BLINK TO TEXT

MINOR PROJECT REPORT

Submitted in partial fulfilment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

by

Mr. Arnav Sahai	Mr. Atishay Jain	Mr. Ayush Rawat	Mr. Harshit Kumar
En No: 01515002715	En No: 01915002715	En No:02115002715	En No: 02915002715

Guided by

Dr. Rinky Dwivedi

Associate Professor, MSIT

To



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY

(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY, DELHI)

C-4, JANAKPURI, NEW DELHI-110058

DECEMBER 2018

CANDIDATE'S DECLARATION

It is hereby certified that the work which is being presented in the B. Tech Major Project Report entitled **"BLINK TO TEXT"** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** and submitted in the **Department of Computer Science & Engineering** of **MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY, New Delhi (Affiliated to Guru Gobind Singh Indraprastha University, Delhi**) is an authentic record of our own work carried out during a period from **July 2018 to November 2018** under the guidance of **Dr. Rinky Dwivedi, Associate Professor**.

The matter presented in the B. Tech Minor Project Report has not been submitted by me for the award of any other degree of this or any other Institute.

Mr. Arnav Sahai	**Mr. Atishay Jain**	**Mr. Ayush Rawat**	**Mr. Harshit Kumar**
En. No: 01515002715	**En. No: 01915002715**	**En. No: 02115002715**	**En No:02915002715**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge. They are permitted to appear in the External Minor Project Examination.

Dr. Rinky Dwivedi	**Dr. Koyel Dutta Gupta**
(Associate Professor)	**(HOD, CSE MSIT)**

The B. Tech Minor Project Viva-Voce Examination of Name of the Student (Enrollment No: XXX), has been held on .....

Project Coordinator	**Project Coordinator**	**(Signature of External Examiner)**

ACKNOWLEDGEMENT

We express our deep gratitude to **Dr. Rinky Dwivedi,** Associate Professor, Department of Computer Science & Engineering for her valuable guidance and suggestion throughout our project work. We are thankful to **Ms. Shaily Malik,** Project Coordinator, for her valuable guidance.

We would like to extend my sincere thanks to **Head of the Department, Dr. Koyel Dutta Gupta** for her time to time suggestions to complete our project work.

At last we thank the almighty, who has given the strength to complete this project on time.

Finally, we would like to thank our parents, all friends, and well-wishers for their valuable help and encouragement throughout the project.

Sign	**Sign**	**Sign**	**Sign**
(Arnav Sahai)	**(Atishay Jain)**	**(Ayush Rawat)**	**(Harshit Kumar)**
(En. No: 01515002715)	**(En. No: 01915002715)**	**(En. No: 02115002715)**	**(En No: 02915002715)**

TABLE OF CONTENTS

TABLE OF CONTENTS iv

LIST OF FIGURES v

LIST OF ABBREVIATION vi

Chapter 1: Introduction 1

1.1 The Problem 1

1.2 The Solution 1

1.3 Technology Used 2

Chapter 2: Objective 5

Chapter 3: Literature Survey And Methodology 6

3.1 Literature Survey 6

3.2 Face Detection with HOG+SVM 10

3.3 Facial Landmarking (One millisecond face alignment with an ensemble of regression tree) 14

3.4 Detecting Blinks 18

3.5 Electron GUI (Java Script) 20

Chapter 4: Main Implementation and Results 24

4.1 Research Methodology 24

4.2 Results and Working of the Application 25

4.3 Code Snippets 26

Chapter 5: Conclusion and Future Scope 34

REFERENCES 35

LIST OF FIGURES

Figure 1 Calculation of Integral Image 6
Figure 2 Important Rectangle Features 7
Figure 3 Meaningful features selected by AdaBoost Algorithm 8
Figure 4 Steps involved in HOG face Detection Technique 9
Figure 5 Pixel Representation 10
Figure 6 Image area 11
Figure 7 Gradient Representation for the Image Area 12
Figure 8 HOG face descriptor 13
Figure 9 Overview of the model 15
Figure 10 Extracting 400 pixels in the detected face 17
Figure 11 Performing similarity transform on the image 17
Figure 12 Facial Landmarking and its different stages 18
Figure 13 Predefined Facial Landmarks 19
Figure 14 EAR formula 19
Figure 15 EAR Value for a Natural Eye Blink 20
Figure 16 Grid containing letters for typing 21
Figure 17 Letter highlighted is the letter to be selected 22
Figure 18 Selected letters appear in the textbox above 22
Figure 19 Camera output with eyes marked is shown in a separate window 23
Figure 20 Displaying text “Hello can you bring me some water” on the GUI 25
Figure 21 Importing necessary python module 26
Figure 22 Definition of EAR and detector function with threshold value 27
Figure 23 Extraction of eye region from Dlib generated 68-point facial array 28
Figure 24 Conversion from BGR color to grey color scheme 29
Figure 25 Detection of blink status 30
Figure 26 Toggling Letters in Grid 31
Figure 27 Autocorrect 32
Figure 28 Detecting State of Blink 33

LIST OF ABBREVIATION

HOG	HISTOGRAM OF GRADIENTS
ANN	ARTIFICIAL NEURAL NETWORK
FD	FACE DETECTION
GUI	GRAPHICAL USER INTERFACE
SVM	SUPPORT VECTOR MACHINE
DS	DATASET
DT	DECISION TREE
ST	SIMILARITY TRANSFORM

Chapter 1: Introduction

1.1 The Problem

Paralyzed people lack the ability to control muscle function in one or more muscle groups. The condition can be caused by stroke, ALS, multiple sclerosis, and many other diseases. Locked-in Syndrome (LIS) is a form of paralysis where patients have lost control of nearly all voluntary muscles. These people are unable to control any part of their body, besides eye movement and blinking. Due to their condition, these people are unable to talk, text, and communicate in general. Even though people that have LIS are cognitively aware, their thoughts and ideas are locked inside of them. These people depend on eye blinks to communicate. They rely on nurses and caretakers to interpret and decode their blinking. Whenever LIS patients do not have a person to read their eye blinks available, they have no means of self-expression.

1.2 The Solution

BlinkToText offers a form of independence to paralyzed people. The software platform converts eye blinks to text. Every feature of the software can be controlled by eye movement. Thus, the software can be independently operated by paralyzed people. Using the software, patients can record messages, recite those messages aloud, and send the messages to others. The software can be run on any low spec computer, from a Raspberry Pi to an IBM Thinkpad. The software uses computer vision to solve the problem and uses Histogram of Gradient to detect face and Facial land marking to detect blink like condition. The program uses language modelling to predict the next words that the user might blink. The software can be easily customized for each patient as well. Blink to Text is free open source software.

1.3 Technology Used

- OS used

  Ubuntu 18.04 LTS

- Programming Language

  Python

- Packages Used

  Imutils

  Scipy

  Dlib

  Cv2(OpenCv)

  Threading

  Numpy

- GUI

  Electron

  JavaScript

  Zerorpc

1.Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. In July 2018, Van Rossum stepped down as the leader in the language community after 30 years.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development

model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

2.Imutils

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and both Python 2.7 and Python 3.

3.Scipy

SciPy, a scientific library for Python is an open source, BSD-licensed library for mathematics, science and engineering. The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The main reason for building the SciPy library is that, it should work with NumPy arrays. It provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. This is an introductory tutorial, which covers the fundamentals of SciPy and describes how to deal with its various modules.

4.Dlib

Dlib is a general-purpose cross-platform open source software library written in the C++ programming language. Its design is heavily influenced by ideas from design by contract and component-based software engineering. This means it is, first and foremost, a collection of independent software components, each accompanied by extensive documentation and thorough debugging modes.

5.Cv2(OpenCv)

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it’s free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform adopted all around the world, OpenCV has more than 47 thousand people of

user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

6.Numpy

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

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

7.Electron

Electron is an open-source framework developed and maintained by GitHub. Electron allows for the development of desktop GUI applications using front and back end components originally developed for web applications: Node.js runtime for the backend and Chromium for the frontend.

8.JavaScript

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

9. Zerorpc

Zerorpc is a light-weight, reliable and language-agnostic library for distributed communication between server-side processes. It builds on top of ZeroMQ and MessagePack. Support for streamed responses - similar to python generators - makes zerorpc more than a typical RPC engine.

Chapter 2: Objective

2.1 Allow paralysis victims to communicate independently

Many paralysis victims already use eye blinks as a form of communication. It is common for nurses and caretakers to read a patient's eye blinks and decode the pattern. The ALS association even offers a communication guide that relies on eye blinks. BlinkToText automates this task. The software reads a person's eye blinks and converts them into text. A key feature of the software is that it can be started, paused, and operated entirely with eye blinks. This allows patients to record their thoughts with complete independence. No nurses or caretakers are required to help patients express themselves. Not only does this reduce the financial burden on paralysis patients, but this form of independence can be morally uplifting as well.

2.2 Be accessible to people with financial constraints

Many companies are developing technologies that are controlled by eye movement. These technologies rely on expensive hardware to track a user`s eyes. While these devices can absolutely help LIS victims, they are only available to people that can afford the technology. BlinkToText focuses on a different demographic that are often ignored. BlinkToText is free and open source. The software runs on wide variety of low-end computers. The only required peripheral is a basic webcam. Not only is this software accessible to paralyzed people, but paralyzed people of almost all financial classes as well.

Chapter 3: Literature Survey And Methodology

3.1 Literature Survey

Viola Jones VS HOG + SVM

The process of face detection however easy for a human mind whether it be the detection of faces in still images, dynamic videos or in real time is quite difficult for computed systems, this difficulty associated with face detection can be attributed to variations in scale, illumination, location, view point, occlusions etc. For the sake of simplicity and to tackle the aforementioned problem a study was conducted as presented in [1] on Viola-Jones face detection method and HOG + SVM detection method to determine which is more suited for our application.

Viola -Jones Detection Technique

As well explained in this technique is divided into three main integral parts. The first is the creation of new image representation called the “Integral Image” which allows for better and faster feature extraction. Integral Image also known as a summed area table is an algorithm for quickly and efficiently computing the sum of values in a rectangular subset of pixel grid. The integral image at location *x, y* contains the sum of pixels above and to the left of *x, y*.

$$ii\,(x, y) = \sum_{x' \le x, y' \le y} i(x', y')$$

where $ii\,(x, y)$ is the integral image and $i\,(x, y)$ is the original image.

Once the summed-area table has been computed, evaluating the sum of intensities over any rectangular area requires exactly four array references regardless of the area size. That is, the notation in the figure at right, having A= (x_0, y_0), B= (x_1, y_0), C= (x_0, y_1) and D= (x_1, y_1), the sum of $i\,(x, y)$ over the rectangle spanned by A, B, C and D is:

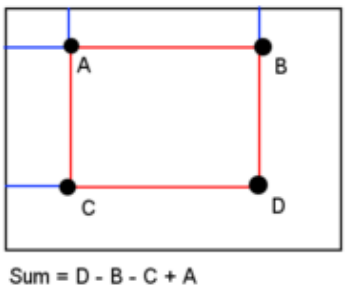


Figure 1 Calculation of Integral Image

$$\sum_{x_0 \le x \le x_1, y_0 \le y \le y_1} i(x, y) = I(D) + I(A) - I(B) - I(C) \qquad (1)$$

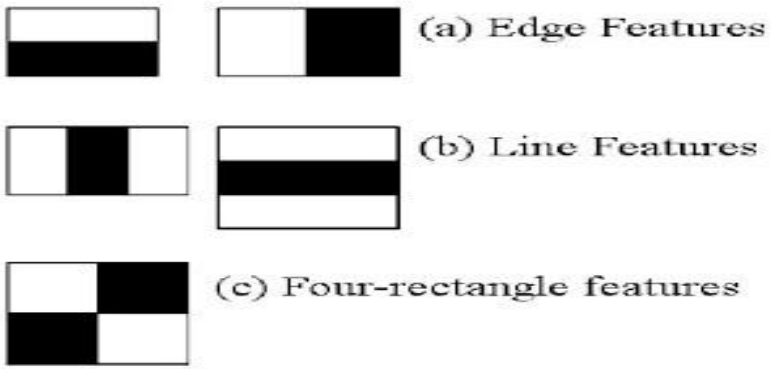


Figure 2 Important Rectangle Features

Example rectangle features shown relative to detection window, the sum of pixels in the white are subtracted from sum of pixels in the dark. Here (a), (b) and (c) depict two, three and four rectangle feature respectively.

The second integral part is the use of learning algorithm AdaBoost which selects a small number of critical visual features from a larger set and yields efficient classifiers. This is because an image sub window has a plethora of Haar like features and only a few are important or needed to be used to detect a face. The features being talked about in this paper are rectangular features and there are many reasons to use them rather than pixels directly, the most common being that features can act to encode ad-hoc domain knowledge which is difficult to learn using a finite quantity of training data set.

Specifically three main features are used, firstly two rectangle feature which is the difference between the sum of the pixels within two rectangular regions which have the same size and shape and are horizontally or vertically adjacent(See Figure 2),Similarly a three rectangle feature computes the sum within two outside rectangles subtracted from the sum in the in a center rectangle and finally a four-rectangle feature which calculates the differences between diagonal pairs of rectangles.

For detection of faces Ada Boost uses meaningful and easily interpreted features such as the region of eyes is darker than the region of nose and cheeks

or the region of eyes are darker than the bridge of nose (See Figure 3). The weak learning algorithm of AdaBoost is designed to select that single rectangle feature which is effectively able to distinguish between positive and negative samples. For each feature that the weak learner encounters it determines the best threshold classification function, so the least number of enables are misclassified. The threshold classification function is as follows:

$$h_j(\mathrm{x}) = \begin{cases} 1 \; if \; p_j f_j(x) \; < \; p_j \theta_j \\ 0 \; otherwise \end{cases} \qquad (2)$$

Where f_j is feature, p_j is polarity, θ_j is threshold

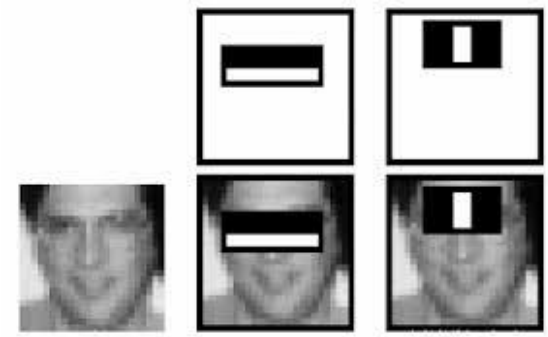


Figure 3 Meaningful features selected by AdaBoost Algorithm

The third integral feature of Viola Jones Detection is a method of combining successively more complex classifiers in a "cascade" structure which precipitates the speed of detector by paying attention to only those regions in an image where the probability of finding the object in consideration, in our case faces is highest. This is based on the understanding that small and more efficient classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances. These simpler classifiers reject the majority of sub windows before more complex classifiers can be called to achieve low false positive rates.

The overall detection process is similar to a degenerate decision tree, which is called a "cascade" (See Figure 4). A positive outcome from the first classifier initiates the detection process by second classifier which in turn is tuned to different adjustment so as to achieve high detection rates. A positive response from second classifier similarly triggers a third classifier and so on. A negative outcome at any point leads to the immediate rejection of the sub-window. Each stage in the cascade is constructed by training classifiers using AdaBoost and then adjusting their threshold to minimize false positive rates. The fact that within every image an

overwhelming majority of sub windows are negative is effectively reflected by the structure of cascade. Like a decision tree every subsequent classifier are trained on examples which pass through the preceding classifier, this makes these examples much "harder" than the typical examples

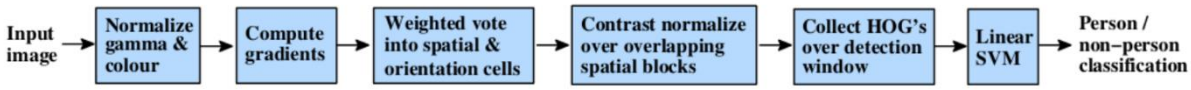


Figure 4 Steps involved in HOG face Detection Technique

Figure 3.4: Schematic depiction of the detection cascade. This shows a sub-window is passed through a series of classifiers where the initial classifier eliminates a large number of negative examples which reduces the number of sub-windows drastically after several stages of processing. Also, with the addition of subsequent layers increases the computational time.

HOG + SVM Detection Technique

As well explained in [3] the method evaluated well- normalized histograms of image gradient orientation into a dense grid. The main idea is that local object appearance and shape can be frequently described effectively by the distribution of local intensity gradients or edge directions without even having any knowledge about the corresponding gradient or edge positions. In this technique the image window is divided into small spatial regions("cells"), for each cell accumulating a local 1-D histogram of gradients or edge orientations over the pixels of the cell. These histograms are combined to form a representation. For independence from illumination, shadowing etc. such that they don't affect in detection the local responses are contrast normalized before using them which is done by accumulating a measure of local histogram "energy", over somewhat larger spatial regions ("blocks") and using the results to normalize all the cells in the block. Covering the detection window with a dense and overlapping grid of HOG descriptors along with using combined feature vector in a conventional SVM based window classifier gives the below illustrated chain (See Figure 5).

3.2 Face Detection with HOG+SVM

Face detection can be regarded as a specific case of object-class detection. In object-class detection, the task is to find the locations and sizes of all objects in an image that belong to a given class. Examples include upper torsos, pedestrians, and cars.

Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stores in database. Any facial feature changes in the database will invalidate the matching process.

Technique used in Face Detection for our use case (HOG-Histogram of Gradient)

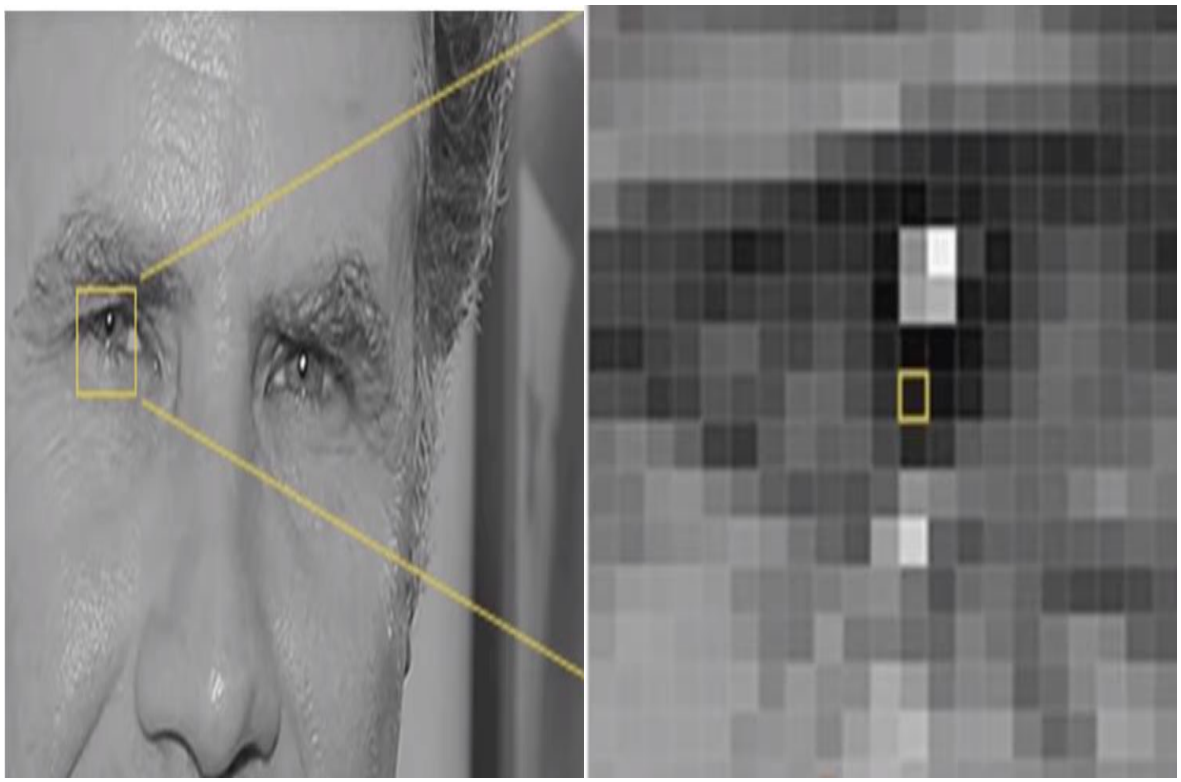


Figure 5 Pixel Representation

3.2.1 What is a HOG?

- Developed by Navneet Dalal and Bill Triggs
- Still uses hand coded features
- But it proved to be much more efficient
- For every single pixel, we looked for pixel surrounding it.
- Goal is how dark is current pixel compared to surrounding pixels.
- Every pixel is replaced by an arrow are called gradient
- (not to be confused by gradient in gradient descent in back propagation)
- Gradient show the flow from light to dark across the entire image
- Then, 16*16-pixel square area is considered
- For each such area such gradient points are counted in each major direction.
- End result is the original image dataset is converted into simple representation that captures the essence of the image.
- Detecting face means find part of our image that looked similar to known HOG pattern (that was generated with the image dataset with hand-coded features it was trained on)



Figure 6 Image area

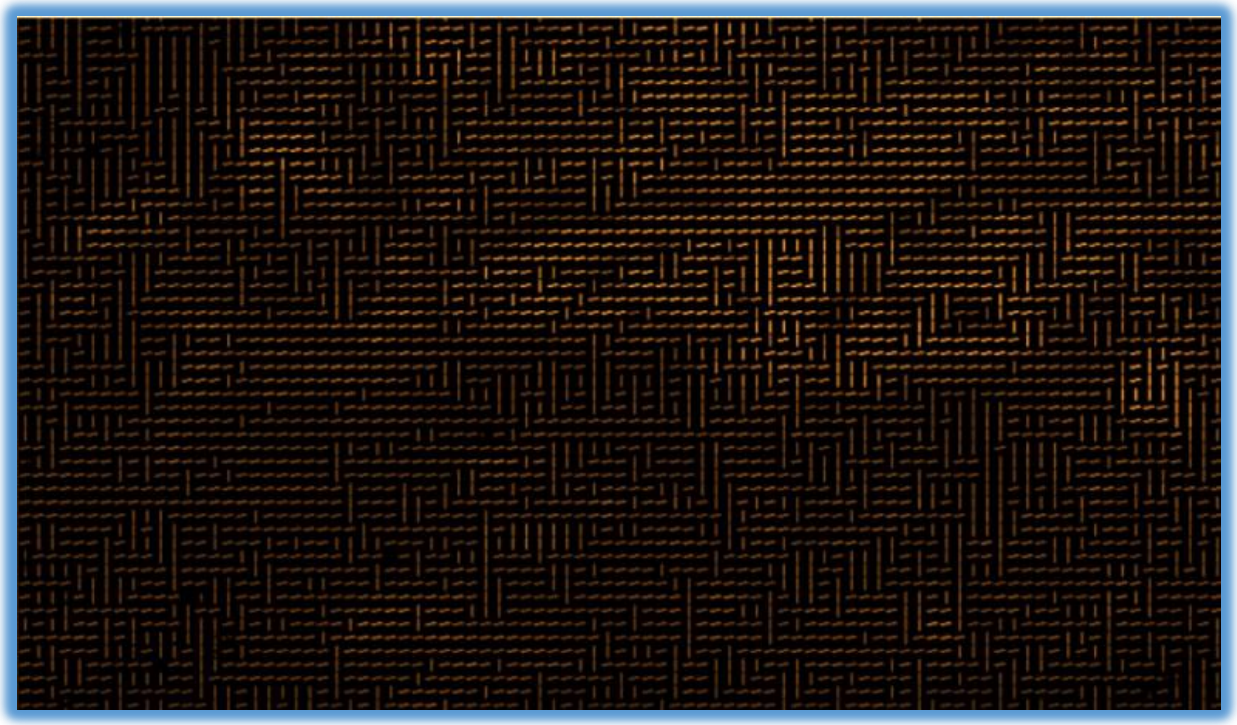


Figure 7 Gradient Representation for the Image Area

3.2.2 How was Hog used to detect face?

HOG is a dense feature extraction method for images. Dense means that it extracts features for all locations in the image (or a region of interest in the image) as opposed to only the local neighborhood of key-points like SIFT.

Intuitively it tries to capture the shape of structures in the region by capturing information about gradients. It does so by dividing the image into small (usually 8x8 pixels) cells and blocks of 4x4 cells. Each cell has a fixed number of gradient orientation bins. Each pixel in the cell votes for a gradient orientation bin with a vote proportional to the gradient magnitude at that pixel.

To reduce aliasing, the pixels votes are bilinearly interpolated. This interpolation happens in both the orientation as position. This statement is important - it means that a pixel will not only vote for its orientation bin, but also for the two neighboring orientation bins (e.g. it the gradient orientation at a pixel is 45 degrees, it will vote with a weight of 0.5 for the 35 to 45-degree bin and a weight of 0.5 for the 45 to 55-degree bin). Similarly, it will vote for these two orientation bins not only in its cell, but also in the 4 neighboring cells of its cell. The weights here are

decided by the distance of the pixel from the cell centers.

Histograms are also normalized based on their energy (regularized L2 norm) across blocks. Since the blocks have a step size of 1 cell, a cell will be part of 4 blocks. This defines four differently normalized versions of the cell's histogram. These 4 histograms are catenated to get the descriptor for the cell. Typically, the elements of histograms are also capped at some value.

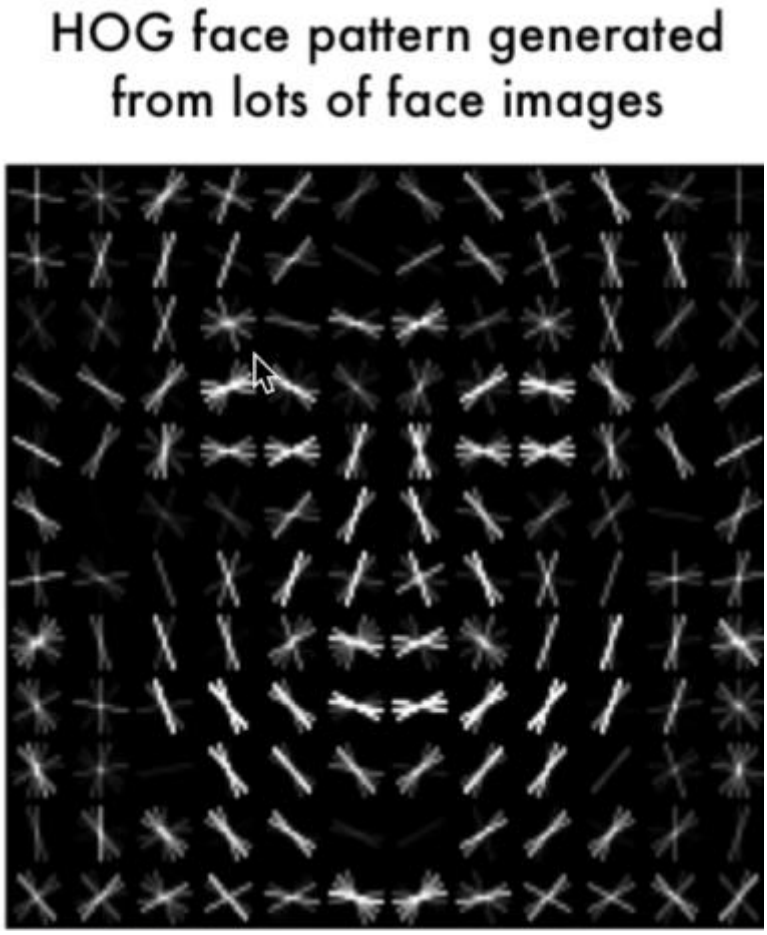


Figure 8 HOG face descriptor

3.3 Facial Landmarking (One millisecond face alignment with an ensemble of regression tree)

Face landmarking, defined as the detection and localization of certain characteristic points on the face, is an important intermediary step for many subsequent face processing operations that range from biometric recognition to the understanding of mental states. Despite its conceptual simplicity, this computer vision problem has proven extremely challenging due to inherent face variability as well as the multitude of confounding factors such as pose, expression, illumination and occlusions. The purpose of this survey is to give an overview of landmarking algorithms and their progress over the last decade, categorize them and show comparative performance statistics of the state of the art. We discuss the main trends and indicate current shortcomings with the expectation that this survey will provide further impetus for the much needed high-performance, real-life face landmarking operating at video rates.

3.3.1: Dataset used –Helen Dataset

It contains large set of candidate photos was gathered using a variety of keyword searches on Flickr. In all cases the query included the keyword ``portrait'' and was augmented with different terms such as ``family'', ``outdoor'', ``studio'', ``boy'', ``wedding'', etc. (An attempt was made to avoid cultural bias by repeating the queries in several different languages.) A face detector was run on the resulting candidate set to identify a subset of images that contain sufficiently large faces (greater than 500 pixels in width). The subset was further filtered by hand to remove false positives, profile views, as well as low quality images. For each accepted face, we generated a cropped version of the original image that includes the face and a proportional amount of background. In some cases, the face is very close or in contact with the edge of the original image and is consequently not centered in the cropped image. Also, the cropped image can contain other face instances since many photos contain more than one person in close proximity.

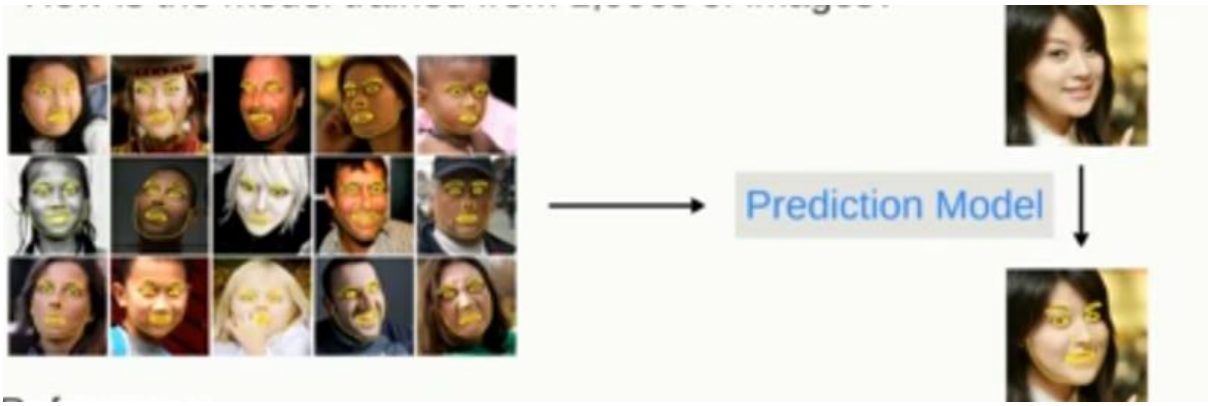


Figure 9 Overview of the model

3.3.2 Face Alignment with an ensemble of Regression Trees

This technique uses a collection of regression trees to estimate the face's landmark positions directly from a sparse subset of pixel intensities often achieving high performance and quality predictions. A general framework is presented that optimizes the sum of square error loss and naturally handles missing or partially labelled data, which is based on gradient boosting for learning an ensemble of regression trees.

The problem of face alignment is solved with the use of cascade of regression functions where each function in the cascade efficiently estimates the shape from an initial estimate and the intensities of a sparse set of pixels indexed relative to this initial estimate. Two key elements are incorporated into learnt regression functions. The first which revolves around indexing of pixel intensities that is relative to the shape of the face. The extracted features of a face image that are represented in a vector can vary greatly due to shape deformation as well as changes in illumination conditions, which makes accurate shape prediction using these extracted features difficult but to accurately predict the shape we need reliable features and on other hand we need an accurate estimate of shape to extract reliable features. To solve this problem the image is transformed into a normalized coordinate system based on the current estimate of the image from where the features are extracted to predict an update vector for the shape parameters rather than regressing the shape parameters based on features extracted in the global coordinate system of the image. The above-mentioned process is usually repeated until convergence.

The second key element takes into consideration, how to combat the problem of inference/prediction? At test time an alignment algorithm estimates the shape in the form of a high dimensional vector that best agrees with the image data and our model of shape. The problem has many local optima which is solved by assuming that the estimated shape must lie in a linear subspace which can be discovered. For example, finding the principal components of the training shape. The above stated assumption thus greatly reduces the number of potential shapes considered during inference and helps in avoiding local optima.

Accompanied with the above two key factors is efficient regression learning wherein an appropriate loss function is optimized and feature selection is performed in a data driven manner. Each regressor is learned via gradient boosting with a squared error loss function. A combination of gradient boosting algorithm and prior probability on the distance between pairs of input pixels is used to select the sparse pixel set which is used as the regressor's input. The prior distribution allows the boosting algorithm to efficiently explore a large number of relevant features which results in cascade of regressors than can localize the facial landmarks when initialized with a mean face pose.

3.3.3 How It Works

- After face detection
- We take the mean shape from the model and apply it to the detected face (Fig 2.1)
- Extract 400 pixels and apply stage 1 regressor

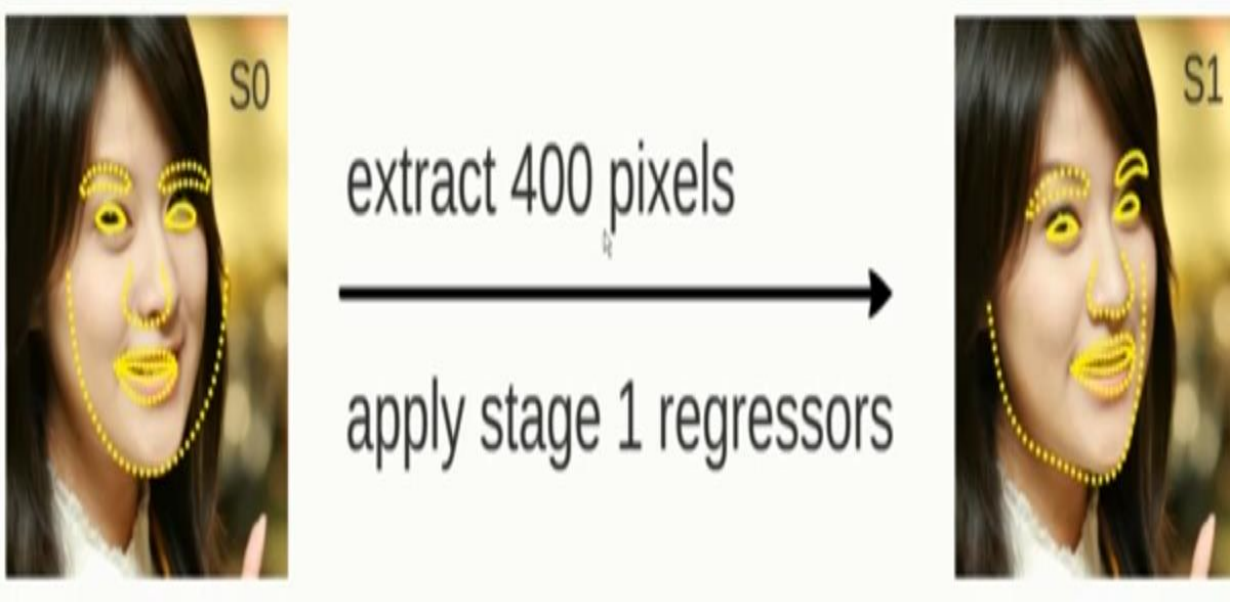


Figure 10 Extracting 400 pixels in the detected face

- From the extracted 400 points we perform a similarity transform to make the landmark shape much better.

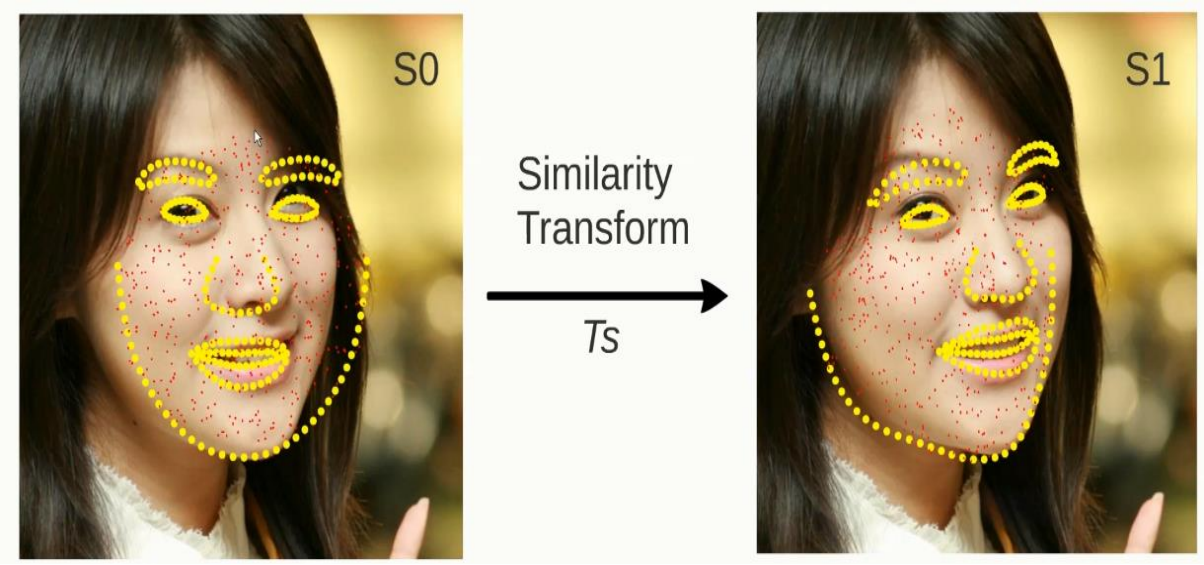


Figure 11 Performing similarity transform on the image

- Ten such regressor are applied to get a perfect facial landmark

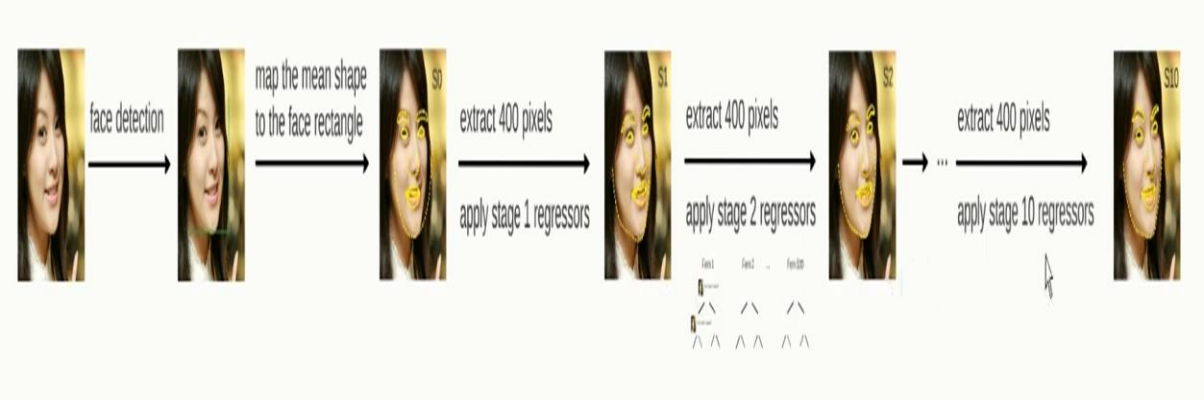


Figure 12 Facial Landmarking and its different stages

- Internally it uses decision tree for coming up to a most probable shape after such similarity transform.

3.4 Detecting Blinks

- The facial landmark detector implemented inside dlib produces 68 *(x, y)*-coordinates that map to *specific facial structures*. These 68 point mappings were obtained by training a shape predictor on the labeled iBUG300-W dataset.
- The *mouth* can be accessed through points *[48, 68]*.
- The *right eyebrow* through points *[17, 22]*.
- The *left eyebrow* through points *[22, 27]*.
- The *right eye* using *[36, 42]*.
- The *left eye* with *[42, 48]*.
- The *nose* using *[27, 35]*.
- And the jaw via *[0, 17]*.

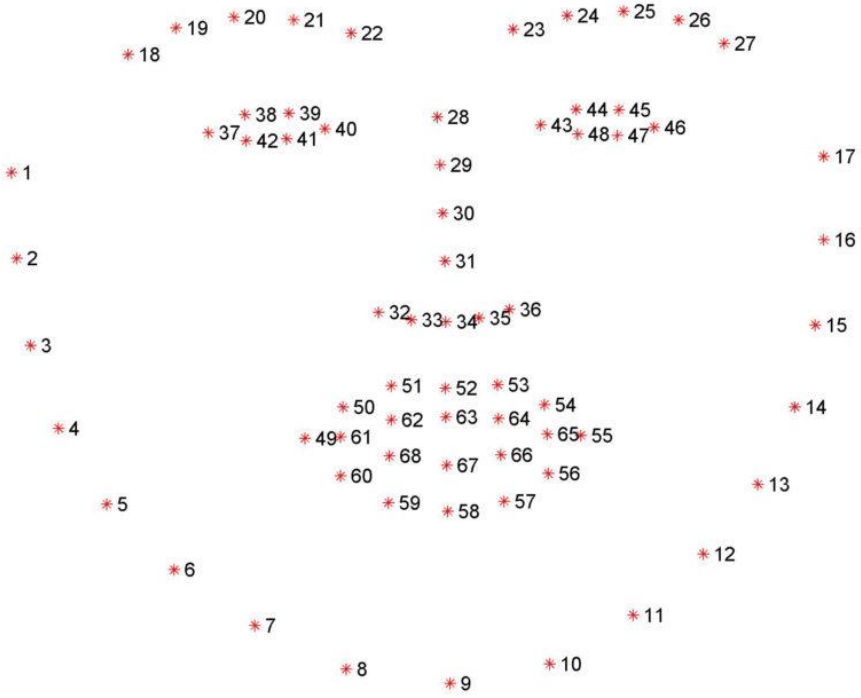


Figure 13 Predefined Facial Landmarks

- After performing the face detection with a pre-trained HOG + linear SVM object detector and facial landmarking on a real time video feed.
- We can further extend it to keep a count of blinks a human performs for the real time video feed.
- Main focus area are EYES markings that can be extracted using [36, 42] and [42, 48] data points.

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

Figure 14 EAR formula

- In terms of blink detection, we are only interested in two sets of facial structures — the eyes.
- 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.

- Relation between the width and the height of these coordinates can further be established.

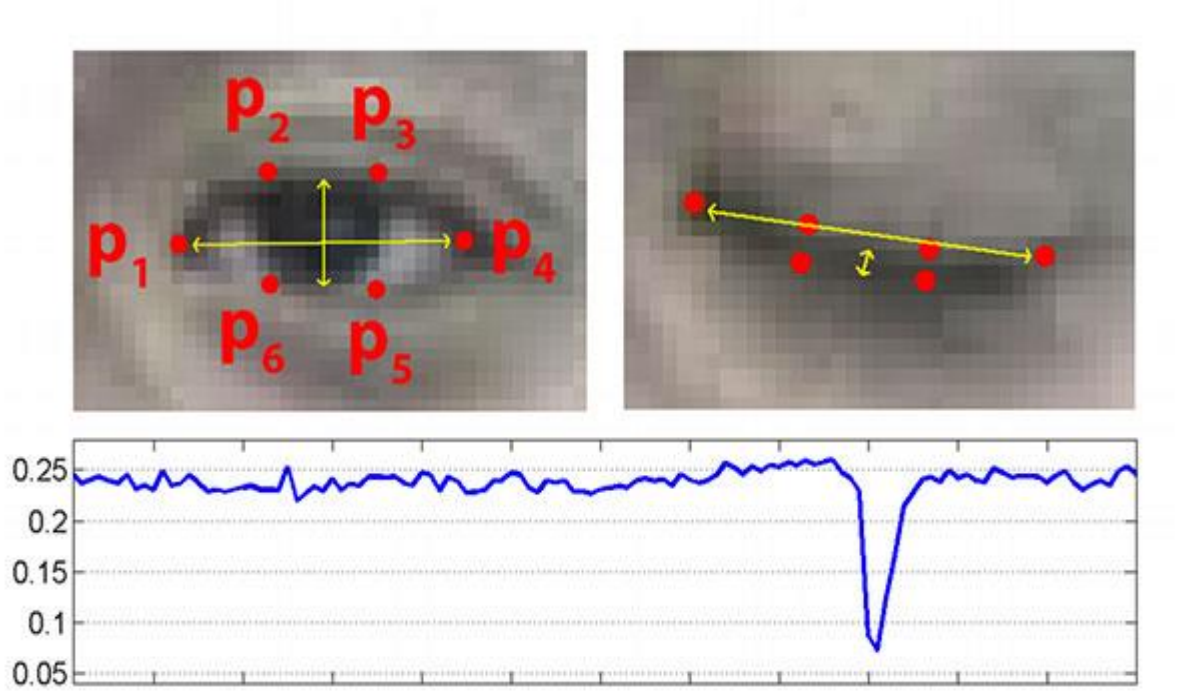


Figure 15 EAR Value for a Natural Eye Blink

- Based on the EAR value analysis we can come up with an EAR value that can be considered for representing an eye blink.
- For our case we have considered a value of 0.26 for our case over 15 consecutive frames to detect an eye blink.

3.5 Electron GUI (Java Script)

Electron is an open-source framework developed and maintained by GitHub. Electron allows for the development of desktop GUI applications using front and back end components originally developed for web applications: Node.js runtime for the backend and Chromium for the frontend.

The front end is written using HTML and CSS. The back end is written using JavaScript.

Following steps are executed by backend:

- GUI executes and makes a call to start execution of python executable file.
- The python executable file set ups a TCP server at 127.0.0.1:4242 and also starts the blink detector window.
- Blink detector broadcasts the state of blink every 0.03s.
- After the GUI starts, by clicking on “start”, the GUI requests the server for state of blink every 0.03ms and generates result depending on the state.

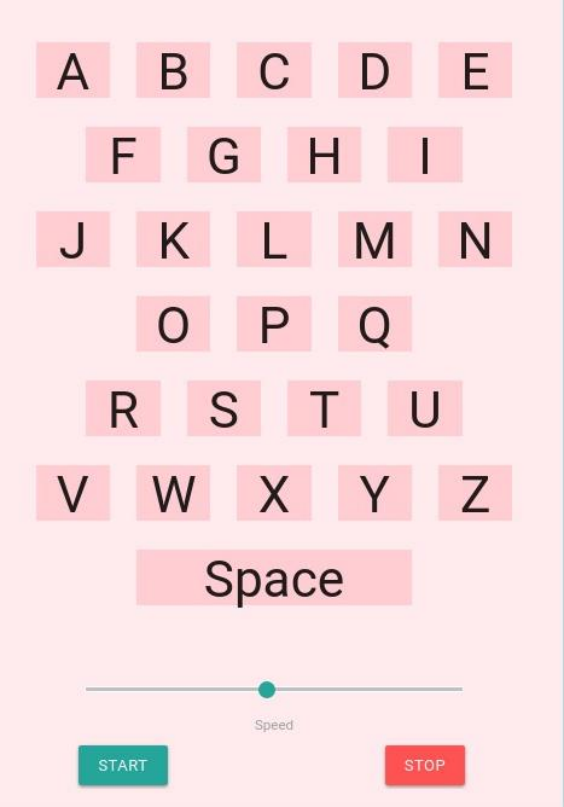


Figure 16 Grid containing letters for typing

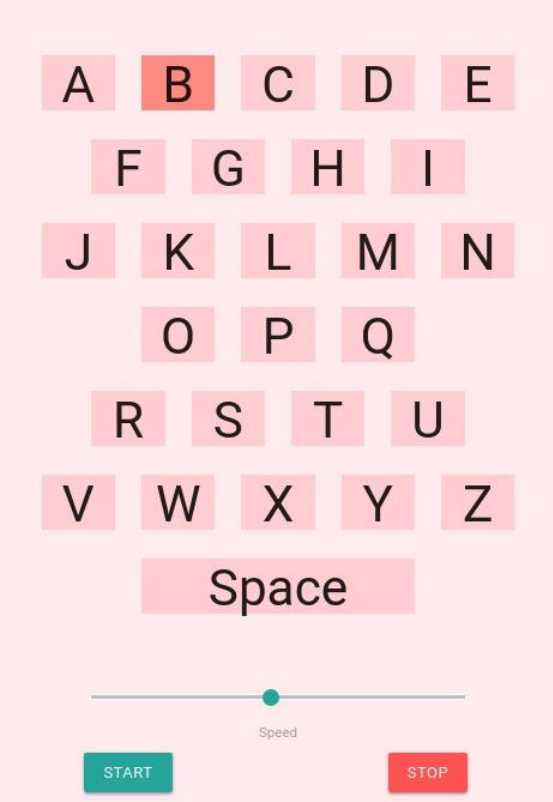


Figure 17 Letter highlighted is the letter to be selected



Figure 18 Selected letters appear in the textbox above

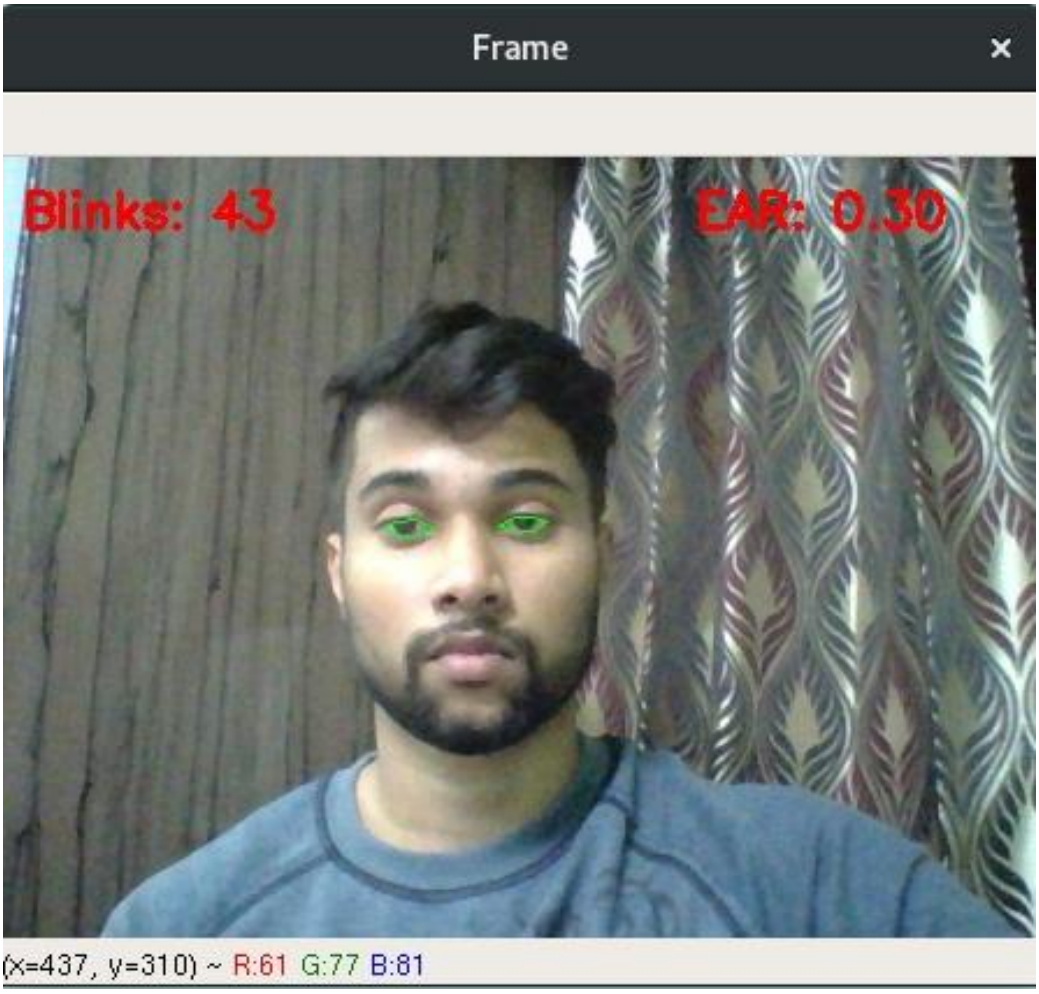


Figure 19 Camera output with eyes marked is shown in a separate window

Chapter 4: Main Implementation and Results

4.1 Research Methodology

Step 1: Planning and designing the application based on the needs of the potential users decided through survey results. Working on the front-end part of the application accordingly.

Step 2: Finding the desired data set for training the model.

Step 3: Implementing different architectures or models mentioned in section for face detection and facial landmarking.

Step 4: Training the models, fine tuning them and making changes in the hyper parameters for bringing state of the art performance on hold out test dataset.

Step 5: The best trained model to be implemented in the backend of the prototype.

Step 6: Additional changes to be made based on the reviews.

4.2 Results and Working of the Application

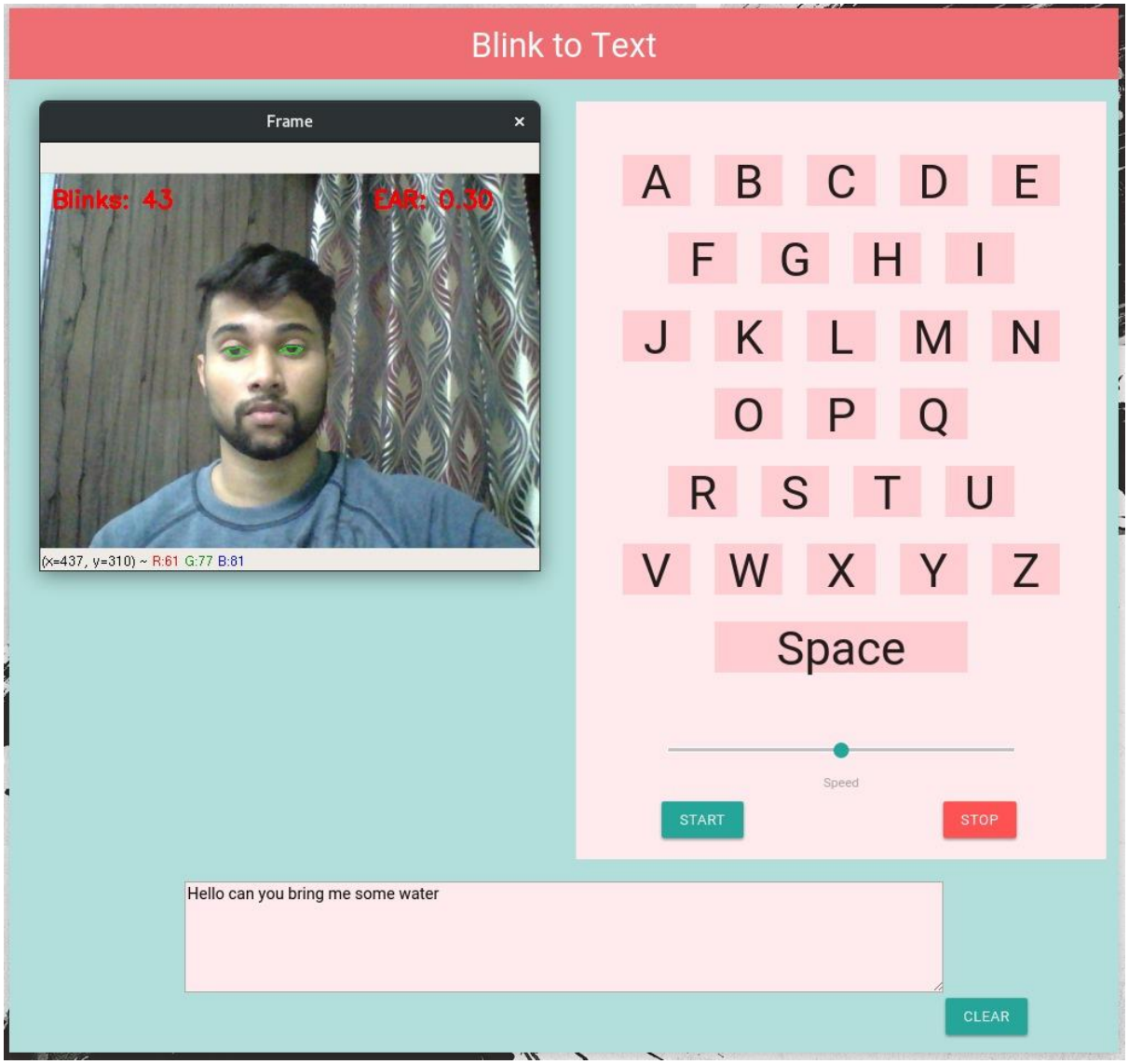


Figure 20 Displaying text "Hello can you bring me some water" on the GUI

4.3 Code Snippets

```
from __future__ import print_function
import sys
import zerorpc
from scipy.spatial import distance as dist
from imutils.video import FileVideoStream
from imutils.video import VideoStream
from imutils import face_utils
import numpy as np
import imutils
import time
import dlib
import cv2
import threading


class Detector(object):
    def blink(self):
        """based on the input text, return the int result"""
        try:
            return blinked
        except Exception as e:
            return 0.0


def parse_port():
    return '4242'


blinked = True


def main():
    t1 = threading.Thread(target=start_detector)
    t1.start()
    addr = 'tcp://127.0.0.1:' + parse_port()
```

Figure 21 Importing necessary python module

```
35      addr = 'tcp://127.0.0.1:' + parse_port()
36      s = zerorpc.Server(Detector())
37      s.bind(addr)
38      print('start running on {}'.format(addr))
39      s.run()
40
41
42  def eye_aspect_ratio(eye):
43      # compute the euclidean distances between the two sets of
44      # vertical eye landmarks (x, y)-coordinates
45      A = dist.euclidean(eye[1], eye[5])
46      B = dist.euclidean(eye[2], eye[4])
47
48      # compute the euclidean distance between the horizontal
49      # eye landmark (x, y)-coordinates
50      C = dist.euclidean(eye[0], eye[3])
51
52      # compute the eye aspect ratio
53      ear = (A + B) / (2.0 * C)
54
55      # return the eye aspect ratio
56      return ear
57
58
59  def start_detector():
60      global blinked
61      print('Detector started')
62      # construct the argument parse and parse the arguments
63      # define two constants, one for the eye aspect ratio to indicate
64      # blink and then a second constant for the number of consecutive
65      # frames the eye must be below the threshold
66      EYE_AR_THRESH = 0.26
67      EYE_AR_CONSEC_FRAMES = 15
68
69      # initialize the frame counters and the total number of blinks
```

Figure 22 Definition of EAR and detector function with threshold value

```
70      COUNTER = 0
71      TOTAL = 0
72
73      # initialize dlib's face detector (HOG-based) and then create
74      # the facial landmark predictor
75      print("[INFO] loading facial landmark predictor...")
76      detector = dlib.get_frontal_face_detector()
77      predictor = dlib.shape_predictor("python_backend/shape_predictor_68_face_landmarks.dat")
78
79      # grab the indexes of the facial landmarks for the left and
80      # right eye, respectively
81      (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
82      (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
83
84      # start the video stream thread
85      print("[INFO] starting video stream thread...")
86      # vs = FileVideoStream(args["video"]).start()
87      fileStream = True
88      vs = VideoStream(src=0).start()
89      # vs = VideoStream(usePiCamera=True).start()
90      fileStream = False
91      time.sleep(1.0)
92
93      # loop over frames from the video stream
94      while True:
95          time.sleep(0.030)
96          # if this is a file video stream, then we need to check if
97          # there any more frames left in the buffer to process
98          if fileStream and not vs.more():
99              break
100
101         # grab the frame from the threaded video file stream, resize
102         # it, and convert it to grayscale
103         # channels)
```

Figure 23 Extraction of eye region from Dlib generated 68-point facial array

```
104         frame = vs.read()
105         frame = imutils.resize(frame, width=450)
106         gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
107 
108         # detect faces in the grayscale frame
109         rects = detector(gray, 0)
110 
111         # loop over the face detections
112         for rect in rects:
113             # determine the facial landmarks for the face region, then
114             # convert the facial landmark (x, y)-coordinates to a NumPy
115             # array
116             shape = predictor(gray, rect)
117             shape = face_utils.shape_to_np(shape)
118 
119             # extract the left and right eye coordinates, then use the
120             # coordinates to compute the eye aspect ratio for both eyes
121             leftEye = shape[lStart:lEnd]
122             rightEye = shape[rStart:rEnd]
123             leftEAR = eye_aspect_ratio(leftEye)
124             rightEAR = eye_aspect_ratio(rightEye)
125 
126             # average the eye aspect ratio together for both eyes
127             ear = (leftEAR + rightEAR) / 2.0
128 
129             # compute the convex hull for the left and right eye, then
130             # visualize each of the eyes
131             leftEyeHull = cv2.convexHull(leftEye)
132             rightEyeHull = cv2.convexHull(rightEye)
133             cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
134             cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
135 
136             # check to see if the eye aspect ratio is below the blink
137             # threshold, and if so, increment the blink frame counter
138             if ear < EYE_AR_THRESH:
```

Figure 24 Conversion from BGR color to grey color scheme

```
            COUNTER += 1
            blinked = False
            print('Not Blinked')

        # otherwise, the eye aspect ratio is not below the blink
        # threshold
        else:
            # if the eyes were closed for a sufficient number of
            # then increment the total number of blinks
            if COUNTER >= EYE_AR_CONSEC_FRAMES:
                TOTAL += 1
                blinked = True
                print('Blinked')
            else:
                blinked = False
                print('Not Blinked')

            # reset the eye frame counter
            COUNTER = 0

        # draw the total number of blinks on the frame along with
        # the computed eye aspect ratio for the frame
        cv2.putText(frame, "Blinks: {}".format(TOTAL), (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # show the frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break
```

Figure 25 Detection of blink status

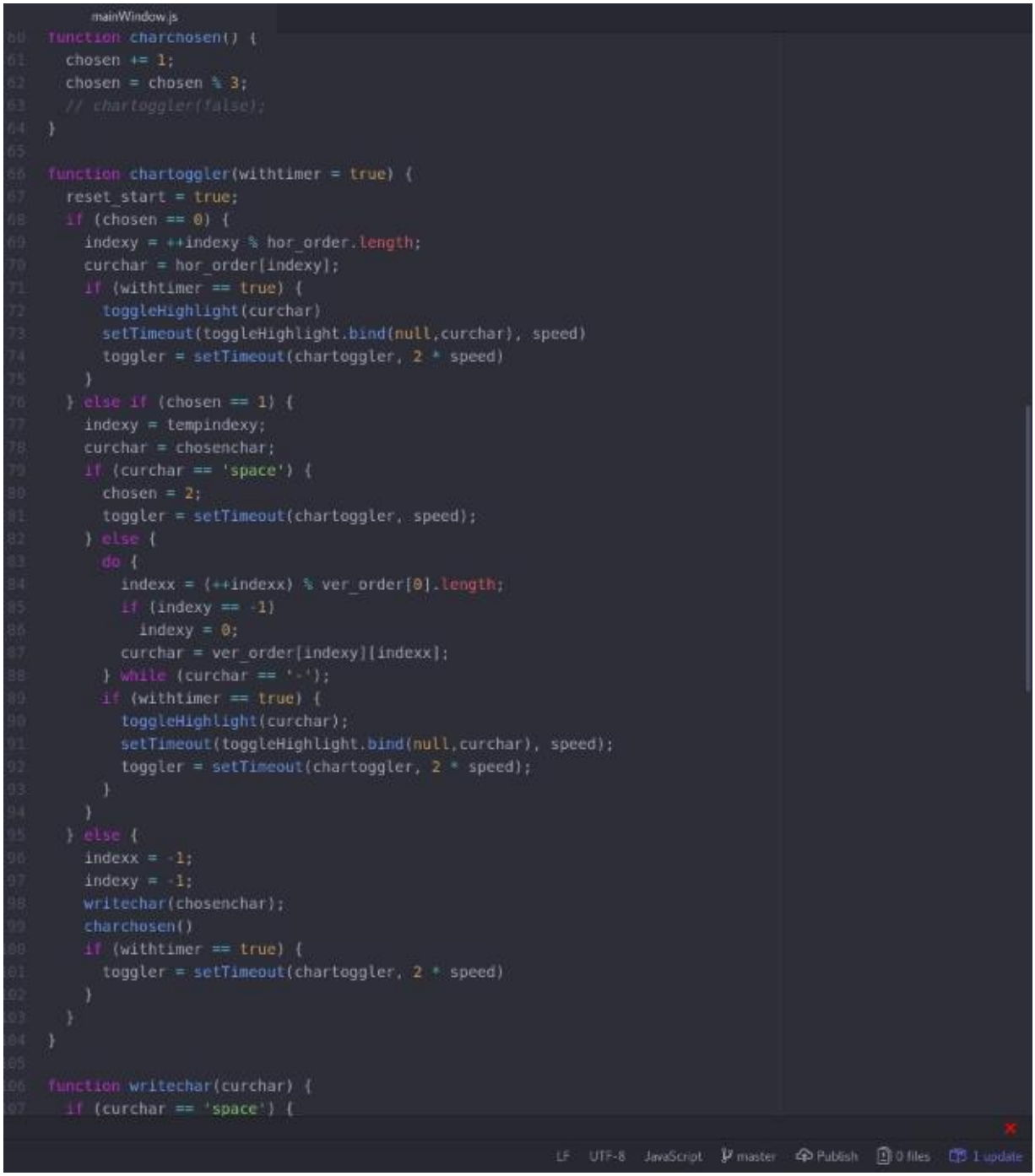


Figure 26 Toggling Letters in Grid



Figure 27 Autocorrect



Figure 28 Detecting State of Blink

Chapter 5: Conclusion and Future Scope

BlinkToText provides a viable option for LIS patients to communicate. The software's accessibility first design LIS patients to express themselves independently. The software being opensource and requiring no high-end hardware ensures that it is widely available across financial classes.

Of course, there is still room for improvement in future iterations of the software. In short term development, the face detection algorithm should be made more robust. The software should migrate to this system. In long term development, more activities should be made within the reach of an eye blink. If a person cannot transverse physical space, she/he should still be able to explore cyberspace. The idea of a web browser controlled completely by eye movement intriguing, imperative, and the next logical step for this project.

REFERENCES

[1] Kaiqi Cen, “Study of viola jones real time face detector”, 2016

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, 2001, pp. I-I.

doi: 10.1109/CVPR.2001.990517

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1.

doi: 10.1109/CVPR.2005.177

[4] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, M. Pantic. “300 Faces in-the-Wild Challenge: The first facial landmark localization Challenge.” Proceedings of IEEE Int’l Conf. on Computer Vision (ICCV-W), 300 Faces in-the-Wild Challenge (300-W). Sydney, Australia, December 2013.

[5] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1867-1874.

doi: 10.1109/CVPR.2014.241

[6] T. Soukupová and J. Cech. “Real-Time Eye Blink Detection using Facial Landmarks.” ,2016.