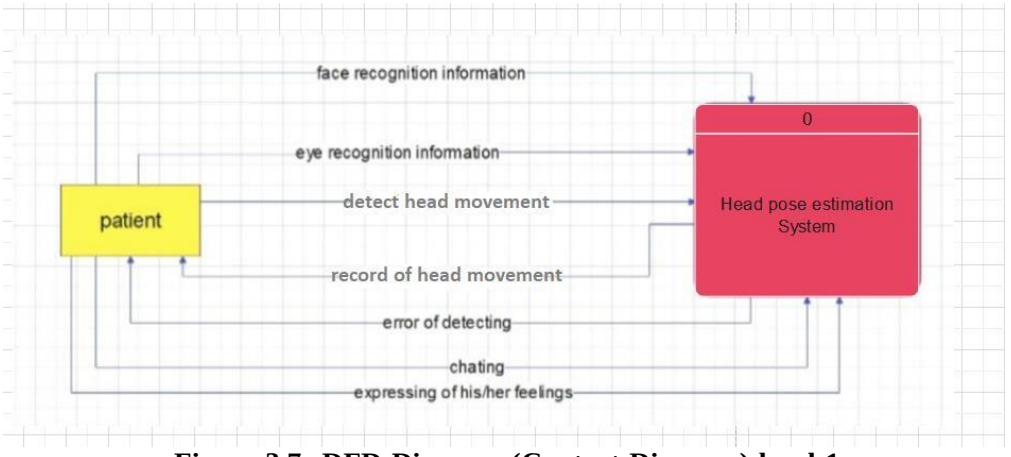


Figure 3.7: DFD Diagram (Context Diagram) level 1

3.4. User Interface Design

The user interface (UI) is the point of human-computer interaction and communication in a device, oursystem contains the following interfaces:

How does it work?

Actually we have two versions will describe each of them The first version (version 1)

Once we run the program, we see window containing respectively: the keyboard and awhite board.

The keyboard contains the right side of the real keyboard which has the letters: Y, U, O, P, H, J, K, L, V, B, N, M, spacebar and I.

- **Press key:** Let‘s supposed that we choose the Right side.

Now we will see the keys contained on the right side. To press a key we need to look at the specific key and wait that the key lights up, and then we blink our eyes. We keep them closed until we hear a sound. That sound means that the key has been pressed.

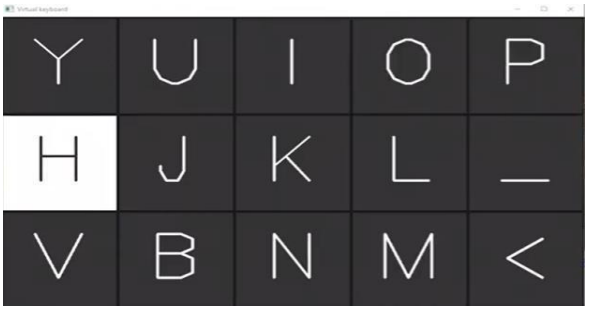


Figure 3.7: Right keyboard

- **App Keyboard:** This interface 3.8contains the key which user choose to be written

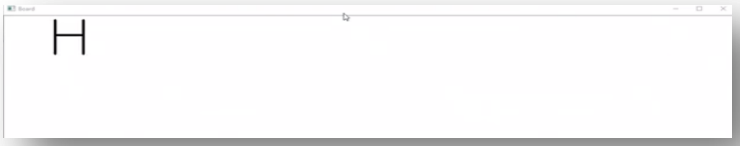


Figure 3.8: App Keyboard

The second version (version 2)

This version consists of two desktop applications, once we run the program, we see two windows one of them to take input from the user to start the system of our project (as shown in figure 3.9) by tracking the webcam user‘s head pose estimation and enables him to control the mouse with the movement of his head.

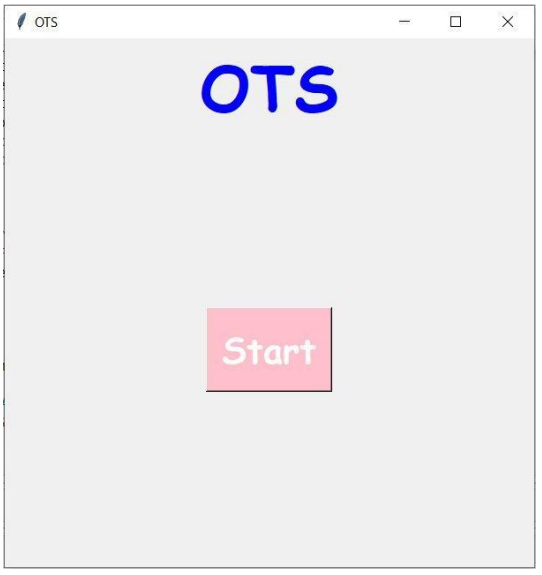


Figure 3.9: App mouse control

The second window is our responsive virtual keyboard which feed by word detection and word prediction to make the writing more easy to the user In the following figure we shown the word detection when user write the “sta” the system gives him all the words that contains the –sta‖ inside it

Figure 3.10: Keyboard(auto complete)

The following figure we shown the word prediction so after the user write for example “states” the system suggest to him the predictive words which come after states

Figure 3.11:word prediction

4

System Implementation

Chapter Summary: In this chapter, we discuss the project implementation, the environment we used, the problems we faced, and display the written codes for each interface

4.1. Introduction

In this section discusses the design and implementation of different Versions of the Eye Tracking System.

4.2. System Development Languages and Environments

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. To implement machine learning concepts and algorithms, we found that python is the easiest language to understand that field for us.

4.3. System Programming

Our project divides into many versions each version is an improvement over the previous one according to the principle of software engineering, to achieve this principle we first focus on the two types of software processes

1- Plan-driven processes are processes where all the process activities are planned in advance and progress is measured against this plan.
2- Agile processes, planning is incremental, and it is easier to change the process to reflect changing customer requirements.

So, when we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities, after that we will focus on software models to choose model that we can apply it in our project, the models we will analysis it are Waterfall model, Reuse Oriented model, Iterative model

- Waterfall model

  The waterfall model is an example of a plan-driven process, you plan and schedule all of the process activities before starting software development

- Reuse Oriented model

  It can be steps of the software development for specific duration in which software is redesigned through creating a sequence of prototypes

known as models, every system is derived from the previous one with constant series of defined rules.

- Iterative model

The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental), allowing software developers to take advantage of what was learned during development of earlier parts or versions of the system.

After our analysis to these models, we found the iterative development model is the best model enables us to produce many versions.

4.3.1. Version #1

In this version, we use CVzone, which is a computer vision package that makes us easy to run like face detection, hand tracking, pose estimation, etc., and image processing and other AI functions. At the core, it uses OpenCV and MediaPipe libraries.
Our version one is a half-keyboard that swaps between letters at a specific time, and when the user wants to type a letter, he makes a blink, so we want to how we can detect the blinking of eyes using CVzone.
First, we detect the face as we show in figure 4.2 below

```
import cv2
import cvzone
from cvzone.FaceMeshModule import FaceMeshDetector


cap=cv2.VideoCapture('Video.mp4')
detector = FaceMeshDetector(maxFaces = 1)

while True:


    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_POS_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES,0)

    success , img = cap.read()
    img , faces = detector.findFaceMesh(img)

    img = cv2.resize(img , (640,360))
    cv2.imshow("image",img)
    cv2.waitKey(1)
```

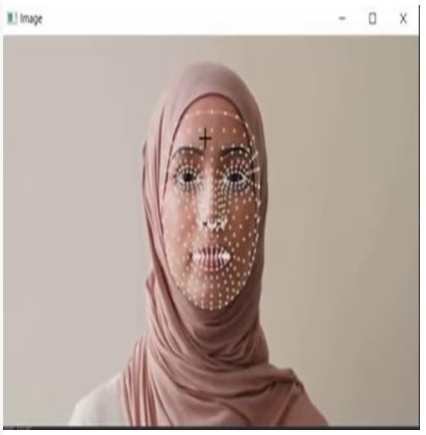


Figure 4.2 face mesh detecting

Second, we now want to reach to what were the points around the eyes.
We found that the points around right eye are
(22,23,24,26,110,157,158,159,160,161,130,243)
so we can detect the right eye as shown in figure 4.3

```
idlist = [22,33,24,26,110,157,158,159,160,161,130,243]

while True:

    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_POS_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES,0)

    success , img = cap.read()
    img , faces = detector.findFaceMesh(img, draw = False)

    if faces:
        face = faces[0]
        for id in idlist:
            cv2.circle(img , face[id], 5 , (255,0,255), cv2.FILLED)
```

Figure 4.3 right eye detecting

At last we now want to detect the blinking of right eye, we take the point on the top(159) and the point on the down(23) then take the distance between them (vertical distance) repeat this process with point on the right(243) and the point on the left(130) and take the distance between them(horizontal distance) after that calculate the ratio between vertical and horizontal distance as shown in the figure 4.4 below

```
if faces:
    face=faces[0]
    for point in point_list:
        cv2.circle(img,face[point],3,(255,0,255),cv2.FILLED)
    lefteye_up=face[159]
    lefteye_down=face[23]
    lefteye_left=face[130]
    lefteye_right=face[243]

    length_vertical,_=detector.findDistance(lefteye_up,lefteye_down)
    length_horizontal,_=detector.findDistance( lefteye_left,lefteye_right)
    cv2.line(img,lefteye_left,lefteye_right,(0,200,0),2)
    cv2.line(img,lefteye_up,lefteye_down,(0,200,0),2)
    ratio=int((length_vertical/length_horizontal)*100)
    font=cv2.FONT_HERSHEY_SIMPLEX
    if ratio<31:
        cv2.putText(img,"Blinking",(50,150),font,3,(255,0,0))
```

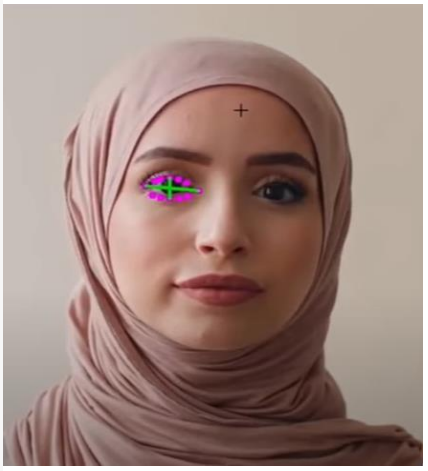


Figure 4.4 detect blinking of right eye

After we have detected the blinking of the right eye we can describe and display the version

Scenario of that version is when the user run the program, he will see **two different windows** containing respectively: **the first window contains half-keyboard and white board and the second window contains capture from the webcam**.

The keyboard is divided in two parts

- left side contains the letters: Q, W, E, R, T, A, S, D, F, G, Z, X, C, V.
- right side contains the letters: Y, U, I, O, P, H, J, K, L, V, N, M

In this version, we build the only left side as we display in figure 4.5

Figure 4.5: Version#1

4.3.2. Version #2

In this version we enable the user to control the mouse using head movement and while the user moving the mouse can use his eye‘s blinking to do mouse functions such as click for example when user need to write a letter he can use our responsive virtual keyboard ,move the mouse using his head then place the mouse on the specific letter and blink his eyes for 10 frames

We publish two releases from this version will describe each of them later ,but before talking about them ,the two releases share the same code of eye blinking so we first will talk about it.

At the first we detect points on eyes that make horizontal and vertical line in the center of the eye.

```
if faces:
    face = faces[0]
    leftUp = face[159]
    leftDown = face[23]
    leftLeft = face[130]
    leftRight = face[243]
    lenghtVer, _ = detector1.findDistance(leftUp, leftDown)
    lenghtHor, _ = detector1.findDistance(leftLeft, leftRight)
```

Figure 4.6:lines

And make two lines connected with each other
as figure 4.6 shown.

And then calculate the ratio between vertical and horizontal length to find best performance about the eye blink. And then calculate the average of number in ratio list in each 3 frames. (as shown in figure 4.7)

```
ratio = int((lenghtVer / lenghtHor) * 100)
ratioList.append(ratio)
if len(ratioList) > 3:
    ratioList.pop(0)
ratioAvg = sum(ratioList) / len(ratioList)
```

Figure 4.7 (calculate the ratio)

In the figure 4.8 we detect the blink when the ratio become less than 27 (in 10 frames because eye still close in many frames in one blink) and make a double click of the mouse to click in any button on widows & make sound like beep.

```
if ratioAvg < 27 and counter == 0:
    blinkCounter += 1
    m.click(clicks=2)
    winsound.Beep(500, 250)
    counter = 1
if counter != 0:
    counter += 1
    if counter > 10:
        counter = 0
        color = (255,0, 255)
```

Figure 4.8(blink)

4.3.2.1 Using Mediapipe

OpenCV solvePnP

In OpenCV the function solvePnP and solvePnPRansac can be used to estimate pose.
solvePnP implements several algorithms for pose estimation which can be selected using the parameter flag. By default it uses the flag SOLVEPNP_ITERATIVE which is essentially the DLT solution followed by Levenberg-Marquardt optimization. SOLVEPNP_P3P uses

only 3 points for calculating the pose and it should be used only when using solvePnPRansac.

OpenCV solvePnP Function

OpenCV library contains two main functions for pose estimation. The two methods are solvePnP() and solvePnPRansac(). Here, we will be looking into the function solvePnP().

The cv2.solvePnP() method is generally used in pose estimation, or in other words, it can be used to estimate the orientation of a 3D object in a 2D image. So for this you need tag some key-points in the 3D model of the object (object Points) and also detect those key-points in the 2D image (image Points).

Code

```
In [1]: import cv2
        import mediapipe as mp
        import numpy as np
        import pyautogui as pyag
        from cvzone.FaceMeshModule import FaceMeshDetector
```

Figure 4.9(libraries)

Using **mediapipe** we can detect face mesh and from there we detect the eye and **pyautogu**i control the mouse and keyboard to automate interactions with other applications.

```
In [2]: pyag.PAUSE = 0.01
        pyag.FAILSAFE = False
        drag = 20
```

Figure 4.10(make mouse smooth)

Setup parameters to control speed of mouse by make it 0.01 second and if the mouse is in any of the four corners of the primary monitor, they will raise a pyautogui. FailSafeException when make it FALSE it prevent error to occur and using parameter drag that represent distance of mouse that can move from one location to another.

```
In [3]: mp_face_mesh = mp.solutions.face_mesh

        face_mesh = mp_face_mesh.FaceMesh(min_detection_confidence=0.5, min_tracking_confidence=0.5)

        mp_drawing = mp.solutions.drawing_utils

        drawing_spec = mp_drawing.DrawingSpec(thickness=1, circle_radius=1)

        detector1 = FaceMeshDetector(maxFaces=1)
```

Figure 4.11(detect face)

Drawing landmarks using mediapipe with minimum detection confidence 50% and by using cvzone make software detect just one face.

```
if results.multi_face_landmarks:
    for face_landmarks in results.multi_face_landmarks:
        for idx, lm in enumerate(face_landmarks.landmark):
            if idx == 33 or idx == 263 or idx == 1 or idx == 61 or idx == 291 or idx == 199:
                if idx == 1:
                    nose_2d = (lm.x * img_w, lm.y * img_h)
                    nose_3d = (lm.x * img_w, lm.y * img_h, lm.z * 3000)

                x, y = int(lm.x * img_w), int(lm.y * img_h)
```

Figure 4.12(extract nose landmark)

Detect 6 points to know head pose estimation

```
                # Get the 2D Coordinates
                face_2d.append([x, y])

                # Get the 3D Coordinates
                face_3d.append([x, y, lm.z])

        # Convert it to the NumPy array
        face_2d = np.array(face_2d, dtype=np.float64)

        # Convert it to the NumPy array
        face_3d = np.array(face_3d, dtype=np.float64)
```

Figure 4.13(detect 2D &3D Face)

using image based approach we need some 3d reference point that we are mapping down to the image plane and then we are trying to find the poles or rotation and translation from theses 3d points down to image plane of our 2d points so it will be corresponding points in 3d predicated down into the image plane where we have our 2d facial landmark

```
# The camera matrix
focal_length = 1 * img_w

cam_matrix = np.array([ [focal_length, 0, img_h / 2],
                        [0, focal_length, img_w / 2],
                        [0, 0, 1]])

# The distortion parameters
dist_matrix = np.zeros((4, 1), dtype=np.float64)

# Solve PnP
success, rot_vec, trans_vec = cv2.solvePnP(face_3d, face_2d, cam_matrix, dist_matrix)

# Get rotational matrix
rmat, jac = cv2.Rodrigues(rot_vec)
```

Figure 4.13(solvepnp)

The syntax for solvePnP() function is

cv2.solvePnP(objectPoints, imagePoints, cameraMatrix, distCoeffs, rvec, tvec, useExtrinsicGuess, flags) → retval, rvec, tvec

Parameters of solvePnp() Function in Python:

- **objectPoints:** It is an array of object points. objectPoints can either be a matrix or a vector of N 3D points.
- **imagePoints:** It is an array of image points. imagePoints can either be a matrix or a vector of N 2D points.
- **cameraMatrix:** It is a 3 by 3 camera matrix taken as an input.
- **distCoeffs:** It is a vector that consists of distortion coefficients. Unless the distortion is huge in the camera being used, we can assign NULL value to this.
- **rvec:** It is an output rotation vector
- **tvec:** It is an output translation vector
- **useExtrinsicGuess:** If the value of useExtrinsicGuess is set to 0, it will use rvec and tvec values for initial calculations as rotation and translation vectors.
- flag: It specifies the method for solving the PnP problem.

```
# Get angles
angles, mtxR, mtxQ, Qx, Qy, Qz = cv2.RQDecomp3x3(rmat)

# Get the y rotation degree
x = angles[0] * 360
y = angles[1] * 360
z = angles[2] * 360


# See where the user's head tilting
if y < -10:
    pyag.moveRel(-drag, 0)
    text = "Looking Left"
elif y > 10:
    pyag.moveRel(drag, 0)
    text = "Looking Right"
elif x < -10:
    pyag.moveRel(0,drag)
    text = "Looking Down"
elif x > 10:
    pyag.moveRel(0,-drag)
    text = "Looking Up"
else:
    text = "Forward"
```

Figure 4.12(mouse control)

in this release the control of mouse is vertically and horizontally while the head movement and ignore the angles between the two axes so the system work when the user‘s head move on the two axes only.

4.3.2.2 Using Dlib

Before we talking about this release first, we 'll analysis face pose estimation using Dlib so we have reached that the face of a person is a 3D object, it can rotate over all three axes — but with some limitations, of course. In a face pose estimation problem, we call these movements as roll, pitch, and yaw, better visualized in the figure below:

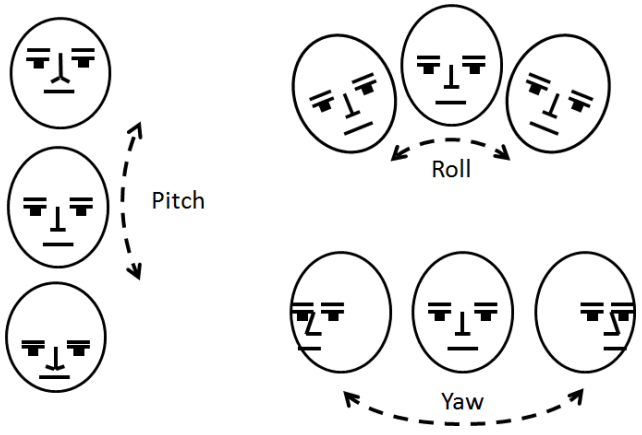


Figure 4.13(faces different angles)

Estimating these poses is useful for liveness detection systems. For instance, it may ask a user to perform some predefined random movements (e.g., –rotate your face to the right‖) to check his liveness.

The Dataset

For this problem, I created a toy database with 6,288 images from different face datasets. For each image, I detected the facial landmarks with Dlib (68 points) and computed the pairwise Euclidean distance between all points. Thus, given 68 points, we ended up with (68*67)/2 = 2,278 features.

Coding of analysis the direction of head over all three axes:

1. The libraries

```
import cv2
import dlib
import numpy as np
import matplotlib.pyplot as plt
import _pickle as pkl
from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

%matplotlib inline
```

Figure 4.14(libraries)

2. The Load Data

```
x, y = pkl.load(open('data/samples.pkl', 'rb'))

print(x.shape, y.shape)
```

```
(6288, 2278) (6288, 3)
```

Figure 4.15(load data)

x.shape has number of images with number of features, while y. Shape has number of images with 3 position.

```
roll, pitch, yaw = y[:, 0], y[:, 1], y[:, 2]

print(roll.min(), roll.max(), roll.mean(), roll.std())
print(pitch.min(), pitch.max(), pitch.mean(), pitch.std())
print(yaw.min(), yaw.max(), yaw.mean(), yaw.std())
```

```
-46.06486893 43.00866699 -0.52579503197 5.17623111671
-29.88856888 34.09674835 2.58468274027 7.96282815186
-75.55059814 86.84925079 -0.116205880073 13.0908391832
```

Figure 4.16(extract angles)

We put the value of roll, pitch & yaw, then we want to the maximin, minimum, mean & standard deviation for each one.

3. Divide your dataset in train, validation, and test

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.5, random_state=42)

print(x_train.shape, y_train.shape)
print(x_val.shape, y_val.shape)
print(x_test.shape, y_test.shape)
```

```
(4401, 2278) (4401, 3)
(943, 2278) (943, 3)
(944, 2278) (944, 3)
```

Figure 4.17(split dataset)

We split our data into train, test & validation, the test size become 30% from our dataset, then we divided the test dataset into test & validation, then the test size become 50% from its dataset.
We print the size of each dataset.

4. Normalization

```
std = StandardScaler()
std.fit(x_train)
x_train = std.transform(x_train)
x_val = std.transform(x_val)
x_test = std.transform(x_test)
```

Figure 4.18(normalization)

5. Building Our Model & Save it

```
model = Sequential()
model.add(Dense(units=20, activation='relu', kernel_regularizer='l2', input_dim=x.shape[1]))
model.add(Dense(units=10, activation='relu', kernel_regularizer='l2'))
model.add(Dense(units=3, activation='linear'))

print(model.summary())
```

Figure 4.19(building model)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                45580
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 33
=================================================================
Total params: 45,823
Trainable params: 45,823
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.20(results)

```
callback_list = [EarlyStopping(monitor='val_loss', patience=25)]

model.compile(optimizer='adam', loss='mean_squared_error')
hist = model.fit(x=x_train, y=y_train, validation_data=(x_val, y_val), batch_size=BATCH_SIZE, epochs=EPOCHS, callbacks=callback_list)
model.save('models/model.h5')
```

Figure 4.21(training)

```
BATCH_SIZE = 64
EPOCHS = 100
```

the parameter we need to building model

Figure 4.22(model parameters)

```
model.compile(optimizer='adam', loss='mean_squared_error')
hist = model.fit(x=x_train, y=y_train, validation_data=(x_val, y_val), batch_size=BATCH_SIZE, epochs=EPOCHS, callbacks=callback_list)
model.save('models/model.h5')
```

Figure 4.23(training)

```
_________________________________________________________________
None
Train on 4401 samples, validate on 943 samples
Epoch 1/100
4401/4401 [==============================] - 1s 120us/step - loss: 66.7419 - val_loss: 54.9232
Epoch 2/100
4401/4401 [==============================] - 0s 60us/step - loss: 50.4013 - val_loss: 53.6252
Epoch 3/100
4401/4401 [==============================] - 0s 61us/step - loss: 42.6244 - val_loss: 38.4167
Epoch 4/100
4401/4401 [==============================] - 0s 58us/step - loss: 37.2493 - val_loss: 39.3467
Epoch 5/100
4401/4401 [==============================] - 0s 64us/step - loss: 35.2210 - val_loss: 35.9847
Epoch 6/100
4401/4401 [==============================] - 0s 62us/step - loss: 34.9180 - val_loss: 35.2145
Epoch 7/100
4401/4401 [==============================] - 0s 60us/step - loss: 35.1205 - val_loss: 34.9544
Epoch 8/100
4401/4401 [==============================] - 0s 60us/step - loss: 34.7459 - val_loss: 35.8672
Epoch 9/100
4401/4401 [==============================] - 0s 60us/step - loss: 33.8960 - val_loss: 37.7349
Epoch 10/100
4401/4401 [==============================] - 0s 60us/step - loss: 33.8952 - val_loss: 37.3769
Epoch 11/100
4401/4401 [==============================] - 0s 60us/step - loss: 33.4407 - val_loss: 35.2131
Epoch 12/100
4401/4401 [==============================] - 0s 60us/step - loss: 32.8623 - val_loss: 41.7251
Epoch 13/100
4401/4401 [==============================] - 0s 61us/step - loss: 34.6161 - val_loss: 34.7076
Epoch 14/100
4401/4401 [==============================] - 0s 59us/step - loss: 32.8888 - val_loss: 38.1413
Epoch 15/100
4401/4401 [==============================] - 0s 61us/step - loss: 33.0120 - val_loss: 34.9183
Epoch 16/100
4401/4401 [==============================] - 0s 60us/step - loss: 33.2761 - val_loss: 34.9895
```

Figure 4.24(results)

6. The Loss Of Our Model

```
print('Train loss:', model.evaluate(x_train, y_train, verbose=0))
print('  Val loss:', model.evaluate(x_val, y_val, verbose=0))
print(' Test loss:', model.evaluate(x_test, y_test, verbose=0))
```

```
Train loss: 29.2128348724
  Val loss: 33.3887316318
 Test loss: 39.9278703463
```

Figure 4.25(model accuracy)

As we can see, our model achieved a good result even in test set.

7. Testing The Model

Now, we will test the model we've just trained. But first, you must have to download the shape model (if you haven't already) to be able to detect the facial landmarks.

```python
def detect_face_points(image):
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor("models/shape_predictor_68_face_landmarks.dat")
    face_rect = detector(image, 1)
    if len(face_rect) != 1: return []

    dlib_points = predictor(image, face_rect[0])
    face_points = []
    for i in range(68):
        x, y = dlib_points.part(i).x, dlib_points.part(i).y
        face_points.append(np.array([x, y]))
    return face_points

def compute_features(face_points):
    assert (len(face_points) == 68), "len(face_points) must be 68"

    face_points = np.array(face_points)
    features = []
    for i in range(68):
        for j in range(i+1, 68):
            features.append(np.linalg.norm(face_points[i]-face_points[j]))

    return np.array(features).reshape(1, -1)
```

Figure 4.26(detect face)

```python
im = cv2.imread('data/lena.png', cv2.IMREAD_COLOR)
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
face_points = detect_face_points(im)

for x, y in face_points:
    cv2.circle(im, (x, y), 1, (0, 255, 0), -1)

features = compute_features(face_points)
features = std.transform(features)

model = load_model('models/model.h5')
y_pred = model.predict(features)

roll_pred, pitch_pred, yaw_pred = y_pred[0]
print(' Roll: {:.2f}°'.format(roll_pred))
print('Pitch: {:.2f}°'.format(pitch_pred))
print('  Yaw: {:.2f}°'.format(yaw_pred))

plt.figure(figsize=(10, 10))
plt.imshow(im)
```

Figure 4.27(show angles in image)

It 'll print the angle of roll, pitch, and yaw

```
 Roll: 0.67°
Pitch: -4.89°
  Yaw: 22.57°
<matplotlib.image.AxesImage at 0x249345594a8>
```

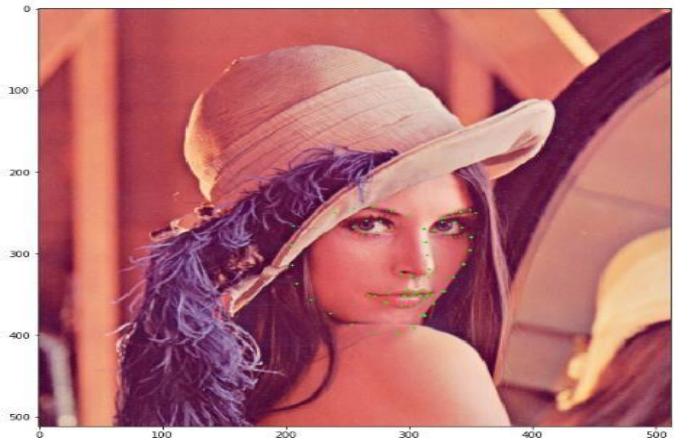


Figure 4.28(image with 3 angles)

Coding of this release

In this version we used face_utils and dlib to extract the user‘s facial landmarks

```
In [ ]: from imutils import face_utils
        import numpy as np
        import dlib
        import cv2
        import pyautogui as m
        import cv2
        import cvzone
        from cvzone.FaceMeshModule import FaceMeshDetector
        from cvzone.PlotModule import LivePlot
        import winsound
```

Figure 4.29(libraries)

This function calculate the view on the screen of the user‘s device

```
In [ ]: def calculateView(x,y):
            xvMax, yvMax = m.size()
            xvMin, yvMin = 0, 0
            xwMax, xwMin = 370, 270
            ywMax, ywMin = 290, 200
            sx = (xvMax - 0) // (xwMax - xwMin)
            sy = (yvMax - 0) // (ywMax - ywMin)
            xv = xvMin + (x - xwMin) * sx
            yv = yvMin + (y - ywMin) * sy
            return xv,yv
```

Figure 4.30(calculate the screen view)

Pre-trained file‖shape_predictor_68_face_landmarks.dat‖ to get the facial landmarks detected

```
In [ ]: PREDICTOR_PATH = "shape_predictor_68_face_landmarks.dat"
```

Figure 4.31(pre trained model)

Here we used dlib based on HO

```
In [ ]: # initialize dlib's face detector (HOG-based) and then create
        detector = dlib.get_frontal_face_detector()
        # the facial landmark predictor
        predictor = dlib.shape_predictor(PREDICTOR_PATH)
```

Figure 4.32(dlib detector)

HOG is a simple and powerful feature descriptor. It is not only used for face detection but also it is widely used for object detection like cars, pets, and fruits. HOG is robust for

object detection because object shape is characterized using the local intensity gradient distribution and edge direction.

extract the indexes of the facial landmarks for the left ,right eye and nose respectively

```
In [ ]: (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
        (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
        (nStart, nEnd) = face_utils.FACIAL_LANDMARKS_IDXS["nose"]
```

Figure 4.33(extract facial landmarks)

Now we start the life video and creating loop to extract each frame convert it to grey scale for faster processing.

```
In [ ]: # start the video stream thread
        vs = cv2.VideoCapture(0)

        # loop over frames from the video stream
        while True:
            # grab the frame from the threaded video file stream, resize
            # it, and convert it to grayscale
            # channels)
            ret, frame = vs.read()
            frame = cv2.flip(frame, 1)
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            # detect faces in the grayscale frame
            rects = detector(gray, 0)
```

Figure 4.34(open live camera)

Inside while we creating loop over the face directions

```
In [ ]:     for rect in rects:
                # determine the facial landmarks for the face region, then
                # convert the facial landmark (x, y)-coordinates to a NumPy
                # array
                shape = predictor(gray, rect)
                shape = face_utils.shape_to_np(shape)
```

Figure 4.35(face predictor)

We extract the nose landmark coordinate as it located in the center of the face to more accuracy when control the mouse using head pose estimation

```
In [ ]: # extract the nose coordinates

                nose = shape[nStart:nEnd]

                # get window coordinates
                xv, yv = nose[0]
                xw = np.int(xv)
                yw = np.int(yv)
                xv,yv = calculateView(xw,yw)
```

Figure 4.36(extract nose coordination)

Then we control the mouse cursor based on the head directions using two numpy arrays to calculate mouse old location and current location in addition to damping factor to make mouse smooth

```
In [ ]:
        mLocOld = np.array([0,0])
        mouseLoc = np.array([0,0])
        DampingFactor = 15          # supposed to make mouse smooth
        # for mouse control
        mouseLoc = mLocOld + ((xv,yv)-mLocOld)//DampingFactor
        #print('nx = {} and ny = {}'.format(mouseLoc[0], mouseLoc[1]))
        m.moveTo(mouseLoc[0],mouseLoc[1],tween =  m.linear(.5))
        mLocOld = mouseLoc
```

Figure 4.37(mouse control)

in this release the system work on three axes when the head is moving

4.3.3. Responsive Virtual Keyboard

Virtual Keyboard using word prediction & auto complete:

First we create a virtual keyboard using buttons and entry to enter the text on it and then save what I write in a file, and there is a list box to see the words are predicting and auto complete while writing (figure **4.38**).

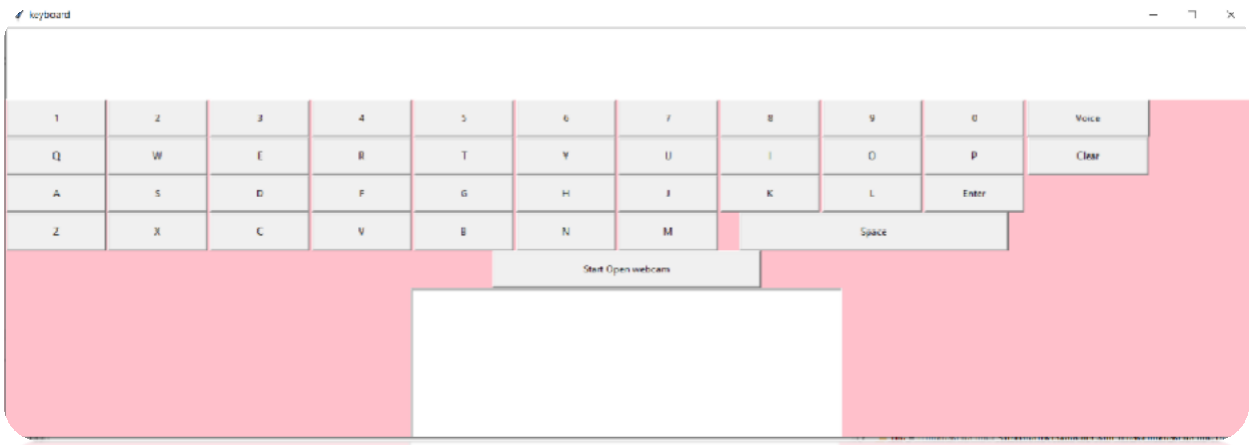


Figure 4.38(Keyboard)

when I press on any button it is call a function called press,
which save the words on text file an write the text on the entry (figure **4.39**).

```
def press(num):
    my_entry.insert(END,str(num))
    s=my_entry.get()
    file = open("voice_file.txt", "w+")
    file.readline()
    file.write(s)
    file.close()
    checkbutton()
```

Figure 4.39(functions)

to make update in the list box to predict and complete words (figure 4.40).

```
def update(data):
    # clear the list box
    my_list.delete(0, END)

    # add topping to listbox
    for item in data:
        my_list.insert(END, item)
```

Figure 4.40(functions)

this function use when I need to convert text to speech (figure **4.**41).

```
def Voice():
    file = open("voice_file.txt", "r")
    data = file.read()
    file.close()
    engine = pyttsx3.init()  # object creati

    """ RATE"""
    rate = engine.getProperty('rate')  # get
    engine.setProperty('rate', 150)  # setti

    """VOLUME"""
    volume = engine.getProperty('volume')  #
    engine.setProperty('volume', 1.0)  # set
    """VOICE"""
    voices = engine.getProperty('voices')  #
    # engine.setProperty('voice', voices[0].
    engine.setProperty('voice', voices[1].id

    engine.say(data)
    engine.runAndWait()
    engine.stop()
```

Figure 4.41 (voice function)

to change list box when any character are added to the text(figure **4.42**).

```
def checkbutton():
    # grap what was typed
    typed = my_entry.get()

    if typed == '':
        data = topping
    elif " " in typed:
        num = typed.rindex(' ')
        typed1 = typed[num + 1:]
        typed2 = typed[:num - 1]
        data = []
        if typed1 == '':
            ls = nlp(f"{typed2} {nlp.tokenizer.mask_token} ")
            a_key = "token_str"
            values_of_key = [a_dict[a_key] for a_dict in ls]
            data = values_of_key
        else:
            for item in toppings:
                if typed1.lower() in item.lower():
                    data.append(item)

    else:
        data = []
        for item in toppings:
            if typed.lower() in item.lower():
                data.append(item)

    # update item in listbox
    update(data)
```

Figure 4.42 (check button function)

to update entry box with list box clicked (figure **4.43**).

```
# update entry box with listbox clicked
def fillout(e):
    typed = my_entry.get()
    if " " not in typed:
        my_entry.delete(0, END)
    elif " " in typed:
        num = typed.rindex(' ')
        my_entry.delete(num+1, END)

    # add clicked list item to entry box
    my_entry.insert(END, my_list.get(ACTIVE))
```

Figure 4.43 (fill out function)

to create keyboard window (figure 4.44).

```
# size window size
key.geometry('1500x500')
key.title("keyboard")
key.configure(bg='pink')  # add background color
```

Figure 4.44 (window)

we use Transformers which provides APIs to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, and save you time from training a model from scratch. So, we use fill-mask model to predict words in our keyboard. (figure 4.45)

```
from transformers import pipeline

nlp=pipeline('fill-mask')
```

Figure 4.45 (transformers)

Here we write states and then the word predicted after states in the list box below.(figure **4.46**)

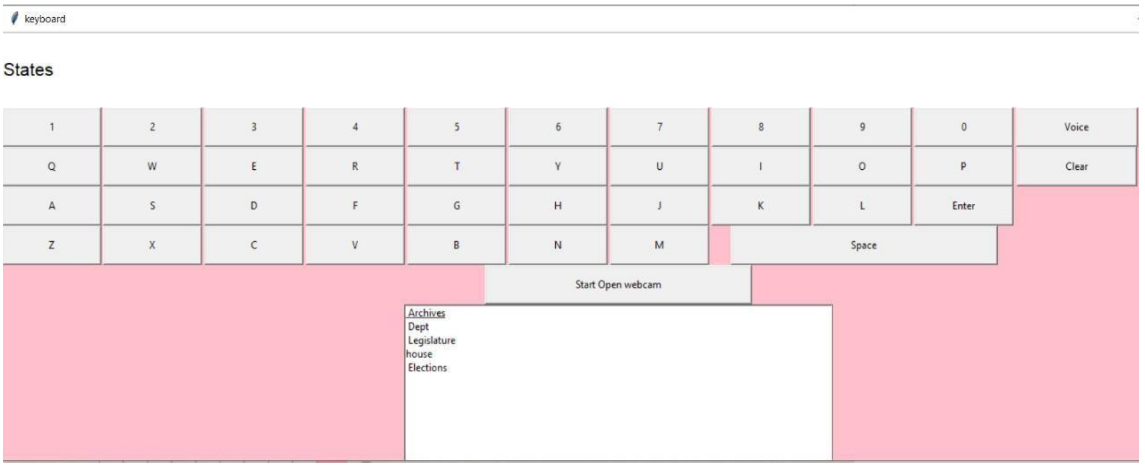


Figure 4.47 (word prediction)

we use text file name pride(Pride and Prejudice by Jane Austen) and make analysis on it to make it just words to auto complete any word I write it in keyboard.(figure 4.48)

```
################################################################################
file = open("pride.txt", "r", encoding = "utf8")
# store file in list
lines = []
for i in file:
    lines.append(i)

# Convert list to string
datas = ""
for i in lines:
    datas = ' '. join(lines)

#replace unnecessary stuff with space
datas = datas.replace('\n', '').replace('\r', '').replace('\ufeff', '').replace('“','').replace('”','').replace('.', '').replace(',','').replace(';','')

#remove unnecessary spaces
datas = datas.split()

datas = list(dict.fromkeys(datas))
print(len(datas))
#####################################################################
```

Figure 4.48 (text analysis)

So when I start write –sta‖ the list box show me a list of words contains –sta‖ in any location on the words in pride text file , so when I just write any character the list box gives to me any words have the same character as we seen in .(figure **4.49**).

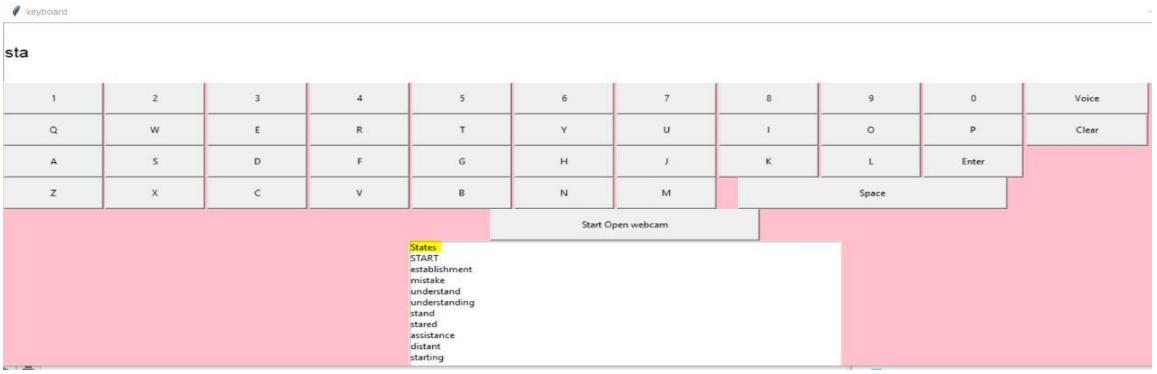


Figure 4.49 (auto complete)

5

Testing

Chapter Summary: In this chapter, we discuss the project testing , detecting eye blinking will

5.1. Blinking Accuracy

First we detect the blink by using this equation we talk about it in the previous chapters.By take the ratio of two lines (Vertical & Horizontal) in the center of eye.

Because when we take just vertical line the numbers changed if I near or far from the camera so we take two lines ratio.

```
lenghtVer, _ = detector.findDistance(leftUp, leftDown)
lenghtHor, _ = detector.findDistance(leftLeft, leftRight)

cv2.line(img, leftUp, leftDown, (0, 200, 0), 3)
cv2.line(img, leftLeft, leftRight, (0, 200, 0), 3)

ratio = int((lenghtVer / lenghtHor) * 100)
ratioList.append(ratio)
if len(ratioList) > 3:
    ratioList.pop(0)
ratioAvg = sum(ratioList) / len(ratioList)
```

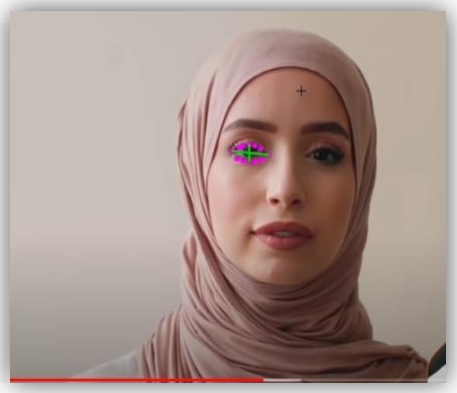


Figure 5.1 (detect lines and calculate the ratio)

So first we create a plot to find when I‘m just flickering or I _m already blink. (figure 5.2)

```
imgPlot = plotY.update(ratioAvg, color)
```

Figure 5.2 (image plot)

Figure 5.3 when I _m not blinking or flickering just open my eyes the ration up 33.

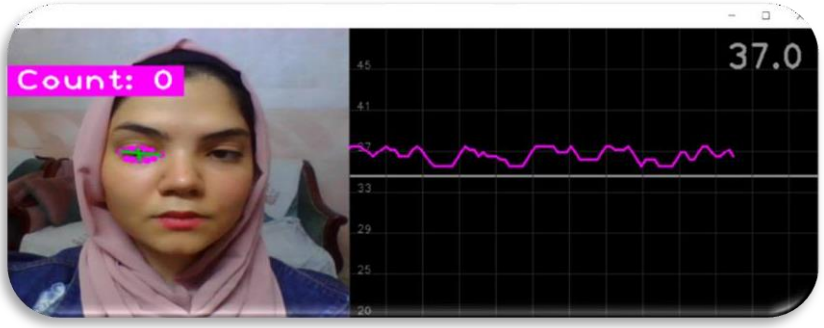


Figure 5.3 (looking forward (not closed))

Figure 5.4 when I‘m just looking down the ration are less than 33 like I _m blinking but I‘m actually not.

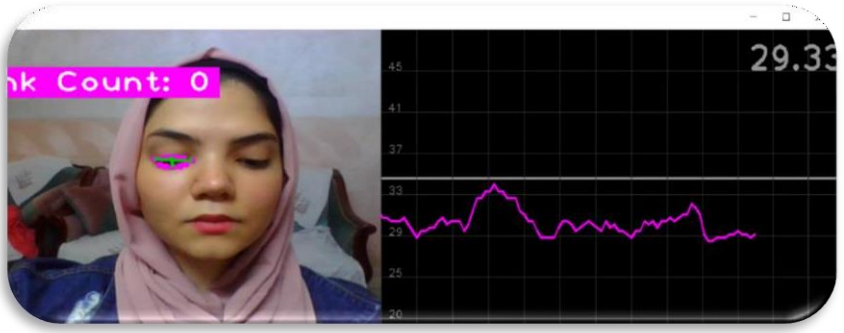


Figure 5.4 (looking down)

Figure 5.5 when I‘m flickering my eyes the ratio between 29 : 33

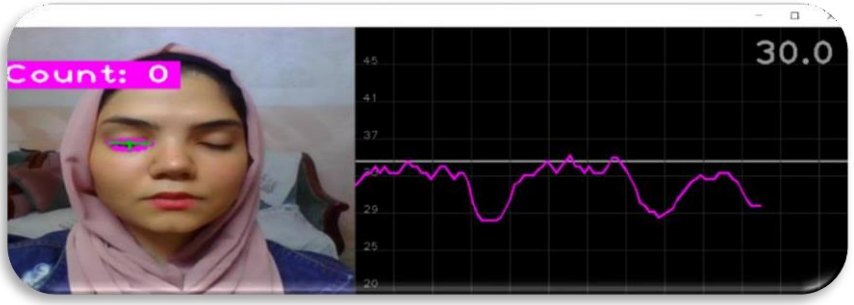


Figure 5.5 (flicker)

Figure 5.6 When I close my eyes well (blinking) the ratio between 25:27

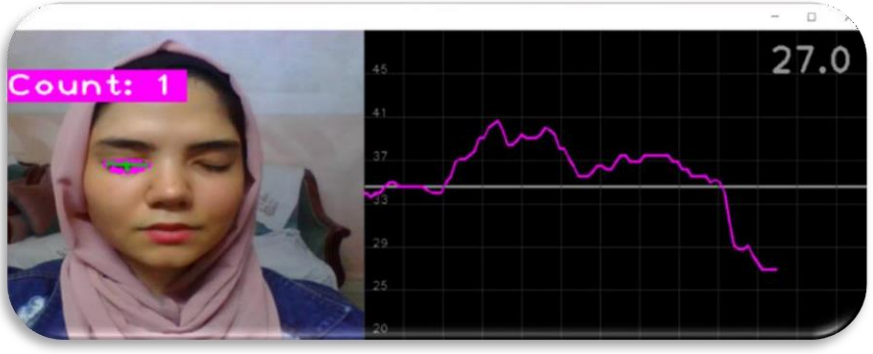


Figure 5.6 (blink)

and this what we want.

So we take any ratio < 27 and the counter count numbers of frame because when we blink it takes like 10 frames closed our eyes on it . (as shown in figure 5.7)

```
if ratioAvg < 27 and counter == 0:
    blinkCounter += 1
    m.click(clicks=2)
    winsound.Beep(500, 250)
    counter = 1
if counter != 0:
    counter += 1
    if counter > 10:
        counter = 0
        color = (255,0, 255)
```

Figure 5.7 (blink)

And we add two double click because some icons in the windos dosen‘t click or open without dlouble click and it will don't be affected on items which one click it have.(as shown in figure 5.8)

```
m.click(clicks=2)
```

Figure 5.8 (double click)

6

Resulting

Chapter Summary: In this chapter, we discuss the project Result , Images Classification with CNN

6.1 Images Classification with CNN

We want to make Image Classification & make our own Dataset by taking many frames in same direction (up, down, left, right). then put its image in train, test & validation then we build our model.

The Dataset

We 'll show you how to make our own dataset

- **Coding**

1. The libraries:

```
import cv2
import os
import time
import uuid
from PIL import Image, ImageOps
```

Figure 6.1 (install the libraries)

2. The Folder for each Direction

```
IMAGES_PATH='Your Path you Want'

labels=['left','right','up','down']

for label in labels:
    path = IMAGES_PATH+"\\"+label
    print(path)
    if not os.path.exists(path):
        !mkdir {path}
```

Figure 6.2 (Make Folder)

3. Take the Photos

```
number_imgs=20

for label in labels:
    cap = cv2.VideoCapture(0)
    print('Collecting images for {}'.format(label))
    time.sleep(4)
    for imgnum in range(number_imgs):
        print('Collecting image {}'.format(imgnum))
        ret, frame = cap.read()
        gray_flip = cv2.flip(frame,1)
        imgname = os.path.join(IMAGES_PATH,label,label+'.'+'{}.jpg'.format(str(uuid.uuid1())))
        cv2.imwrite(imgname, gray_flip)
        cv2.imshow('frame', gray_flip)
        time.sleep(2)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
cap.release()
cv2.destroyAllWindows()
```

Figure 6.3 (Open Webcam & Take Photo)

```
Collecting images for left
Collecting image 0
Collecting image 1
Collecting image 2
Collecting image 3
```

Figure 6.4 (Result)

The Image Augmentation

Due to the small size of the data, we sought to enlarge the size of the data, by changing the lighting and blur, as well as to train the model in all situations and all environments that the image may be exposed to. Until the number of photos reached nearly 1,700 photos.

- **coding**

1. The libraries

```
import imgaug.augmenters as iaa
import cv2
import globa as gb
import os
import imageio
```

Figure 6.5(install the libraries)

2. Loading Images

```
images = []
images_path = gb.glob("path of your images")
for img_path in images_path:
    img = cv2.imread(img_path)
    images.append(img)
```

Figure 6.6(Choose the path, then load images)

3. Apply Filter on Images

```
augmentation = iaa.Sequential([
    # 3. Multiply
        iaa.BlendAlpha([0.25, 0.75], iaa.MedianBlur(13)),
         # 1. Flip
        iaa.Fliplr(0.5),
        iaa.Flipud(0.5),

        # 2. Affine
        iaa.Affine(translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
                   rotate=(-30, 30),
                   scale=(0.5, 1.5)),

        # 3. Multiply
        iaa.Multiply((0.8, 1.2)),

        # 4. Linearcontrast
        iaa.LinearContrast((0.6, 1.4)),

        # Perform methods below only sometimes
        iaa.Sometimes(0.5,
            # 5. GaussianBlur
            iaa.GaussianBlur((0.0, 3.0))
            )

    ])
```

Figure 6.7 (Adding filter on Images)

4. Save Images after adding Filter

```
path="write new Path that you want add Image after adding filter"
os.chdir(path)
```

```
# 3. Show Images
imglist = []
while True:
    augmented_images = augmentation(images=images)
    for img in augmented_images:
        image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        imglist.append(image)

    for i, image_aug in enumerate(images_aug):
        imageio.imwrite("F %d.jpg" % (i,), image_aug)


        cv2.waitKey(0)
```

Figure 6.8(Save Image)

Image Classification

After we make our dataset, we need to make classification

- **coding**

1. The libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
import seaborn as sns
sns.set(style="whitegrid")
import os
import glob as gb
import cv2
import tensorflow as tf
import keras
```

Figure 6.9(install the libraries)

2. Check & Preprocessing our data

```
trainPath='Your path of train dataset'
testPath='Your path of Test dataset'
valPath='Your path of Validtion dataset'
```

```
for folder in os.listdir(trainPath):
    files=gb.glob(pathname=str(trainPath+'//'+folder+'/*.jpg'))
    print(f'For training data,found {len(files)} in folder {folder}')
```

To know how many files/ photos in each direction in train

```
For training data,found 380 in folder down
For training data,found 380 in folder left
For training data,found 380 in folder right
For training data,found 380 in folder up
```

Figure 6.10 (Know how many Images in each folder)

```
for folder in os.listdir(testPath):
    files=gb.glob(pathname=str(testPath+'//'+folder+'/*.jpg'))
    print(f'For test data,found {len(files)} in folder {folder}')
```

files/ photos in each direction in test dataset

```
For test data,found 13 in folder down
For test data,found 13 in folder left
For test data,found 13 in folder right
For test data,found 13 in folder up
```

```
files=gb.glob(pathname=str(valPath+'/*.jpg'))
print(f'For validation data found {len(files)}')
```

files/ photos in each direction in validation

```
For validation data found 20
```

Figure 6.11 (Know how many Images in each folder)

3. Loading Images

```
[ ]  s1=480
     s2=640
```

```
[ ]  code={'down':0,'left':1,'right':2,'up':3}

     def getcode(n):
         for x,y in code.items():
             if n==y:
                 return x
```

```
[ ]  x_train =[]
     y_train =[]
     for folder in os.listdir(trainPath):
         files=gb.glob(pathname=str(trainPath+'//'+folder+'/*.jpg'))
         for file in files:
             #image =cv2.imread(file)
             image = mpimg.imread(file)#RGB
             #image_array=cv2.resize(image,(s,s))
             x_train.append(list(image))
             y_train.append(code[folder])
```

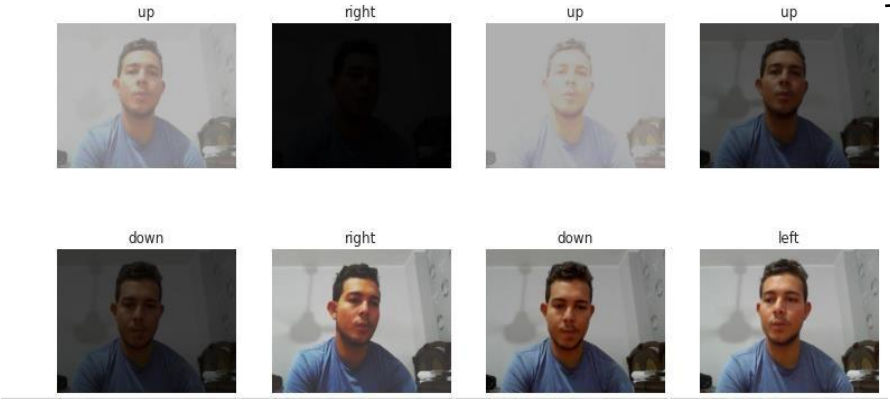
```
[ ]  print(f'we have {len(x_train)} items in X_train')
```

Loading images of training data set

Figure 6.12 (Loading Image of training data)

```
[ ]  plt.figure(figsize=(20,20))
     for n,i in enumerate(list(np.random.randint(0,len(x_train),36))):
         plt.subplot(6,6,n+1)

         plt.imshow(x_train[i])
         plt.axis('off')
         plt.title(getcode(y_train[i]))
```



To show our train dataset

Figure 6.13 (To Show our data)

We will repeat the code in each dataset (test & validation)

```
x_test =[]
y_test =[]
for folder in os.listdir(testPath):
    files=gb.glob(pathname=str(testPath+'//'+folder+'/*.jpg'))
    for file in files:
        #image =cv2.imread(file)
        image = mpimg.imread(file)#RGB
        #image_array=cv2.resize(image,(s,s))
        x_test.append(list(image))
        y_test.append(code[folder])
```

```
print(f'we have {len(x_test)} items in x_test')
```

```
we have 20 items in x_test
```

```
plt.figure(figsize=(20,20))
for n,i in enumerate(list(np.random.randint(0,len(x_test),10))):
    plt.subplot(5,2,n+1)
    plt.imshow(x_test[i])
    plt.axis('off')
    plt.title(getcode(y_test[i]))
```

Figure 6.14 (Loading Image of testing data)

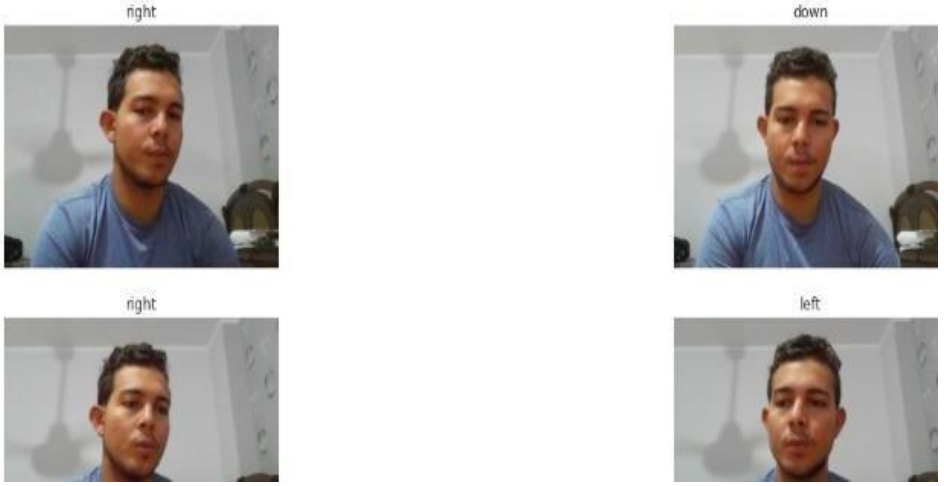


Figure 6.15(To Show our data)

```
x_pred =[]
files=gb.glob(pathname=str(valPath+'/*.jpg'))
for file in files:
    image = mpimg.imread(file)#RGB
    x_pred.append(list(image))
```

```
print(f'we have {len(x_pred)} items in x_pred')
```

```
we have 8 items in x_pred
```

```
plt.figure(figsize=(20,20))
for n,i in enumerate(list(np.random.randint(0,len(x_pred),4))):
    plt.subplot(2,2,n+1)
    plt.imshow(x_pred[i])
    plt.axis('off')
```

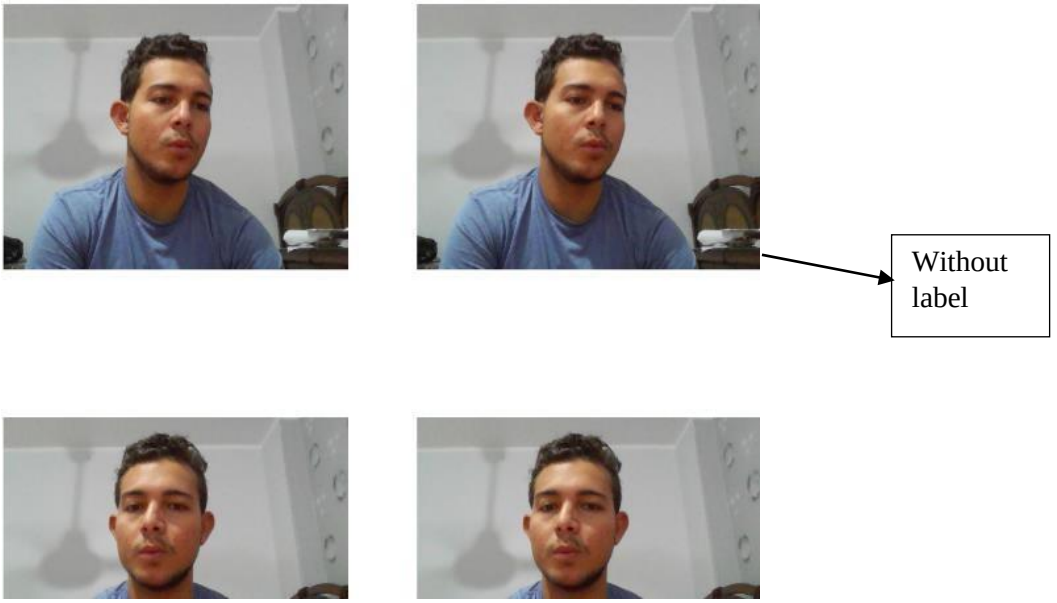


Figure 6.16 (Loading Image of testing data &To Show our data)

4. Changes the Images into array of pixel to deal with Model

```
x_train=np.array(x_train)
x_test=np.array(x_test)
x_pred_arry=np.array(x_pred)
y_train=np.array(y_train)
y_test=np.array(y_test)

print(f'X_train shape is {x_train.shape}')
print(f'X_test shape is {x_test.shape}')
print(f'X_pred shape is {x_pred_arry.shape}')
print(f'y_train shape is {x_train.shape}')
print(f'y_test shape is {y_test.shape}')
```

```
X_train shape is (1605, 480, 640, 3)
X_test shape is (20, 480, 640, 3)
X_pred shape is (8, 480, 640, 3)
y_train shape is (1605, 480, 640, 3)
y_test shape is (20,)
```

Figure 6.17 (pixel data)

5. Building our CNN Model

```
KerasModel = keras.models.Sequential([
        keras.layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(s1,s2 ,3)),
        keras.layers.Conv2D(150,kernel_size=(3,3),activation='relu'),
        keras.layers.MaxPool2D(4,4),
        keras.layers.Conv2D(120,kernel_size=(3,3),activation='relu'),
        keras.layers.Conv2D(80,kernel_size=(3,3),activation='relu'),
        keras.layers.Conv2D(50,kernel_size=(3,3),activation='relu'),
        keras.layers.MaxPool2D(4,4),
        keras.layers.Flatten() ,
        keras.layers.Dense(120,activation='relu') ,
        keras.layers.Dense(100,activation='relu') ,
        keras.layers.Dense(50,activation='relu') ,
        keras.layers.Dropout(rate=0.5) ,
        keras.layers.Dense(4,activation='softmax') ,
        ])
```

```
KerasModel.compile(optimizer ='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

Figure 6.18 (Building Model)

```
Epoch 1/15
6/6 [==============================] - 8s 1s/step - loss: 59.6667 - accuracy: 0.2308
Epoch 2/15
6/6 [==============================] - 8s 1s/step - loss: 4.2852 - accuracy: 0.2500
Epoch 3/15
6/6 [==============================] - 8s 1s/step - loss: 1.3607 - accuracy: 0.4423
Epoch 4/15
6/6 [==============================] - 8s 1s/step - loss: 1.4039 - accuracy: 0.3077
Epoch 5/15
6/6 [==============================] - 8s 1s/step - loss: 1.3364 - accuracy: 0.4808
Epoch 6/15
6/6 [==============================] - 8s 1s/step - loss: 1.1524 - accuracy: 0.4423
Epoch 7/15
6/6 [==============================] - 8s 1s/step - loss: 0.8995 - accuracy: 0.5769
Epoch 8/15
6/6 [==============================] - 8s 1s/step - loss: 0.8806 - accuracy: 0.7692
Epoch 9/15
6/6 [==============================] - 8s 1s/step - loss: 0.2990 - accuracy: 0.8846
Epoch 10/15
```

Figure 6.18 (Result)

6. The Accuracy of our Model

```
ModelLoss, ModelAccuracy = KerasModel.evaluate(x_test, y_test)

print('Test Loss is {}'.format(ModelLoss))
print('Test Accuracy is {}'.format(ModelAccuracy ))
```

```
1/1 [==============================] - 38s 38s/step - loss: 0.2690 - accuracy: 0.9000
Test Loss is 0.26900041103363037
Test Accuracy is 0.8999999761581421
```

Figure 6.19 (Accuracy & loss)

7. Testing our Model

```
y_result = KerasModel.predict(x_pred_arry)

print('Prediction Shape is {}'.format(y_result.shape))
```

```
Prediction Shape is (8, 4)
```

```
plt.figure(figsize=(20,20))
for n , i in enumerate(list(np.random.randint(0,len(x_pred),6)))
    plt.subplot(6,6,n+1)
    plt.imshow(x_pred[i])
    plt.axis('off')
    plt.title(getcode(np.argmax(y_result[i])))
```

Figure 6.20(result of our Prediction)

Our Model get most of validation dataset with correct label

7

Conclusion & Future works

Chapter Summary: In this chapter, we discuss the project conclusion and the future works

7.1 Future work

We had hoped in our project to develop it to make it pupil movement, and this would be the third and final version for us, but due to time constraints, we also faced some challenges that made it more complicated

7.2 Conclusion

This report proposed new system in order to make user interact with computer naturally and conveniently by only using their head and eye, we provide our system to achieve this point. The system combines both the mouse functions and keyboard functions, so that users can use our system to achieve almost all of the inputs to the computer without traditional input equipment such as mouse and keyboard so the user becomes able to move the mouse according to his head pose estimation in all directions , this report also proposed a new approach to assistive virtual keyboard that aims to optimize the performance of typing, reduce the typing errors and minimize the effort required to input text by patients with severe motor disabilities. Entering performance will be optimized through the prediction of letters and words. Finally, a new method has been proposed to both decrease the typing effort and increase the typing speed. The keyboard we proposed in this report is in the first phase of creating is not optimal so we create another updated version. The current version interface is a final product and better ergonomic suitable for use in a real situation is design

8

Appendix

Chapter Summary: In this chapter, we discuss the project media in a website

8.1 Our Website

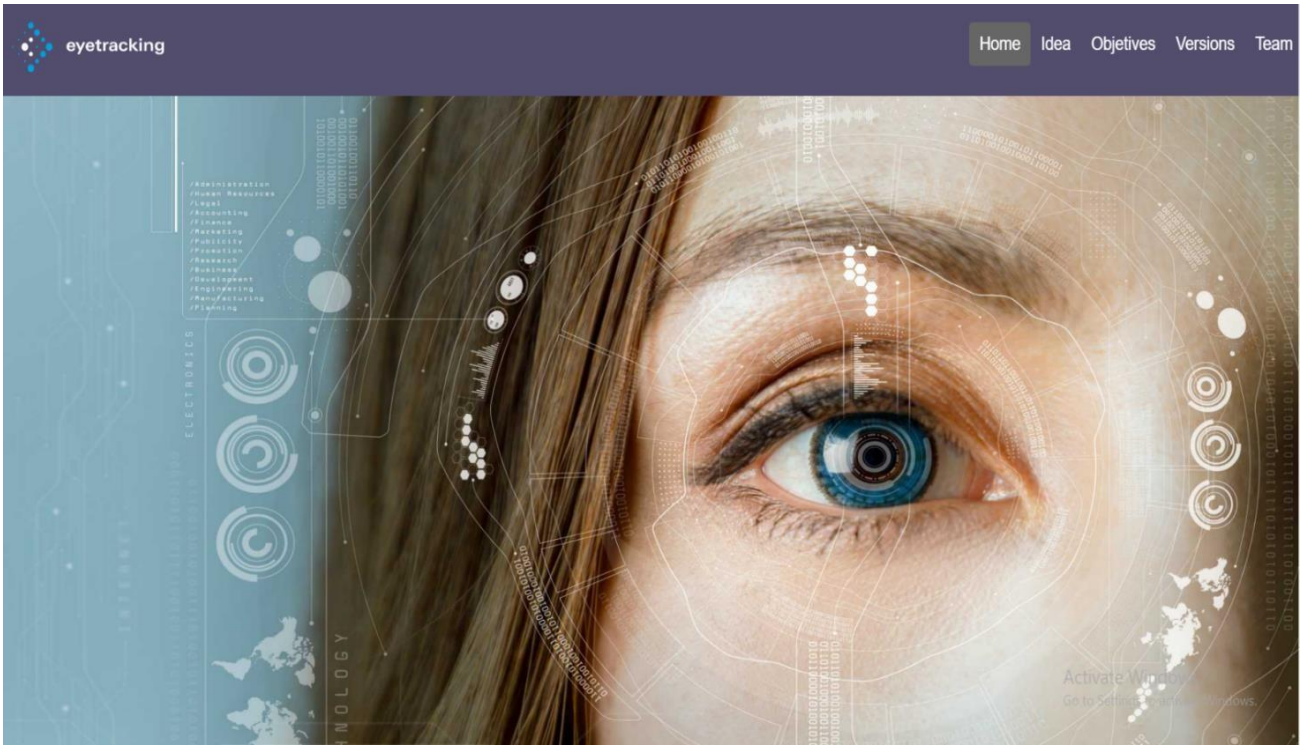


Figure 8.1 (Website)

8.2 Project idea

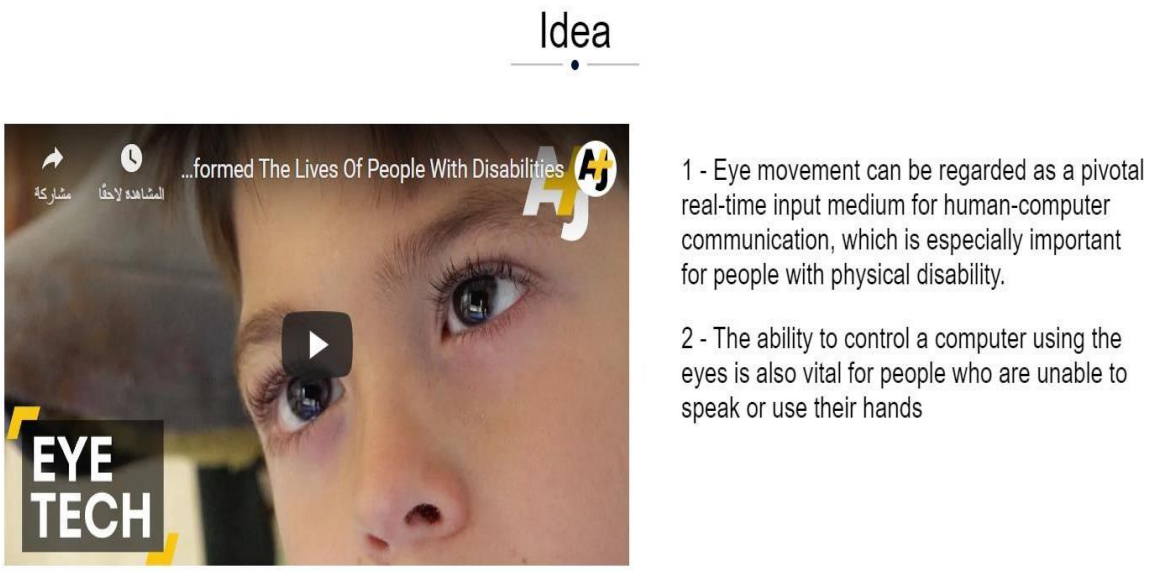


Figure 8.2 (Project idea)

8.3 Project Objective

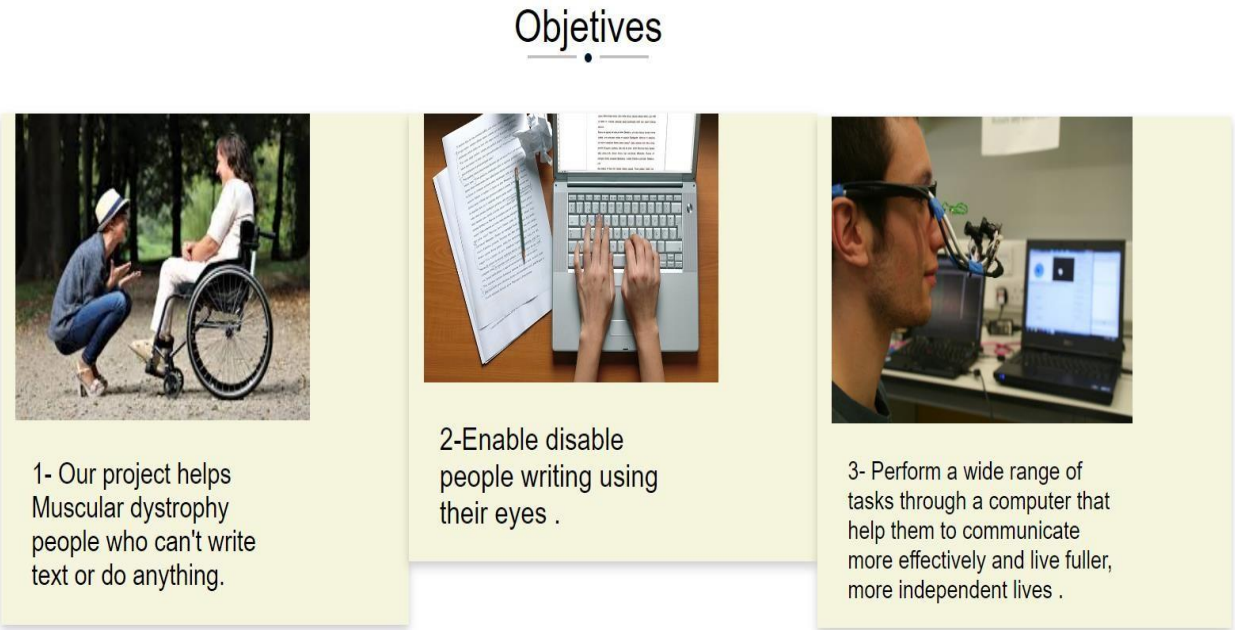


Figure 8.3 (Project idea)

8.3 Project Versions

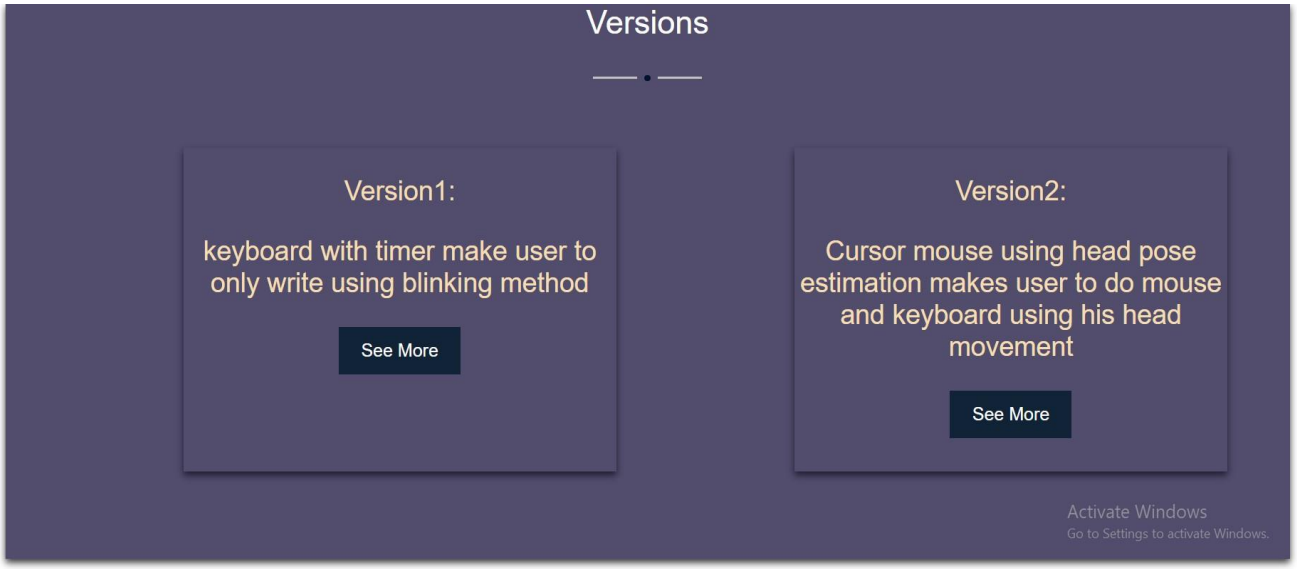


Figure 8.4 (Versions)

8.3 Project Team

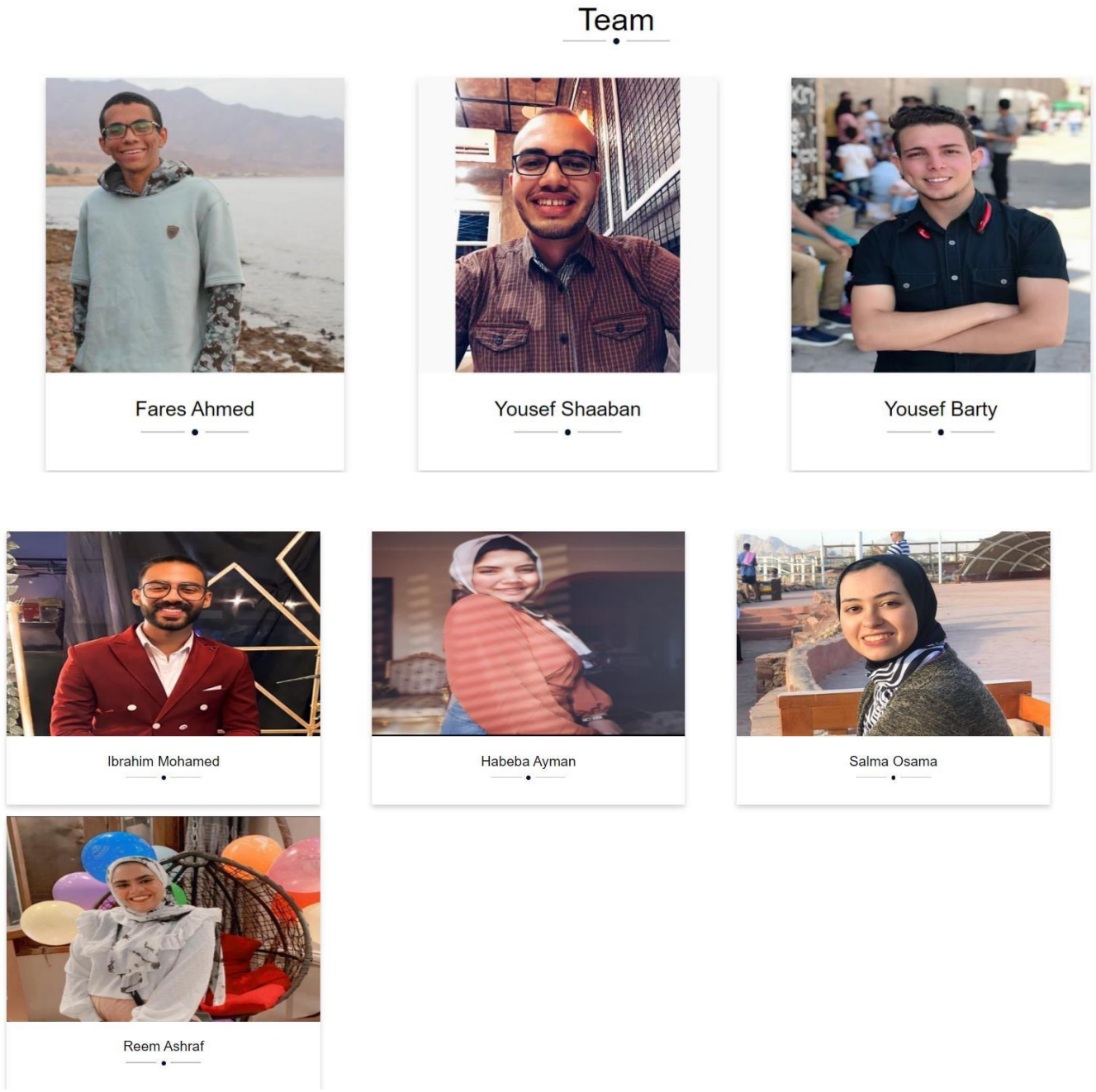


Figure 8.5 (Team member)

REFERENCES

[1]. Sharafi, Zohreh, Timothy Shaffer, and Bonita Sharif, "Eye-Tracking Metrics in Software Engineering, AsiaPacific SoftwareEngineering Conference (APSEC), IEEE pp. 96-103., 2015.

[2]. Shang, L., Zhang, C., & Wu, H, " Eye Focus Detection Based on OpenCV ",.6th International Conference on Systems andInformatics (ICSAI),2019 .

[3]. Chandra, S., Sharma, G., Malhotra, S., Jha, D. and Mittal, A.P.," Eye tracking based human computer interaction: Applications and their uses", International Conference on Man and Machine Interfacing (MAMI) (pp. 1-5). IEEE, December, 2015.

[4]. Zhang, Y., Zheng, X., Hong, W. and Mou, X," A comparison study of stationary and mobile eye tracking on EXITs design in a wayfinding system ", In 2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference(APSIPA). IEEE, December , 2015.

[5]. Miyamoto, D., Iimura, T., Blanc, G., Tazaki, H. and Kadobayashi, Y," EyeBit: eye-tracking approach for enforcing phishing prevention habits ", In 2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS). IEEE, September,2014.

[6]. Tantisatirapong, S., & Phothisonothai, M.," Design of User-Friendly Virtual Thai Keyboard Based on Eye- Tracking Controlled System", 18th International Symposium on Communicationsand InformationTechnologies (ISCIT), 2018.

[7]. Nutthanan Wanluk, Aniwat Juhong," Smart wheelchair based on eye tracking ",. In Design and Technology in Electronic Packaging (SIITME)& IEEE 21st International Symposium for (pp. 161164). IEEE,2018.

[8]. Kshitij Meena, Manish-Kumar, Mohit Jangra, " Eye- tracking for disabled people ", In Design and Technology in Electronic Packaging (SIITME)& IEEE 21st International Symposium for (pp. 139-142). IEEE,2020.

[9]. Sugi, T., Minoda, M., Matsuda, Y., Goto, S., Egashira, N., Oishi, A., & Yamasaki, T ,"Development of a Non- Contact Nurse Call System by Image Processing of Eye Movement. Transactions of the Institute of Systems ", Control and Information Engineers, 2019.

[10]. Dahmani, M., Chowdhury, M. E. H., Khandakar, A.,Rahman, T., Al-Jayyousi, K., Hefny, A., & Kiranyaz,S," An Intelligent and Low-Cost Eye-Tracking System for Motorized Wheelchair Control. Sensors ", 2020

[11]. Barbara, N., Camilleri, T. A., & Camilleri, K. P, " EOG-based eye movement detection andgaze estimationfor an asynchronous virtual keyboard ". Biomedical Signal Processing and Control, 2020

[12]. A Computer vision package that makes its easy to run Image processing (pythonawesome.com)[13]. Software Engineering tenth edition Ian Sommerville

[14]. paper.dvi (cmu.edu)

[15]. Face Detection - mediapipe (google.github.io)[16]. Iris - mediapipe (google.github.io)

[17]. Face Recognition — Face Recognition 1.4.0 documentation (face-recognition.readthedocs.io)

[18] Ivan Culjak “A brief introduction to OpenCV” MIPRO, 2012 Proceedings of the 35th InternationalConvention.

[19] Anupam Agrawal, Rohit Raj and Shubha Porwal “Vision-based multimodal human-computer interaction using hand and head gestures” Information & Communication Technologies (ICT),2013 IEEE Conference.

[20] https://ethanjli.github.io/doc/projects/cs-231a-webcam-cursor-control/report.pdf

[21] https://cogsys.uni-bamberg.de/theses/rieger/Masterarbeit_HeadPoseEstimation_InesRieger.pdf

[22] https://medium.com/analytics-vidhya/face-pose-estimation-with-deep-learning-eebd0e62dbaf

[23] https://onlinelibrary.wiley.com/doi/full/10.1111/exsy.12398