Department of Electrical Power Engineering and Mechatronics

Development of tiredness assessing system based on webcam image processing and behavioral biometrics

Pilditöötlusel ja biomeetrilistel andmetel põhinev väsimuse hindamise süsteem

MASTER THESIS

Student: Sourabh Aryabhatta
Student ID: 165547MAHM
Supervisor: Dmitry Shvarts, PhD

Tallinn, 2019

Author's Declaration

Hereby I declare, that I have written this thesis independently.

No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." 2019

Author: .....
/signature/

Thesis is in accordance with terms and requirements

"....." 2019

Supervisor: .....
/signature/

Accepted for defense

"....." 2019

Chairman of theses defense commission: .....
/name and signature/

Department of Electrical Power Engineering and Mechatronics

THESIS TASK

Student:	Sourabh Aryabhatta, 165547MAHM
Study programme, main specialty:	MAHM02/13 - Mechatronics
Supervisor:	Dmitry Shvarts, PhD

Thesis Topic:

(in English)	Development of tiredness assessing system based on webcam image processing and behavioral biometrics.
(in Estonian)	Pilditöötlusel ja biomeetrilistel andmetel põhinev väsimuse hindamise süsteem.

Thesis main objectives:

- Design a vision system to identify small variation of face components using image sensor.
- Develop an algorithm to assess tiredness level based on designed vision system.
- Train a ML model to utilize keystroke dynamics when tiredness is identified.

Thesis tasks and time schedule:

Nr.	Description of task	Completion date
1.	Literature study of related works on tiredness assessment	15.08.2019
2.	Formulation of first concept with required parameters, algorithm design; selection of camera sensor	30.08.2019
3.	Target platform, programming language and library selection; Interface design of prototype application	15.09.2019
4.	Development of prototype application to detect facial components and monitor keystroke dynamics for ML model training	30.09.2019
5.	Continuous development of prototype application to measure small variation of facial components and determine tiredness level	15.10.2019
6.	Test and optimization of prototype application	30.10.2019
7.	Conclusion and area of improvement for future work	08.11.2019
8.	Submission of first draft (soft copy)	15.11.2019
9.	Submission of final draft based on technical feedback	22.11.2019
10.	Compilation of thesis for final submission	29.11.2019

Engineering & economic problems to be solved: Employee drowsiness or sleepiness is a major contributor of low productivity and poor work-performance. Thus it affects the quality of deliverable products. This paper aims to design a cost-effective vision-based sleepiness detection system that can help to improve the situation effectively.

Language of thesis: English
Deadline for submission of application in OIS for defense: 16.12.2019
Deadline for submission of thesis (hard copy): 03.01.2020
Student: Sourabh Aryabhatta
Supervisor: Dmitry Shvarts, PhD

List of symbols

α_t	Amount of say
b	Bounding box
B	Total number of bounding boxes in a grid
C	Class probability
$d(p_1, p_2)$	Euclidean distance between two points p_1 and p_2
D	Distribution
D_t	Distribution at time t
ϵ	Error
ϵ_t	Error at time t
f	Percentage eye openness tracking (PERCLOS)
F_N	N-th Haar-like feature
$F(x, y)$	Two-dimensional function representing digital image
b_h	Height of the bounding box b
$h(x)$	Weak classifier of object $x \in X$
$H(x)$	Strong classifier of object $x \in X$
$i(x, y)$	Intensity of pixel (x, y)
IOU	Intersection over union
Pr	Probability
$s(x, y)$	Integral image at pixel (x, y)
t	Time
T_c	Aggregated duration of eye closure
T_t	Total time of experiment
(b_x, b_y)	Center of bounding box b
b_w	Width of the bounding box b
X	Set of training objects
Z_t	Normalization constant

List of abbreviations

2D	2-dimensional
3D	3-dimensional
AAM	Active appearance model
AdaBoost	Adaptive boosting
ADC	Analog to digital converter
AI	Artificial intelligence
ANN	Artificial neural network
API	Application programming interface
BGR	Blue green red
CCD	Charge-coupled device
CMOS	Complementary metal-oxide-semiconductor
CNN	Convolutional neural network
CUDA	Compute unified device architecture
DIP	Digital image processing
DL	Deep-learning
DOF	Degrees of freedom
EAR	Eye aspect ratio
EEG	Electroencephalography
EMG	Electromyogram
EmguCV	Cross platform .NET wrapper of OpenCV library
EOG	Electro-oculography
fEMG	Facial electromyography
FER	Face expression recognition
FL	Focal length
FNR	False negative rate
FOQA	Flight operational quality assurance
FOV	Field of view
FN	False negative
FP	False positive
FPR	False positive rate
FPS	Frame per second
FSS	Fatigue severity scale
GB	Gigabyte
GHz	Giga-Hertz
GPU	Graphics processing unit
HD	High definition
ICA	Independent component analysis
IDE	Integrated development environment

IR	Infrared
IT	Information technology
KD	Keystroke dynamics
KSS	Karolinska sleepiness scale
LBF	Local binary features
LBP	Local binary patterns
LDA	Linear discriminant analysis
MAR	Mouth aspect ratio
MD	Mouse dynamics
ML	Machine-learning
MMG	Mechanomyogram
NASA	TLX NASA task load index
OOP	Object-oriented programming
OpenCV	Open source computer vision library for C and C++
OS	Operating system
PCA	Principal component analysis
PERCLOS	Percentage eye openness tracking
PIXEL	Picture element
PVT	Psychomotor vigilance task
RAM	Random-access memory
R&D	Research and development
ROI	Region of interest
SD	Sensor dimension
SVM	Support-vector machine
TLX	Task load index
TN	True negative
TP	True positive
USB	Universal serial bus
VAS	Visual analogue scale
WD	Working distance
WFA	Windows form application
YOLO	You only look once

Foreword

In English

This thesis paper proposes a non-invasive and non-intrusive system that evaluates an office employee's tiredness level. This research work is dedicated to the field of Machine Vision and Artificial Intelligence.

First of all, I would like to thank my supervisor Dmitry Shvarts for his constant supervision and helpful directions. I express my heart-felt gratitude to my parents and my wife for their continuous support. Big thanks goes to Ingel-Marie Hiiekivi, who kindly translated the foreword and conclusion for me.

This research paper is done as a part of *Graduation Thesis*, under the Department of Electrical Power Engineering and Mechatronics, Taltech University. If this paper or its LaTeX source code is needed for some reference, please request permission from the Department of Electrical Power Engineering and Mechatronics, Taltech University.

In Estonian

Käesolev diplomitöö pakub välja mitteinvasiivse ja -intrusiivse süsteemi, mis hindab kontoritöötaja väsimustaset. See lõputöö on pühendatud masinnägemise ja tehisintellekti valdkondadele.

Esiteks soovin tänada oma juhendajat Dmitry Shvarts pideva juhendamise ja abistavate juhiste eest. Avaldan südamest tänu pideva toetuse eest oma vanematele ja oma abikaasale. Suur tänu Ingel-Marie Hiiekivile, kes oli abiks eessõna ja kokkuvõtte tõlkimisel.

Antud uurimustöö on tehtud lõputöö osana Taltechi ülikooli Elektroenergeetika ja mehhatroonika instituudi osakonnas. Kui antud tööle või tema LaTeX-i lähtekoodile on vaja viidata, palun pöörduge Taltechi ülikooli Elektroenergeetika ja mehhatroonika instituudi poole.

Contents

Chapter 1

Introduction

According to a public survey (2015-2016) in Estonia among office workers, it is found that sitting at desk in front of computer for long hours is the primary contributor of occupational tiredness. The survey was conducted among more than 150 creative R&D employees (including applied researchers, engineers, IT and product developers) and it was carried out by a group of medical researchers and economists from Department of Economics and Finance of Taltech University [1].

Occupational tiredness often leads to sleepiness; thus it impacts worker's performance and productivity very badly. As a result, the quality of products degrades significantly and it ends up with heavy loss of profit, time and reputation. In addition, sleepiness hampers workplace safety and might drive to industrial accidents. To prevent this situation, it is important to monitor the tiredness level of a desk employee consistently. This paper intends to design a vision-based alert system that can assess employee tiredness (sleepiness) in real time.

A number of works related to tiredness detection (using contact-based sensors) have been done in controlled laboratory environment. But unfortunately their application in real life environment is still very limited, because users don't feel comfortable to have invasive measurements. Again, scientists found that fatigue level experienced in laboratory environment varies significantly with the actual fatigue in daily life [2]. Hence, this paper makes an effort to design a cost-effective vision-based tiredness detection system that can be implemented in real workplace.

1.1 Motivation

The primary motivation of this research work is to bring more intelligence to machine vision system and analyze the observed small variation of physiological change (e.g, sleepy eye, yawn etc.) in order to evaluate occupational tiredness of an individual. Although similar topics (in other fields) have been discussed by experts in recent years, but unfortunately there is no such implementation in the official work environment.

The current beneficiary of the system will be desk-employees who spend a lot of time sitting consistently in front of their computers. But other kinds of computer users, for example, students can use this system. Again, the system can be implemented in various cases including computer control through individual's motion and facial expression. So, applicability, reliability and robustness are the key factors that need to be considered.

1.2 Task Objective

This paper aims to develop a unique methodology and respective system that measures small variation of physiological activity of a computer user in real time through webcam image processing. With the help of designed algorithms, it will assess the level of tiredness (sleepiness) of that individual. It is expected that necessary features (e.g, charts, graphs etc.) will be implemented in the system for further analysis of the obtained results. In addition, if the individual is identified as tired, it will store keystroke dynamics data of the person into a database in order to train a ML model for future usage. Please note that, the system will be fully non-invasive and cost-effective too.

Hence a series of verification tests will be done in order to inspect the system's applicability and reliability. A confusion matrix will be used in order to study the system's performance (accuracy and precision). Finally, a comparative study with other existing methods will be done in order to evaluate whether the system can be implementable in real life environment or not.

1.3 Literature Review and Background

There exist many internal and external factors that can modulate the onset and presence of tiredness. These factors include sleep deprivation, naps, noise, heat, mood, motivation, time of day, workload and most essentially the individual's profile (age, gender, professional occupation, consumption of alcohol and drugs etc.) [3]. As a result, several visual analogue scales (**VAS**) were introduced to assess these factors in order to determine the level of fatigue of employees (e.g, pilots, flight-crew members etc.). Table A.1 presents a short description of these analogue tools. Again, fatigue can be defined objectively by measuring degraded performance using another set of different tools. Short description of these tools is presented in Table A.2.

Since there exist strong possibilities to cheat the subjective assessment tools, because one can manipulate the result easily by providing incorrect answers; so it was obvious to introduce technology to detect the level of tiredness. Most of these technologies use contact-based sensors such as force gauge, mechanomyogram (**MMG**) and electromyogram (**EMG**). Among these systems, force gauge is considered to be minimal invasive; but it requires a hand grip dynamometer. On the other hand, MMG uses an accelerometer or goniometer that needs direct skin contact and it is sensitive to noise as well. Again, EMG requires electrodes that need to wear adhesive gel patches [4]. However, real-life application of these systems is still very limited, because people do not prefer to interact with such invasive and intrusive systems.

With the growth of modern image-processing technology, many algorithms have been established to detect real-time fatigue in drivers and pilots in recent years. In 2007, Volvo introduced the world's first driver tiredness detection system (Driver Alert Control) based on car's movement. Other popular car manufacturing companies invested a lot of money to develop vision system that can identify driver drowsiness as well. In 2015, an algorithm for detecting visual fatigue by analyzing the blink rate using computer vision was introduced by Clavijo and others [5]. In the same year, Abdulin and Komogortsev established another novel technique to detect user eye fatigue via eye movement behavior [6]. Few real-time sleepiness detection algorithms based on driver's eye-state (open or closed) were proposed by Ghimire and others in 2016 [7, 8]. On the other hand, Zhang and Hua showed that facial expression analysis using local binary patterns can be used to recognize tiredness as well [9].

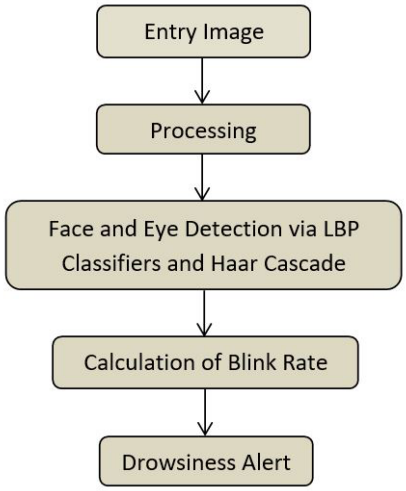


Figure 1.1: Flowchart of visual fatigue detection by analyzing blink rate [5]

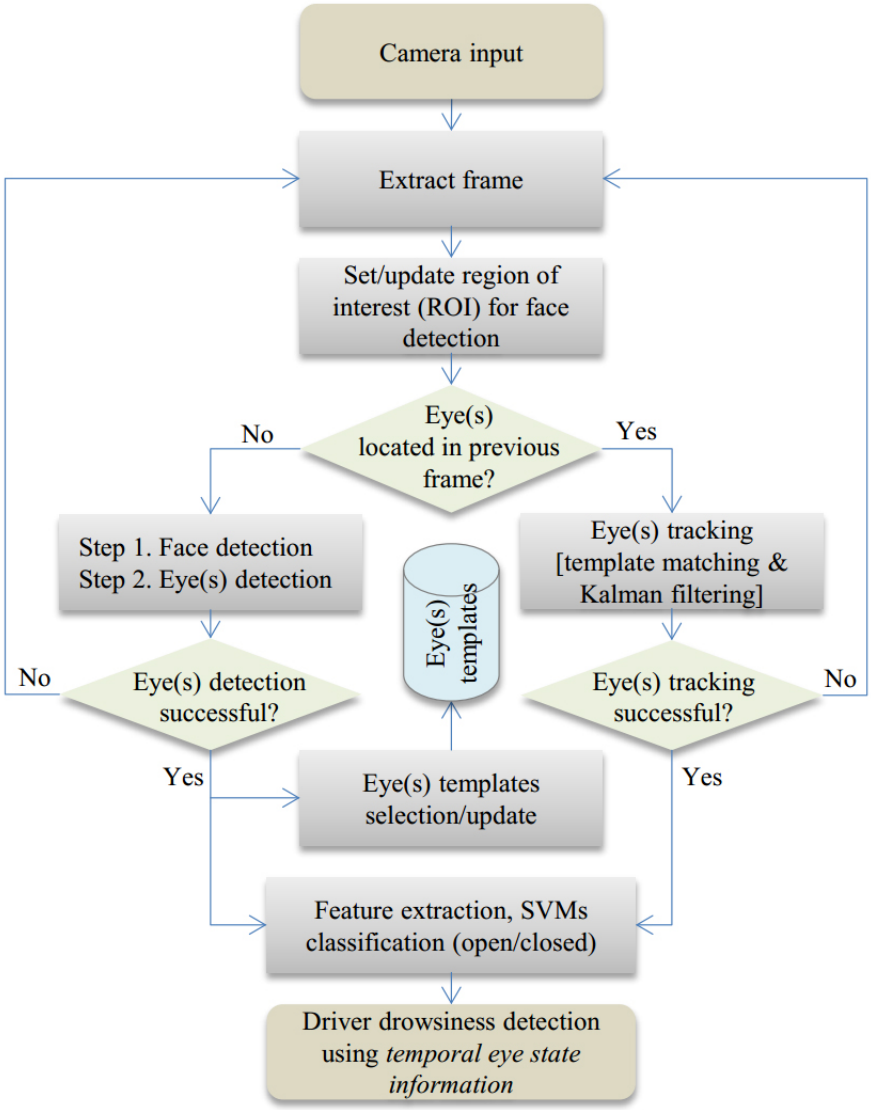


Figure 1.2: Flowchart of the driver's drowsiness detection system [7]

1.4 Explanation for Developing Current Application

We already know that smart cars (installed with smart vision system) are able to trigger alarm while the driver gets sleepy. However, the facility is still limited to those cars and the system is quite costly as well, because it uses very expensive infrared camera for vision system so that it can operate in extremely low light condition. Most importantly, the technology is a business-secret as well.

Although there exist a few standalone vision-based fatigue detection systems, but they can deliver acceptable results only in laboratory environment. Because it is evident to have various potential challenges in their application level such as installation in real life environment, cost of installation, necessity of external dependencies (e.g, lighting), maintenance support etc. So there exist tons of opportunities to research in this field and introduce vision-based monitoring system to detect occupational tiredness.

In order to develop such system, our idea is to identify very small variation of physiological change using vision system, apply efficient algorithm to analyze the change and detect tiredness in real time. At the same time, the system should be able to collect the individual's biometric data (keystroke dynamics) too, so that it could be used for training a ML model and later the model can predict tiredness of that person with some confidence level as well. In principle, we require to handle two processes, which are as follows:

1. Facial expression recognition
2. Behavioral biometrics acquisition

1.4.1 Facial Expression Recognition

Automatic facial expression recognition (**FER**) has become a challenging area in the field of computer vision and one of its popular applications is operator tiredness detection. Figure 1.3 shows a standard FER block diagram [10].

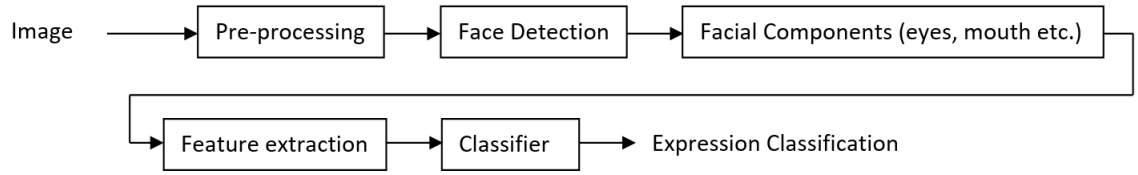


Figure 1.3: Facial expression classification block diagram [10]

In pre-processing phase, noise-removal is done by considering a time series of images (from neutral to an expression) as input and gives the face region as output. Then region of interest (**ROI**) for eyes, nose, cheeks, mouth are detected during the facial component detection phase. Feature extraction phase is responsible for extracting features from the selected ROIs. Popular feature extraction techniques include Gabor filters [11], Local Binary Patterns (**LBP**) [12], Principal Component Analysis (**PCA**) [13], Independent Component Analysis (**ICA**) [14], Linear Discriminant Analysis (**LDA**) [15] etc. Figure 1.4 shows example of some of these feature extraction techniques applied on an input image.

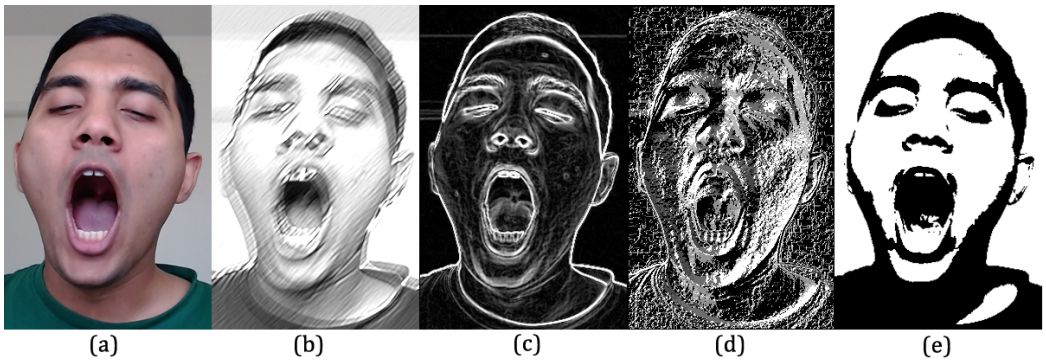


Figure 1.4: Example of popular feature extraction techniques – (a) Input image (yawning person), (b) Gabor filter, (c) Sobel edge detector, (d) LBP, (e) Threshold binarization

1.4.2 Behavioral Biometrics Acquisition

In 2018, Ulinskas and others found that keystroke dynamics is one of the significant biometric characteristics that can define the tiredness level of office workers up to some extent [16]. In the context of our application we are interested to analyze some of the keystroke attributes, because, for example, a tired person's keystroke data will be different from normal condition over the period of interest. If we can collect & analyze these data over a period of time, we can extract a probability-driven result with less margin of error.

1.4.3 Justification

Although many researchers have been trying to develop a system in order to assess fatigue level of desk employees; but our approach is unique from all of them. Since, we do not use any contact-based sensor, so our developed system will be fully non-invasive and non-intrusive. Again, our development cost will be cheapest among all the other existing systems, because only one image sensor will be mounted. From development perspective, we want to use **OpenCV** and **EmguCV** library functions for facial component recognition; both are open source libraries, along with **Microsoft Visual Studio 2019** as IDE and **Microsoft Access 2016** as database to store keystroke dynamics data for future testing. Most importantly, migration to mobile & other platforms and feature extension should not be difficult as well.

1.5 Thesis Structure

The rest of this paper is organized as follows. Chapter 2 discusses about different measurement techniques to detect **Tiredness**. Chapter 3 describes about the application of **Computer Vision**. Chapter 4 describes about the application of **Behavioral Biometrics**. Chapter 5 explains the **Proposed Methodology** to identify individual tiredness and respective algorithms. Chapter 6 discusses about the **Implementation** of the system in terms of hardware and software. Chapter 7 presents the **Results** obtained from the developed prototype application. Chapter 8 provides a detailed **Discussion** on system's performance analysis and comparative study with other similar systems. Finally, Chapter 9 wraps up with **Conclusion** with future directions.

Chapter 2

Tiredness

In this chapter we explain the term "Tiredness" and describe possible ways of **tiredness measurements** in details. In general, tiredness refers to a state of human body with lack of energy and motivation compared to regular time. It is divided into two types − physical and mental. Physical tiredness is known as a temporary physical inability of our muscles to perform optimally due to lack of strength, or muscle weakness. For example, daily activities like doing physical exercise, climbing stairs or carrying shopping bag etc. might seem much harder than before. On the other hand, mental tiredness is a temporary inability to keep up optimal cognitive performance. Less concentration in work, unwilling to get up in the morning etc. are common examples of mental tiredness. It frequently appears together with physical tiredness, but not always. Both physical and mental tiredness are dangerous, especially when the person is a health-care professional or driver or pilot or military personnel, because a little mistake can lead to loss of many lives. Figure 2.1 shows an example of visual tiredness in our everyday life.

Figure 2.1: Visual tiredness [17]

Now-a-days it is highly important to monitor tiredness of office workers as far as the productivity and quality is concerned. Employees working long time with computer have quite often problems with eyes, neck and spine muscles. The reason is watching screen and sitting in one position for long time. As a result, personal health problems (like vision related problem, back-pain, or even coronary diseases) initiate and productivity decreases significantly. Periodic rest during the work improves the situation considerably, and in some cases, it helps to avoid severe health problems completely.

In order to identify individual's tiredness level, we have tried to classify the possible measurements into following categories:

1. Contact-based measurements
2. Time-based measurements
3. Behavioral measurements

In the following sections, we will discuss about these measurements in details.

2.1 Contact-based Measurements

With the growth of modern technology, it is even possible to track very small change on facial muscles with the help of contact-based sensors and thus it helps to identify tiredness as well. Facial electromyography (**fEMG**), electrooculography (**EOG**), electroencephalography (**EEG**) are good examples of such measurement techniques.

Generally, fEMG uses surface EMG electrodes to detect activity at the *zygomaticus* (cheek), *orbicularis oculi* (under eye), and/or *corrugator supercilii* (brow) muscle regions. Once the setup is ready, the candidate is exposed to a stimulus signal, e.g, sound, picture, or smell for psychophysiology or neuromarketing studies [18]. Figure 2.2 shows a typical fEMG carried out for target muscle regions.

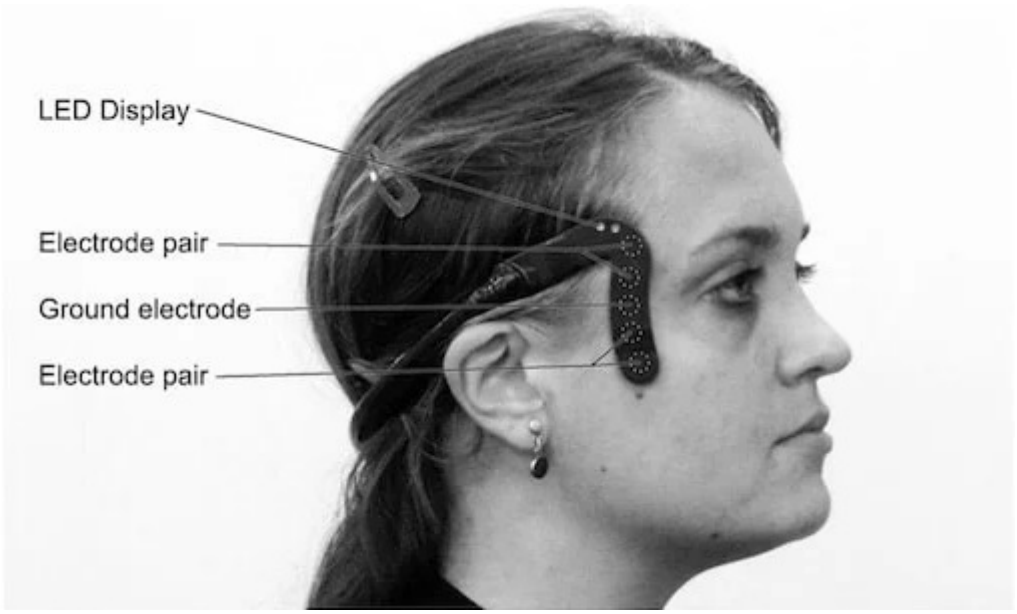


Figure 2.2: Facial electromyography (fEMG) [19]

Again, EOG measures eye movement using pairs of electrodes that need to be placed either above and below the eye or to the left and right of the eye. On the other hand, EEG measures an individual's ability to resist sleep by tracking his/her brain's electrical activity using small, metal discs (electrodes) attached to scalp. Again, it is important to note that, all of the above mentioned techniques involve contact-based electrodes.

2.2 Time-based Measurements

The most common visual symptoms of tiredness of desk-workers are as follows:

1. Eye flickering (closing of the eyelid due to sleepiness)
2. Yawn (reflex that consists of the simultaneous inhalation of air)
3. Nod (quick downward motion of the head)

While we speak about eye-flickering, it is found that percentage eye openness tracking (**PERCLOS**) is one of the best quantitative measures for fatigue detection. It can be estimated from continuous video stream of the eye images. It is defined as the proportion of time the eyelids are at least 80% closed over the pupil. So it is described as a temporal ratio over a given time frame and measured in percentage [20]. Mathematically we can describe PERCLOS by the following equation:

$$f = \frac{T_c}{T_t} * 100\% \tag{2.1}$$

In equation 2.1, f is the PERCLOS value, T_c is the aggregated duration of eye closure and T_t is the total time of experiment. Figure 2.3 illustrates the measurement of PERCLOS in details, followed by necessary mathematical expressions.

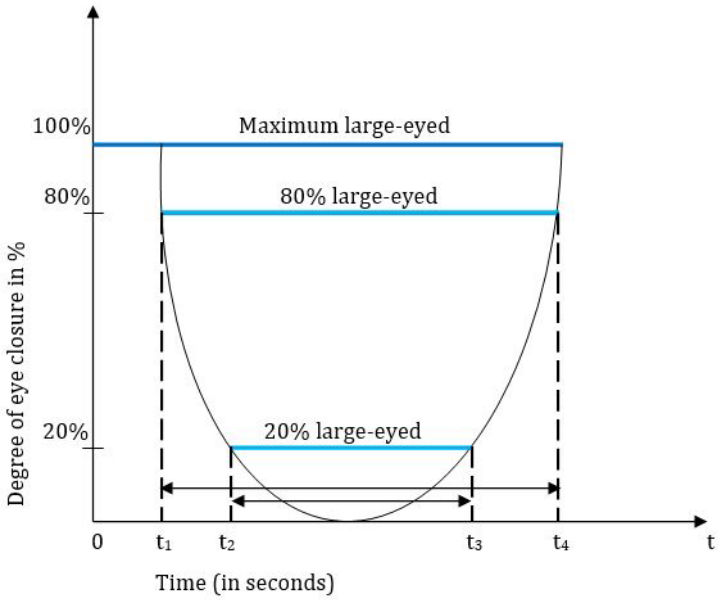


Figure 2.3: Percentage eye openness tracking (**PERCLOS**)

According to figure 2.3, total time of experiment is the time in between t_1 and t_4. Again, the time in between t_2 and t_3 is the time when the eyes are closed at least 80% or even more. So we can measure PERCLOS value by rewriting equation 2.1 as follows:

$$f = \frac{t_3 - t_2}{t_4 - t_1} * 100\% \tag{2.2}$$

Again, the frequency of yawn, downward motion of head etc. are considered as time-based measurements for tiredness detection.

2.3 Behavioral Measurements

Behavioral biometric data is widely used for personality validation, because it makes each person unique in terms of his/her behavioral characteristics. Generally, behavioral characteristics include keystroke dynamics, gait, voice, personal signature. Keystroke dynamics is also known as keystroke biometrics or typing biometrics. It refers to the detailed timing data which tells exactly when each key was pressed and released by a computer user. In recent years, keystroke features have been successfully used for automated stress detection [16].

Undoubtedly the most useful advantages of utilizing keystroke dynamics are its universality and collectability, because almost everyone has access to keyboard of a computer. And at the same time, **no special hardware is required** to capture this biometric data. As a result, we do not need any additional driver to be installed. Only system libraries are sufficient to capture the timing information.

On the other hand, a major drawback of keystroke dynamics is that, this method uses confidence measurement, instead of the classical pass-fail measurement. Thus it makes false positive rate (**FPR**) and false negative rates (**FNR**) to be high - which unexpectedly makes the result less reliable than physiological biometrics. But one of the primary contributors of this high FPR and FNR is individual's tiredness level. So, eventually it helps to determine an individual's tiredness level with some approximation. Similarly, we can try to utilize mouse dynamics information for tiredness detection if we can get sufficient time.

2.4 Summary

In this chapter, we have discussed about different measurement techniques to detect individual's tiredness. Since we are interested to develop a non-invasive and non-intrusive system, so we decline the possibility of using contact-based sensors. Now we would like to focus on the following time-based measurements:

- Status of eyes (e.g, duration and frequency of eye-closeness)
- Status of mouth (e.g, duration and frequency of yawns)
- Nodding (e.g, motion of visible facial components, head etc.)

So, we need an image sensor (built-in or externally-mounted HD webcam) for our vision system. The system is expected to monitor facial components of a desk employee as it's region of interest (**ROI**) in real-time. Next, image processing algorithm will help us to measure, calculate and analyze the information in order to detect tiredness level of that person. In parallel threads, necessary keystroke dynamics will be monitored as a part of behavioral measurements. So, these are our necessary parameters for formulating first concept of our project. With the help of all these parameters, we should be able to determine tiredness level (possible symptoms of sleepiness) of an individual in real time.

Chapter 3

Computer Vision

In this chapter, we discuss some basic topics of computer vision that will be used throughout the rest of the thesis. In brief, the study of computer vision is concerned to extract information from images and utilize them for intended application. More detailed definition of computer vision is discussed in section A.2.1. In section 3.1, we explore different image sensors and their functionality so that we can choose the proper sensor for our designed system.

3.1 Image Sensors

3.1.1 Charge-coupled Device (CCD)

CCD consists of millions of pixels. When these pixels are exposed to incoming light, they convert the light into charge and the charge gets accumulated in these pixels. Once the charge is collected by these pixels, this charge is transferred using the horizontal shift registers. This charge is transferred to the last column's vertical shift registers. Then each charge is converted into voltage. After voltage conversion, each voltage is amplified using the amplifier. Once the vertical shift register is emptied, the same procedure is followed by the remaining charges. Once all the voltage amplification is completed, the output signal is converted into digital signal using ADC (Analog to Digital Converter). Figure 3.1 illustrates a simple overview of how CCD sensor works [21].

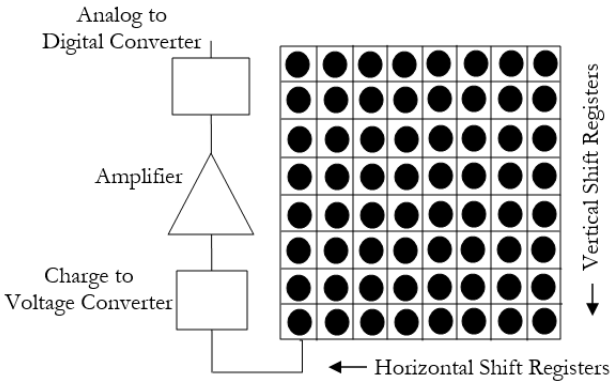


Figure 3.1: CCD sensor

3.1.2 Complementary metal-oxide-semiconductor (CMOS) Sensor

In CMOS sensor, the fabrication technology is very similar to the fabrication technology of the integrated circuits. Because of that, many peripheral circuits can be integrated to the single chip. So in case of the CMOS sensor, the charge to voltage conversion, as well as the voltage-amplification is carried out in the pixel itself. As the charge-to-voltage conversion & voltage-amplification carried out in the pixel itself, so the processing speed is much higher than the processing speed of CCD sensors. So, in case of CMOS sensors, the voltage generated by each pixel is being read in a line-by-line fashion. First of all, the first row of the pixels is activated using the pixel select switch. This pixel select switch connects this output voltage of this pixel to the column line. By activating the column select switch one-by-one, we can read the data of each pixel. And the same procedure is repeated for the remaining lines. In this way, in CMOS sensor, data is read in a line-by-line fashion. Figure 3.2 illustrates a simple overview of how CMOS sensor works [21].

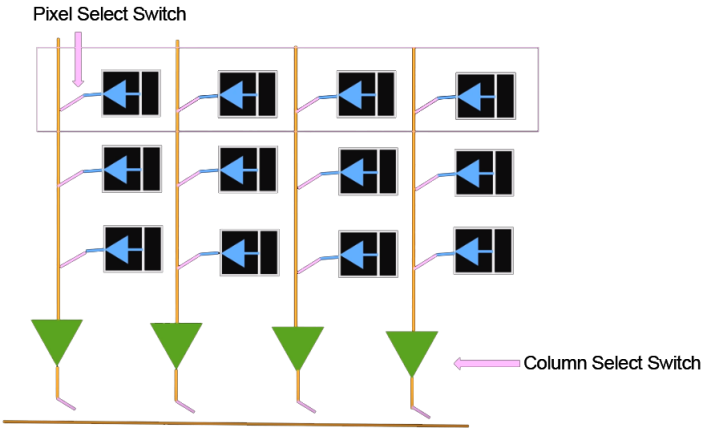


Figure 3.2: CMOS sensor [21]

3.1.3 Comparison between CCD and CMOS Sensor

Feature	**CCD**	**CMOS Sensor**
System Integration	Not possible	Possible
Power Consumption	High	Low
Processing Speed	Less	High
Image distortion	Blooming effect	Rolling shutter effect
Noise	Low	High
Sensitivity	High	Low

Table 3.1: Comparison between CCD and CMOS Sensor

A detailed comparison between CCD and CMOS sensor is described in section A.2.3.

In a summary, when we emphasize on high dynamic range and low noise, we consider CCD sensor. On the other hand, when we put more importance on fast-processing-speed and as well as low-power-consumption, CMOS sensor is preferred over the CCD sensor. Since we are concerned about the processing speed, so we would prefer to select **CMOS sensor** for our designed system. In section 3.2, we will discuss about **digital image processing** in details.

3.2 Digital Image Processing

3.2.1 What is a Digital Image?

A digital image is a numeric representation of a two-dimensional image. These images have a finite set of digital values, called pixels. The digital image contains a fixed number of rows and columns of pixels. Pixels are the smallest individual element in an image, holding quantized values that represent the brightness of a given color at any specific point [22].

Mathematically, a digital image can be defined as a two-dimensional function, $F(x, y)$, where x and y are spatial coordinates, and the amplitude of F at any pair of coordinates (x, y) is called the intensity of that image at that point [23].

3.2.2 Digital Image as a Matrix

Digital images can be represented as a matrix, like the following [23]:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & f(0,2) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & f(1,2) & \dots & f(1,N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & f(M-1,2) & \dots & f(M-1,N-1) \end{bmatrix}$$

Classification of digital images are discussed in section A.2.4.

3.2.3 Image Processing

According to computer science, digital image processing (DIP) means the use of computer algorithms to perform image processing on digital images [24]. Basically it deals with manipulation of digital images using digital computer. It focuses on developing a computer system that can perform processing on an image. In simple words, the input is digital image and the output is some useful information (or it can be in a form of another digital image) [25]. Adobe Photoshop is a common example of widely used application for digital image processing. On the other hand, Matlab, LabView etc. are also frequently used by researchers and professionals. Figure 3.3 illustrates how a digital image is processed using a computer system.

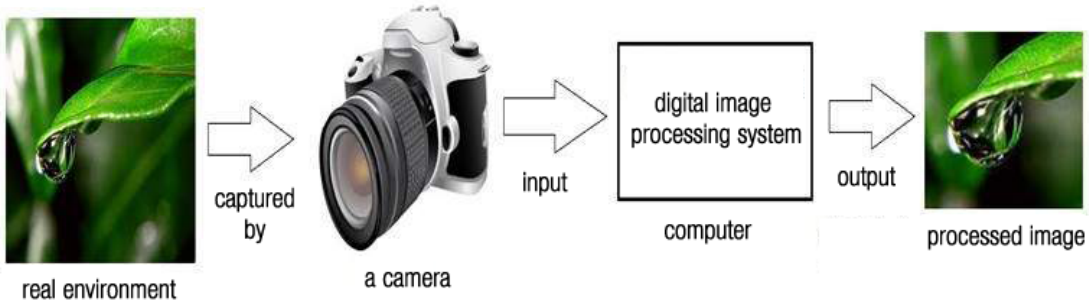


Figure 3.3: Digital image processing [25]

Now the next phase will be to recognize a face. In principle, in order to identify a face using machine vision system, we can utilize any one of the following two methods:

1. Machine-Learning based approach
2. Deep-Learning based approach

Initially, we will try to explore both approaches in details, then compare the detection methods and finally choose the appropriate method that suits for our prototype application.

3.3 Machine-Learning based Approach

It is the most classical approach for object detection that uses machine learning model with specific features. In simple words, ML algorithm develops a model that can categorize an input (image) based on specific features and this model is trained using a defined dataset (thousands of positive and negative images). For instance, the Viola-Jones object detection framework uses Adaptive Boosting (**AdaBoost algorithm**) that selects images among thousands and it is based on Haar-like features [26].

3.3.1 AdaBoost Algorithm

AdaBoost is a ML meta-algorithm introduced by Yoav Freund and Robert Schapire. It is the first really successful boosting algorithm developed for binary classification. Please note that, boosting is a general ensemble method that creates a strong classifier from a number of weak classifiers [27]. There exist two kinds of classifiers in ML which are as follows:

- Weak classifier
- Strong classifier

A weak classifier (or weak learner) is a classifier that is slightly correlated with the true classification (performs better than random guessing). On the other hand, a strong classifier (or strong learner) is a classifier that is arbitrarily well-correlated with the true classification. AdaBoost is based on the following three ideas [28].

1. It combines a lot of weak classifiers to make classifications. The weak classifiers are almost always **stumps**[1].
2. Some stumps get more **say**[2] in the classification than others.
3. Each stump is made by taking the previous stump's mistakes into account.

Pseudocode

Now we are interested to know how AdaBoost algorithm works on a given dataset. So, the pseudocode of the algorithm is described below [29].

Given training data $(x_1, y_1), \ldots, (x_m, y_m)$

$y_i \in \{-1, +1\}, x_i \in X$ is the object or instance, y_i is the classification.

[1] A stump is known as a tree with just one node and two leaves

[2] Amount of say of a stump is determined by the following formula, Amount of Say $= \frac{1}{2}\log\frac{1-\text{Total Error}}{\text{Total Error}}$, where **Total Error** refers to the sum of the weights associated with the incorrectly classified samples.

for $t = 1, \ldots, T$

create distribution D_t on $\{1, \ldots, m\}$

select weak classifier with smallest error ϵ_t on D_t

$\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$

$h_t : X \to \{-1, +1\}$

output single classifier $H_{final}(x)$

Mathematical Derivation

It is important to understand how AdaBoost generates the result classifier in a step-by-step approach. So figure 3.4 illustrates how it obtains the output classifier.

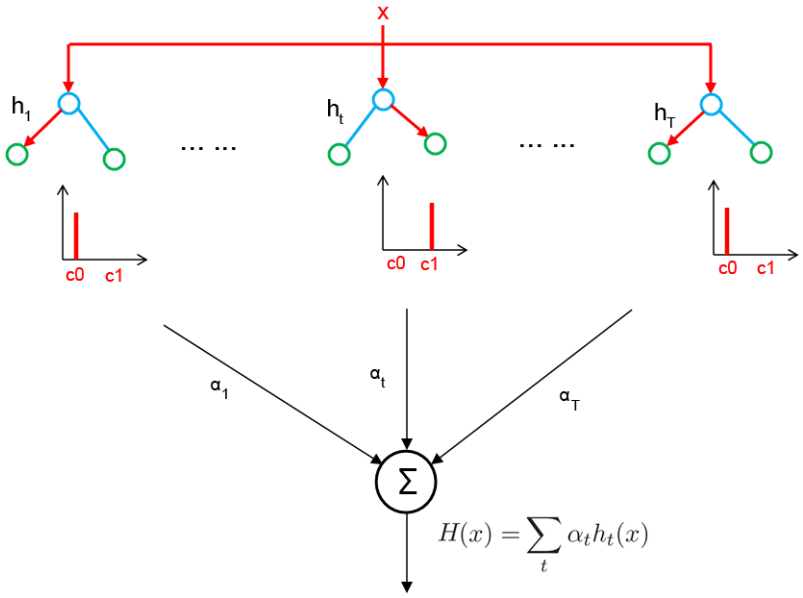


Figure 3.4: AdaBoost classifier

Now we are interested to see how the distributions are constructed.

$$D_1(i) = \frac{1}{m}$$

and given D_t and h_t

$$D_{t+1} = \frac{D_t(i)}{Z_t} c(x)$$

$$c(x) = \begin{cases} e^{-\alpha t} & : y_i = h_t(x_i) \\ e^{\alpha t} & : y_i \neq h_t(x_i) \end{cases}$$

$$D_{t+1} = \frac{D_t(i)}{Z_t} e^{-\alpha_t y_i h_t(x_i)}$$

where Z_t = normalization constant

$$\alpha_t = \frac{1}{2} ln \frac{1-\epsilon_t}{\epsilon_t} > 0$$

Finally, we get a single classifier using the following formula: $H_{final}(x) = \text{sign} \sum_t \alpha_t h_t(x)$

Since we want to obtain the classifier in quickest possible time, so **Haar-like features** will be the most suitable choice over the other features, as far as the calculation speed and accuracy is concerned.

3.3.2 Haar-like Features

In simple words, haar-like features are digital image features which are used in object recognition. Object Detection using Haar feature-based cascade classifiers is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images [30].

Here our primary objective is to work with face detection. Later, we will extend the same idea for eye and mouth detection. Initially, the algorithm requires a significant amount of positive images (images of faces) and negative images (images without faces) to train the classifier. Then necessary features need to be extracted from it. So, Haar-like features are used. Figure 3.5 illustrates the Haar-like features in details. Each feature represents a single value obtained by subtracting sum of pixels inside the white rectangle from sum of pixels inside the black rectangle.

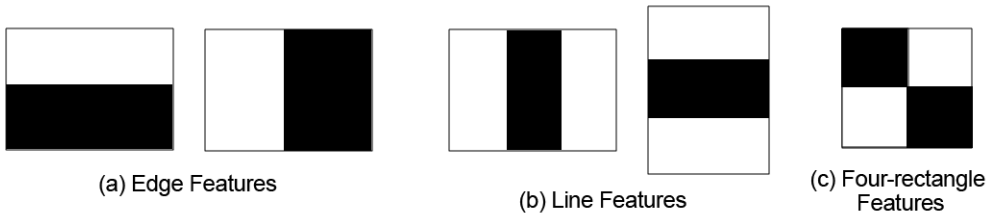


Figure 3.5: Haar-like features

After extracting features from an image, every feature should be associated with an identifier, which is known as the feature descriptor. From figure 3.5, we can see that there exists three kind of feature descriptors.

1. **Bi-rectangle feature descriptor**: Obtained by subtracting the sum of pixels of two rectangular regions (either horizontally or vertically adjacent) of same size and shape.
2. **Tri-rectangle feature descriptor**: Obtained by subtracting the sum of pixels of two outer rectangles from the sum of pixels of central rectangle (either horizontally or vertically adjacent).
3. **Tetra-rectangle feature descriptor**: Obtained by subtracting the sum of pixels of two diagonally-paired rectangles from the remaining pair.

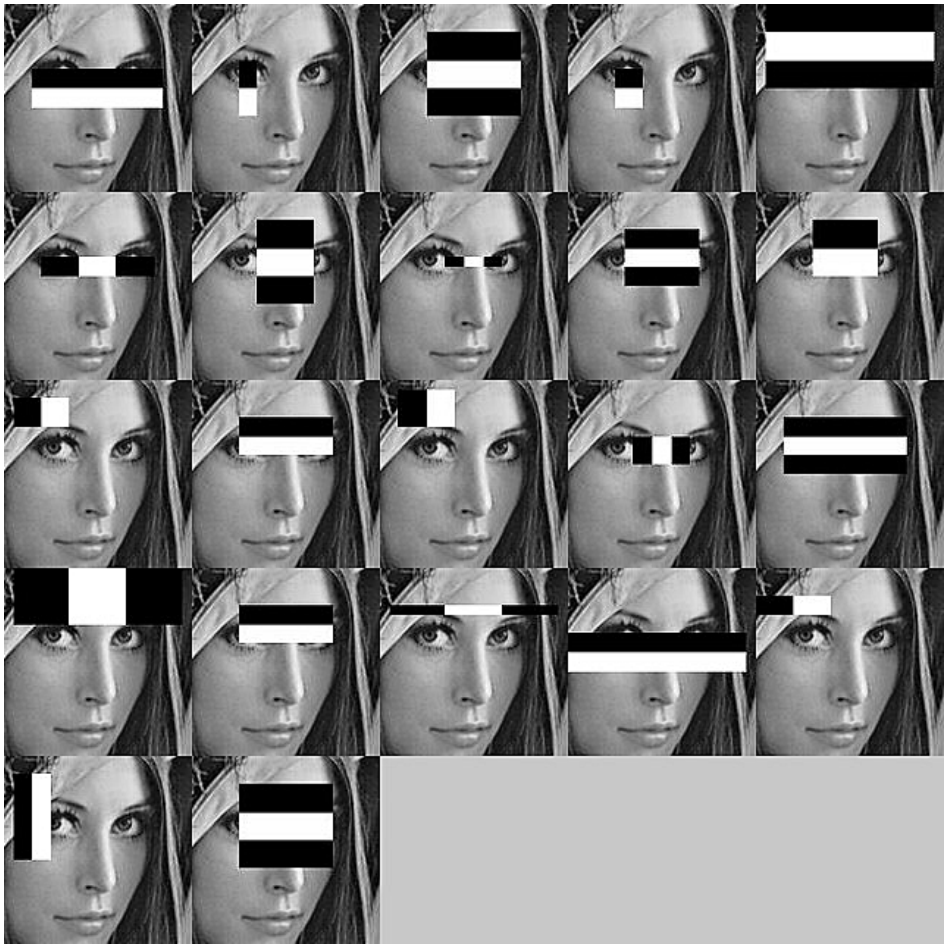


Figure 3.6: Haar-like features used for face detection [31]

Figure 3.6 shows different kind of Haar-like features used for face detection while executing Viola-Jones algorithm. Since most of the regions in an image is non-face region, so it is easier to check if a window is not a face region. If it is a non-face region, it is immediately discarded and never processed again. On the other hand, if a window passes, the second stage of features is applied and the process is continued. The window which passes all the stages is identified as a face region. Basically this approach introduced the concept of **cascade of classifiers** which we will discuss in the following section.

3.3.3 Cascade of Classifiers

While we train a classifier to identify positive and negative images, it is always important to choose the features with minimum error rate. In the context of face recognition, this number of features can be extremely high. For instance, a 24×24 size window might consist of 160000+ features. With the help of **AdaBoost algorithm**, it is possible to reduce the number of features around 6000. But still it is very much inefficient and time-consuming to apply this number of features.

In order to solve this problem, Paul Viola and Michael Jones introduced the concept of cascading classifiers. Instead of trying all 6000 features on a defined window, these are grouped into different stages of classifiers and applied one-by-one. If a window fails in first stage, it is automatically discarded - which means the remaining features are not applied on it. If it passes, the second stage of features are applied and thus the process continues. The

window which passes all stages is a face region.

Figure 3.7 illustrates a simple cascade of classifiers. Here $F_1, F_2, F_3, \ldots, F_N$ denotes Haar-like features that we have already discussed in section 3.3.2.

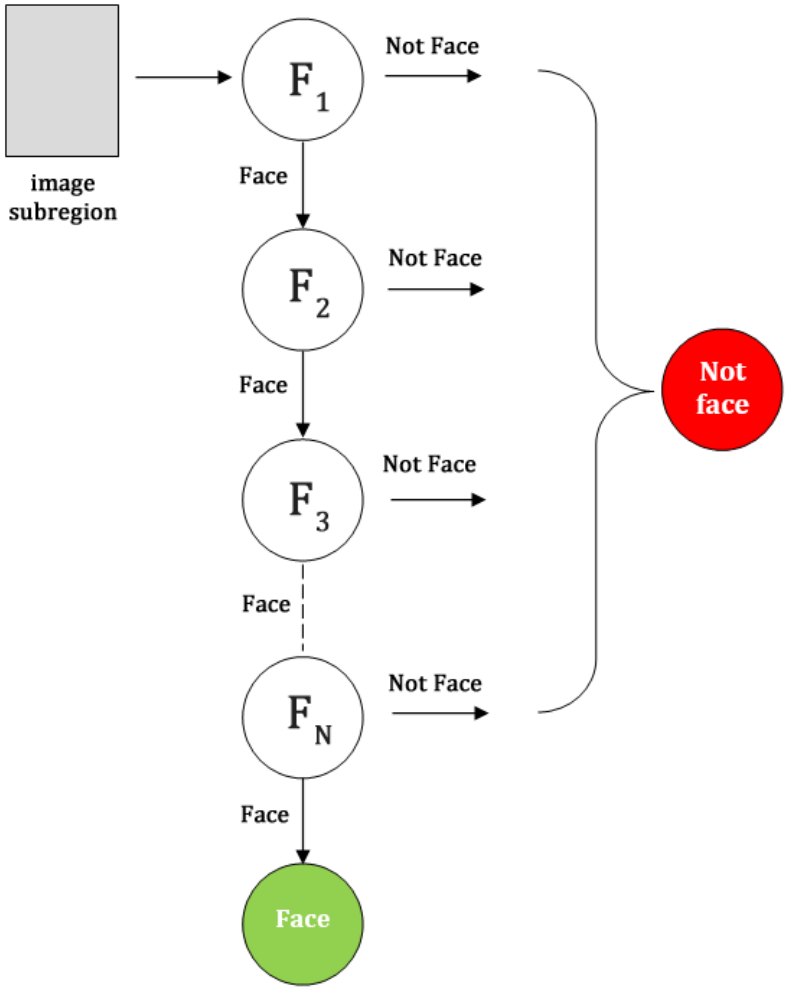


Figure 3.7: Cascade of classifiers

Please note that, Haar-like feature of any size can be calculated in constant time by using the concept of "summed-area table" (approximately 60 microprocessor instructions for a bi-rectangle feature descriptor). In the image processing domain, summed-area table is also known as **integral image**.

3.3.4 Integral Image

Integral image is used for calculating the sum of pixel values in a given image or a rectangular subset of a grid (the given image). It can be used for calculating the average intensity within a given image. Before using the integral image, input image should be in grayscale first [32].

In computer science domain, the concept of integral image is well-known as summed-area table. In this table, any point (x, y) is represented by a value, which is basically the sum of all the pixel values above, to the left and of course including the original pixel value of (x, y) itself. Figure 3.8 shows how to obtain integral image at any point $s(x, y)$.

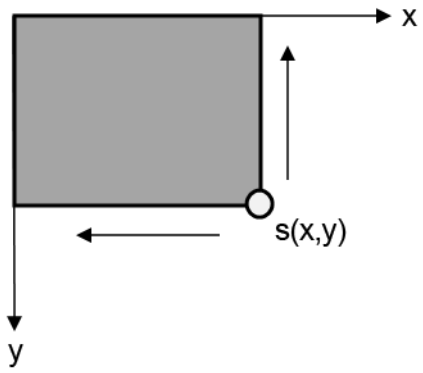


Figure 3.8: Integral image

Ideally, integral image is calculated using the following equation:

$$s(x, y) = i(x, y) + s(x - 1, y) + s(x, y - 1) - s(x - 1, y - 1) \tag{3.1}$$

Figure 3.9 illustrates how an integral image is calculated at each point $s(x, y)$.

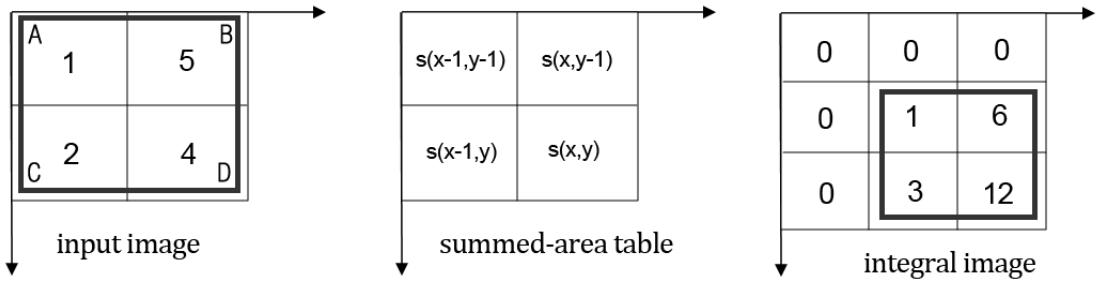


Figure 3.9: Calculation of integral image

Now we are interested to calculate integral image for each point A, B, C and D in figure 3.9 in a step-by-step approach. Please note that, the order of calculation is important here, which means, we have to calculate for point A first, then for point B, then for point C and finally for point D.

Calculation for Point A

- $i(x, y) = 1$, pixel value obtained from input image
- $s(x - 1, y) = 0$, because $x - 1$ is outside of image boundary; so value is 0
- $s(x, y - 1) = 0$, because $y - 1$ is outside of image boundary; so value is 0
- $s(x - 1, y - 1) = 0$, because both $x - 1$ and $y - 1$ are outside; so value is 0

Now, if we apply equation 3.1 for point A, we get, $s(A) = 1 + 0 + 0 - 0 = 1$

Calculation for Point B

- $i(x, y) = 5$, pixel value obtained from input image
- $s(x-1, y) = 1$, because we already calculated $s(A) = 1$
- $s(x, y-1) = 0$, because $y-1$ is outside of image boundary; so value is 0
- $s(x-1, y-1) = 0$, because both $x-1$ and $y-1$ are outside; so value is 0

Now, if we apply equation 3.1 for point B, we get, $s(B) = 5 + 1 + 0 - 0 = 6$

Calculation for Point C

- $i(x, y) = 2$, pixel value obtained from input image
- $s(x-1, y) = 0$, because $x-1$ is outside of image boundary; so value is 0
- $s(x, y-1) = 1$, because we already calculated $s(A) = 1$
- $s(x-1, y-1) = 0$, because both $x-1$ and $y-1$ are outside; so value is 0

Now, if we apply equation 3.1 for point C, we get, $s(C) = 2 + 0 + 1 - 0 = 3$

Calculation for Point D

- $i(x, y) = 4$, pixel value obtained from input image
- $s(x-1, y) = 3$, because we already calculated $s(C) = 3$
- $s(x, y-1) = 6$, because we already calculated $s(B) = 6$
- $s(x-1, y-1) = 1$, because we already calculated $s(A) = 1$

Now, if we apply equation 3.1 for point C, we get, $s(D) = 4 + 3 + 6 - 1 = 12$

In a summary, it does not matter however large is our input image, the calculation involves only four pixels for a given pixel - which makes the process super-fast. Next, we are interested to see the **workflow of face detection** in order to identify necessary facial components.

3.3.5 Workflow of Face Detection

In chapter 1, we already acknowledged about some of the popular detection algorithms for facial components recognition. For example, a standard facial expression recognition is explained in figure 1.3. The classical idea is a step-by-step approach, like below:

1. We pre-process the input image in our desired colorspace (e.g, grayscale)
2. We identify the non-facial and facial regions. Non-facial region is immediately discarded.
3. If it is a facial region, we try to identify different facial components like eye, mouth etc.

Figure 3.10 illustrates the workflow of facial components detection algorithm in details.

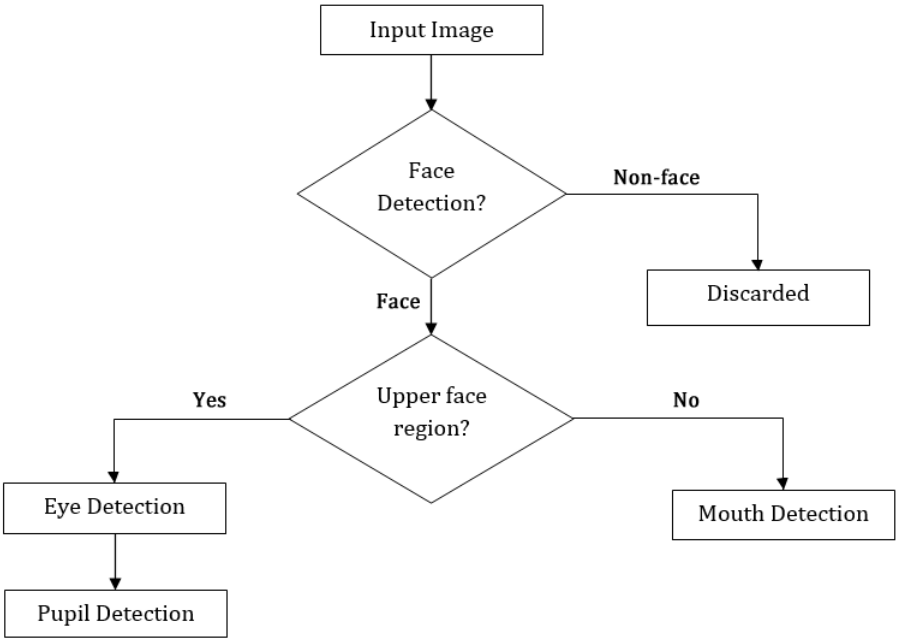


Figure 3.10: Workflow of facial components detection

At each of its detection block, we will use the **CascadeClassifier.DetectMultiScale** method, which is described in section C.1.1. In the following sections, we will describe face, eye and mouth detection in details.

Face Detection

For face detection, `haarcascade_frontalface_default.xml` [33] is a popular stump-based 24×24 discrete Adaboost frontal face classifier. It contains 25 stages and 24×24 Haar-like features. Figure 3.11 shows how our application detects face in different lighting conditions.

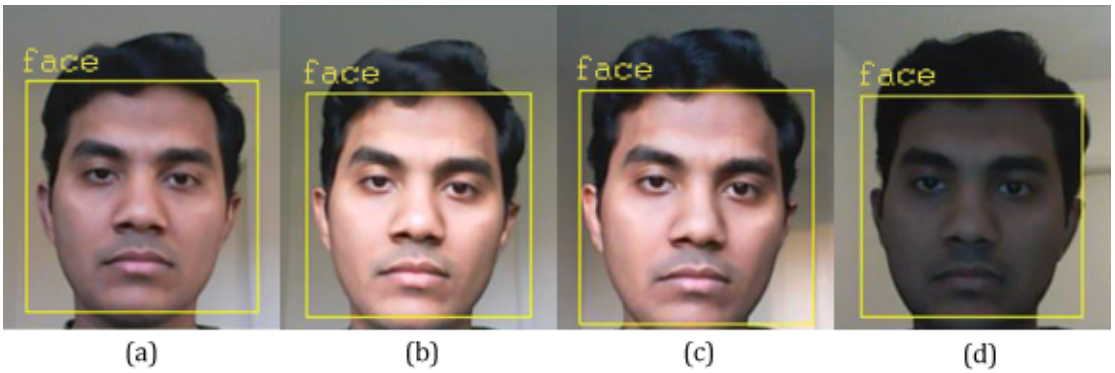


Figure 3.11: Face detection in different lighting conditions (a) 10:00 morning (no extra light), (b) 14:00 afternoon (no extra light), (c) 18:00 evening (no extra light), (d) 20:00 night (with 5 Watt yellow bulb)

Eye Detection

In the context of eye detection, `haarcascade_eye.xml` [34] is a stump-based 20×20 discrete Adaboost eye classifier. It is a well-known eye detector that can work inside detected face region. It contains 24 stages and 20×20 Haar-like features. Figure 3.12 shows how our application detects eyes in different lighting conditions.

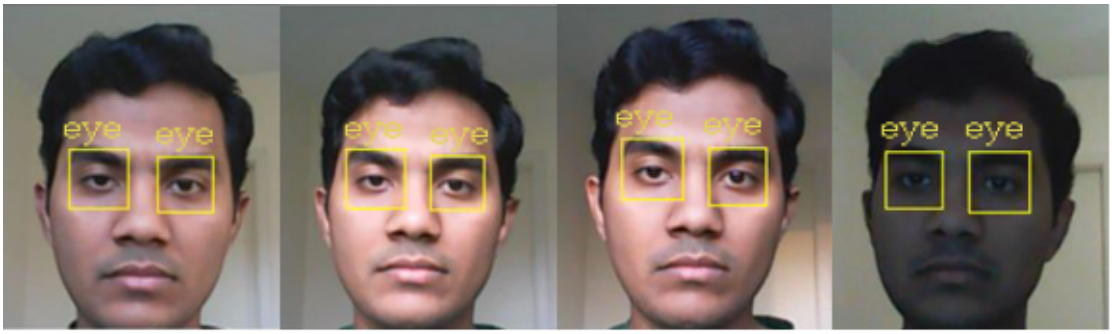


Figure 3.12: Eye detection in different lighting conditions (a) 10:00 morning (no extra light), (b) 14:00 afternoon (no extra light), (c) 18:00 evening (no extra light), (d) 20:00 night (with 5 Watt yellow bulb)

Pupil Detection

Since there exists no discrete Adaboost pupil classifier, so we need to use the detected eye regions and apply customized algorithm to detect pupil. Firstly, we need to transfer the detected eye regions from BGR to GRAYSCALE colorspace. Then we have to apply down-sampling and upsampling steps of Gaussian pyramid decomposition sequentially. So it will make the eye regions smoother than before. Now we have to do binary threshold operation on the regions in a way that if a pixel value is greater than the threshold value then it will become WHITE; otherwise it will become BLACK. So, it will clean up everything except the pupil area. Finally, we have to apply circle-finding algorithm in order to detect the pupil. In section C.1.1, the sequence of activities is written in C# language. Figure 3.13 shows how our application detects eyes in different lighting conditions.

Figure 3.13: Pupil detection in different lighting conditions (a) 10:00 morning (no extra light), (b) 14:00 afternoon (no extra light), (c) 18:00 evening (no extra light), (d) 20:00 night (with 5 Watt yellow bulb)

Mouth Detection

A stump-based 25 × 15 discrete Adaboost mouth detector named `haarcascade_mouth.xml` [35] is a good classifier that can be used for mouth detection inside detected lower-face region. It contains 17 stages and 25 × 15 Haar-like features. Figure 3.14 shows how our application detects mouth in different lighting conditions.

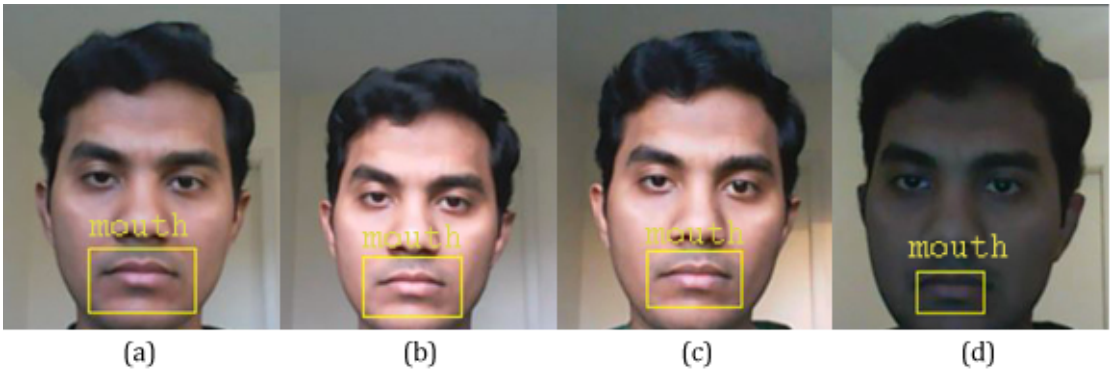


Figure 3.14: Mouth detection in different lighting conditions (a) 10:00 morning (no extra light), (b) 14:00 afternoon (no extra light), (c) 18:00 evening (no extra light), (d) 20:00 night (with 5 Watt yellow bulb)

Now it is clear that we use different classifiers for different facial component detection. When we speak about training our classifiers, in order to use them for detection algorithms, it is important to understand the concept of **convolutional neural network**.

3.3.6 Convolutional Neural Network

D. H. Hubel and T. N. Wiesel proposed a cascading model that explains how mammals visually perceive the world around them using a layered architecture of neurons within the visual cortex, that consists of two types of basic visual cells - simple cell and complex cell. According to their hypothesis, complex functional responses created by "complex cells" are constructed from more simplistic responses from "simple cells". For example, simple cells might respond to oriented edges and lines, while complex cells will not only respond to oriented edges and lines, but also with a degree of spatial invariances. This concept inspired researchers to develop similar pattern recognition mechanisms in computer vision [36].

A simple Convolutional Neural Network (CNN) is a sequence of layers where every layer transforms one volume of activations to another through a differential function. It means, each layer is responsible to convert the information from the values, available in the previous layers, into some more complex information and propagate to the next layers for further generalization [36].

The CNN consists of two basic building blocks:

- **Convolution Block**: It contains the Convolution Layer and the Pooling Layer. This layer is responsible for Feature-Extraction.
- **Fully Connected Block**: It contains a fully connected simple neural network architecture. This layer is responsible for Classification, based on the input from the convolutional block.

Figure 3.15 illustrates the basic structure of a CNN.

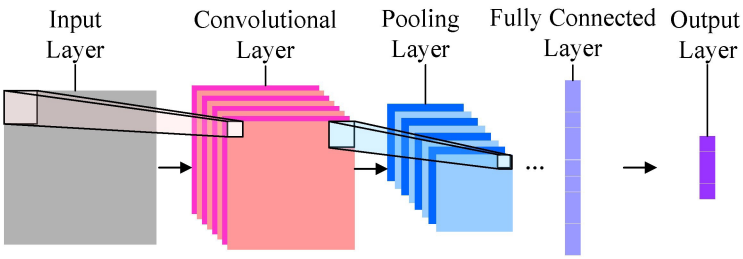


Figure 3.15: Convolutional neural network [36]

Here the convolutional layer is responsible for feature extraction. So we want to extract Haar-like features (described in section 3.3.2) in this layer. For example, we are given an input image as shown in figure 3.16a. We are interested to extract the edge feature (like figure 3.16b) from it. So, we prepare the filter matrix as figure 3.16c and then apply convolution operation on the given image. Figure 3.17 illustrates how the filter matrix sweeps over the input image and performs convolution operation at each step.

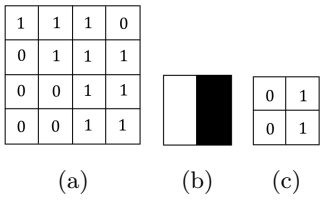


Figure 3.16: (a) input image (b) edge feature (c) filter matrix

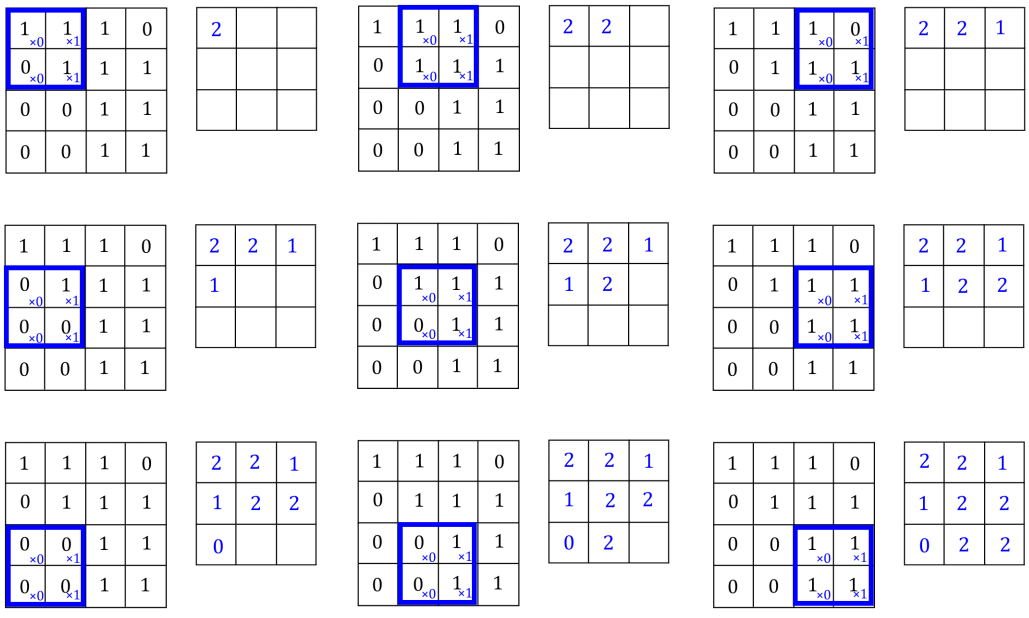


Figure 3.17: An example about how convolution works

3.3.7 Discussion

Machine-learning approach (using Haar-cascade classifier and OpenCV library methods) is a classical object-detection framework as far as the applicability, robustness, speed and resource-availability are concerned. Both in laboratory and real-life environment, the overall system performance is satisfactory. From technical perspective, it is easy to implement, install and maintain the system. Since the file-size of the cascading classifiers (`.xml`) is quite low, so it does not affect the system performance, even if the application requires to do LIVE-streaming. On the other hand, training mechanism is comparatively easier than other known methods, because OpenCV provides necessary library function support for cascade-training. However, the accuracy of the system needs further improvement, because we could notice a good percentage (in between $10\% - 15\%$) of false-positive detection in the frame.

3.4 Deep-Learning based Approach

It is the latest approach for object detection that uses deep learning networks, in a similar way how our human brain works to solve problems. In simple words, a deep learning algorithm develops an ANN model that sends the input (data of image) through different layers of the network and classifies the input based on decision being made after processing outputs at each layer using specific features respectively.

For instance, the **YOLO object detection framework** uses a single Deep Learning Network that divides the input (data of image) into regions (predicts spatially separated bounding boxes) to apply probabilities on them and classifies the input based on the probability results obtained from each layer [37]. So, we are interested to explore this detection method.

3.4.1 YOLO Object Detection Framework

YOLO stands for **You Only Look Once** - which justifies the statement of this object detection framework, because it looks at the whole image only once during test time. It redefines the concept of object detection from pixel level calculation to bounding box coordinates and class probabilities. YOLO divides input image into $S \times S$ grid and each grid predicts only one object. In the following sections, we will discuss shortly about **YOLO model**, **YOLO network architecture** and **classification** of different YOLO frameworks.

YOLO Model

YOLO strongly maintains a unified detection model, which means it unifies the separate components of object detection into a single neural network. It uses features from the entire image to predict each bounding box. At the same time, it predicts all bounding boxes across all classes for an image. It divides the input image into an $S \times S$ grid. If the center of an object falls into a grid, that grid is responsible to detect that object [37].

Each grid predicts B bounding boxes and confidence scores for those boxes. Each bounding box consists of 5 predictions: x, y, w, h and confidence. The (x, y) coordinate is the center, w is the width and h is the height of the bounding box. Confidence can be defined as $\Pr(Object) * \text{IOU}^{\text{truth}}_{\text{pred}}$. If no object exists in a grid, then the confidence score should be zero. Otherwise, it will be intersection over union (**IOU**) between the predicted box and the ground truth [37].

Each grid also predicts conditional class probability, $C = \Pr(Class_i|Object)$. During test time, class-specific confidence scores for each box is calculated by multiplying the conditional class probabilities and the individual box confidence predictions, like below [37]:

$$\Pr(Class_i|Object) * \Pr(Object) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(Class_i) * \text{IOU}^{\text{truth}}_{\text{pred}} \tag{3.2}$$

YOLO Network Architecture

YOLO network architecture is inspired by GoogLeNet model for image classification. It consists of 24 convolutional layers followed by 2 fully connected layers. Instead of the inception modules used by GoogLeNet, it simply uses 1×1 reduction layers followed by 3×3 convolutional layers [37]. The final output of this network is the $7 \times 7 \times 30$ tensor of predictions. Figure 3.18 shows the full architecture of the YOLO detection network.

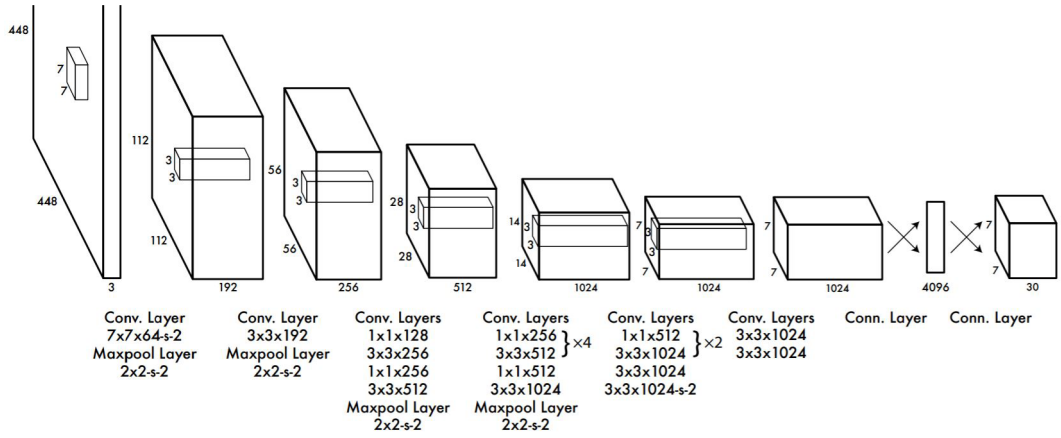


Figure 3.18: YOLO network architecture [37]

Classification

Currently there exists 3 variants of YOLO detection system which are as follows:

- **YOLO** or **YOLOv1**
- **YOLO9000** or **YOLOv2**
- **YOLOv3**

YOLOv1 is the first implementation of YOLO. It uses DarkNet framework and ImageNet-1000 dataset. It cannot identify small objects if they appear in cluster. **YOLOv2** is the improved version of YOLO. It uses DarkNet-19 framework and ImageNet. When it uses COCO dataset, it struggles to find clothing or equipment, because COCO does not have bounding box label for any type of clothing or equipment. Finally, **YOLOv3** is the improvement of YOLO9000. It uses DarkNet-53 and OpenImages dataset. Like YOLO9000, bounding box prediction (using anchor box) is the backbone of YOLOv3 detection system. Figure 3.19 illustrates how the 4 coordinates are predicted in YOLOv3 detection system.

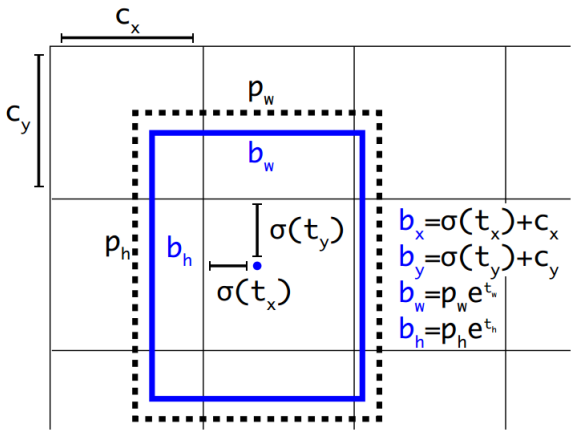


Figure 3.19: Bounding box prediction in YOLOv3 detection system [38]

As figure 3.19 shows, the YOLOv3 network tries to predict the coordinates of each bounding box, t_x, t_y, t_w, t_h. If it finds that the cell has offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width p_w and height p_h, then the predictions correspond to the following formula [38]:

$$b_x = \sigma(t_x) + c_x \tag{3.3}$$

$$b_y = \sigma(t_y) + c_y \tag{3.4}$$

$$b_w = p_w e^{t_w} \tag{3.5}$$

$$b_h = p_h e^{t_h} \tag{3.6}$$

Now we will discuss shortly about **Darknet framework** in the following section.

3.4.2 Darknet Framework

Darknet is an open source neural network framework written in C programming language and CUDA. CUDA stands for **Compute Unified Device Architecture** - which is an API model created by Nvidia in order to support CUDA-enabled GPU for general purpose processing. Darknet is fast, easy to install, and supports CPU and GPU computation [39]. It features YOLO (already discussed in section 3.4.1), ImageNet Classification (classify images for the 1000-class ImageNet challenge with ResNet and ResNeXt). Additionally, Darknet can be used to run neural networks backward which is known as Nightmare feature.

Now we are interested to know whether Darknet and YOLO can be applicable for our prototype application or not. In order to do this feasibility analysis, we will learn about the **workflow of face detection** in YOLO and then try to develop a dummy application for face detection.

3.4.3 Workflow of Face Detection

Figure 3.20 shows the workflow of face detection in YOLO. Section A.2.6 provides details information of implementation in a step-by-step approach.

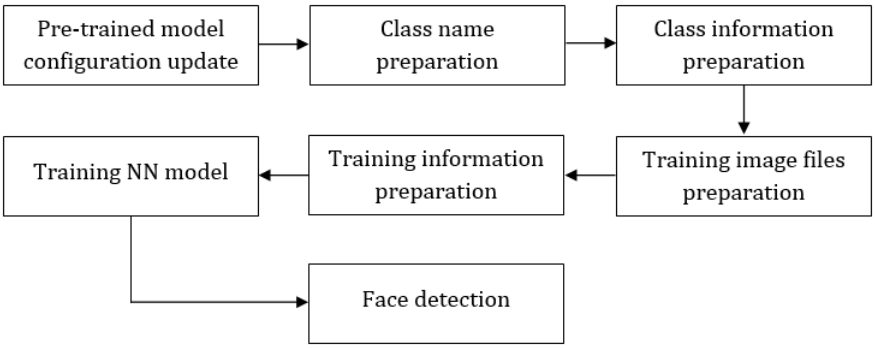


Figure 3.20: Workflow of face detection in YOLO

The minimal implementation of face detection using `Alturos.Yolo` wrapper class is described in section C.1.2. Figure 3.21 shows how it detects face in an average daylight condition.

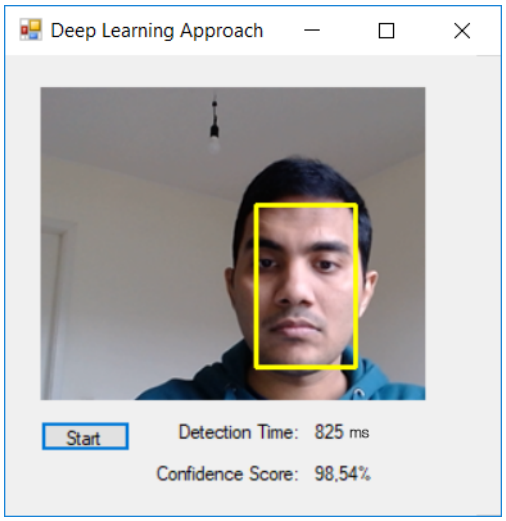


Figure 3.21: Face detection using Darknet and YOLO

3.4.4 Discussion

Deep-learning approach (using Darknet with YOLO object detection system) is the latest object-detection framework introduced by Joseph Redmon and others [37]. Though there exists tradeoff between speed and accuracy of the detection framework, but the overall system performance is quite good; even it can be excellent when multiple GPU-RAM is available. In principle, the system proposes a robust network that can make class-predictability while looking at the input image only once during test-time. However, the technical implementation still has a lot of limitations. For example, it might take several days, or even weeks to train a single-class model if there is no GPU-RAM available. Again, the Darknet framework can be compiled in Windows OS, if and only if `x64` build configuration (64-bit platform) is set in Visual Studio. Finally, it might take some time to load the pre-trained model (configuration and weight file) in the code and thus the application can take some time to detect objects as well. So, significant amount of RAM and high GPU-RAM should be pre-requisite before executing training or testing operations.

3.5 Comparison between Face Detection Approaches

Since face detection is the main pre-requisite of our system, so we inspect both machine-learning approach and deep-learning approach in terms of delivering acceptable results. However, both approaches have some merits and demerits too - which we have already discussed in section 3.3.7 and 3.4.4. For our application, a comparison between these two approaches would be essential in terms of detection time and confidence.

System Configuration	**(Machine-learning algorithm) Detection Time**	**(Deep-learning algorithm) Detection Time**
AMD x64-based processor (1.80 GHz) 6.00 GB RAM, Radeon R6 Graphics 64-bit OS (Windows 10)	60 ms	4655 ms
Intel core i3 x64-based processor (2.40 GHz) 4.00 GB RAM, Intel HD Graphics 620 64-bit OS (Windows 10)	50 ms	3385 ms
Intel core i7 x64-based processor (1.90 GHz) 32.00 GB RAM, Intel UHD Graphics 620 64-bit OS (Windows 10)	30 ms	825 ms

Table 3.2: Detection time comparison between ML-based and DL-based algorithm

System Configuration	**(Machine-learning algorithm) Confidence**	**(Deep-learning algorithm) Confidence**
AMD x64-based processor (1.80 GHz) 6.00 GB RAM, Radeon R6 Graphics 64-bit OS (Windows 10)	97.92%	99.98%
Intel core i3 x64-based processor (2.40 GHz) 4.00 GB RAM, Intel HD Graphics 620 64-bit OS (Windows 10)	96.29%	99.42%
Intel core i7 x64-based processor (1.90 GHz) 32.00 GB RAM, Intel UHD Graphics 620 64-bit OS (Windows 10)	95.04%	98.54%

Table 3.3: Confidence comparison between ML-based and DL-based algorithm

3.6 Summary

Based on the study of this chapter, we choose classical machine-learning based approach (**OpenCV with Haar-like feature-based algorithm**) for our prototype application. Because we understand that low detection time is more important than accuracy, as far as the real-time video streaming and feature extraction processes are concerned. From technical perspective, we get familiarized with the step-by-step approach of face and facial component detection using OpenCV library functions. We also learn about how Darknet and YOLO framework works. In addition, we develop a few C# form applications for some experimental tests and our plan is to integrate those modules in our main application later.

Chapter 4

Behavioral Biometrics

In this chapter, we describe, in brief, some basic topics of behavioral biometrics with necessary explanations. In simple words, biometrics refers to measurement, calculation, analysis or even prediction of human activity over time. It is divided into two categories - physiological biometrics and behavioral biometrics. Physiological biometrics include face recognition, iris recognition, palm print, hand geometry etc. On the other hand, behavioral biometrics include gait analysis, keystroke dynamics, voice recognition etc. [40]. When we narrow down the behavioral biometrics to the context of computer usage, then we can refer to keystroke dynamics and mouse dynamics. In section 4.1 and 4.2, we discuss about keystroke dynamics and mouse dynamics respectively.

4.1 Keystroke Dynamics

It is possible to describe an individual's keystroke dynamics (**KD**) template by five attributes which are as follows:

- Hold time: The time difference between pressing and releasing a key.
- Release - Press delay: The time difference between the release key and the subsequent key press times.
- Press - Press delay: The time difference between the subsequent keys press times.
- Release - Release delay: The time difference between the subsequent keys release times.
- Press - Release delay: The time difference between the key press and the subsequent key release times.

Figure 4.1 illustrates a simple KD template with above-mentioned five attributes.

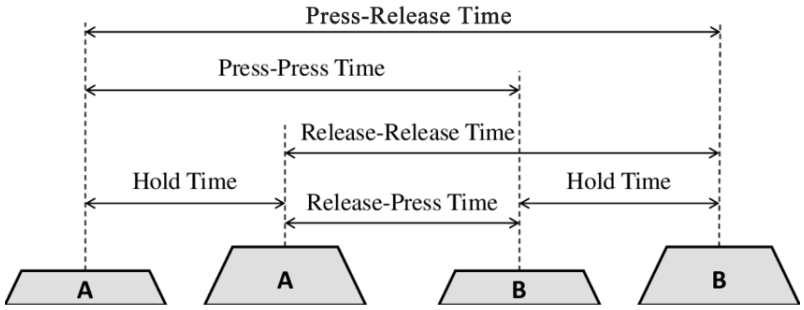


Figure 4.1: Attributes of keystroke dynamics [41]

4.2 Mouse Dynamics

Mouse dynamics is represented as a unique set of values (computed by utilizing statistical data of mouse actions) that can characterize a person's behavior measured over a period of time [42]. In order to describe an individual's mouse dynamics (**MD**) template, the following actions are considered:

- Mouse movement: Trace of every location change of mouse pointer in terms of distance, time and angle.
- Drag-n-drop: Action of dragging one or more items from one location to another location.
- Point and click: Trace of the mouse pointer's current location and number of clicks (single or double clicks).
- Silence: No activity in a particular session.

Figure 4.2 shows an example MD template with mouse move, point click and drag-n-drop.

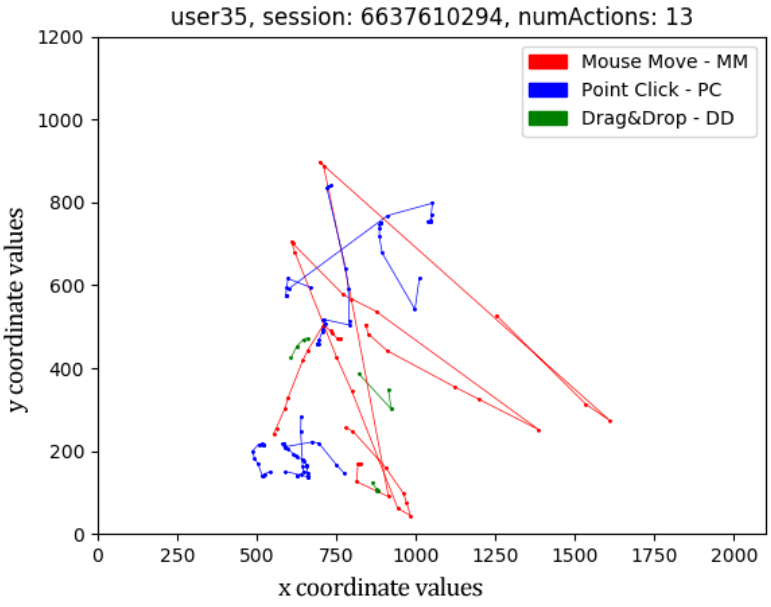


Figure 4.2: Example of mouse dynamics [43]

4.3 Behavioral Data for Tiredness Detection

According to a statistical study, workplace-fatigue can impact the drop in productivity up to 6% [44]. So it is always important to assess the tiredness level of an employee, when that person will be working in front of a computer for a long time. In a smart office, advanced software can be used to detect tiredness of a desk-employee in a silent mode by capturing the mouse and keyboard activities, and then processing the information with the help of intelligent algorithm.

A popular approach is to execute certain benchmark tasks to determine tiredness level of an employee. For example, typing the same string of characters in different day-time and measuring the keystroke dynamics - is a classical approach of determining tiredness level. In the following section, we will discuss the workflow of capturing behavioral biometrics data.

4.4 Workflow of Capturing Behavioral Biometrics

Basically, our idea is to develop a ML model with keystroke and mouse dynamics data when the user is already identified as tired. Our initial target is to detect keyboard and mouse activity when our application runs in background. In order to do that, we need global event handlers that always keep listening to the keyboard and mouse activity events. Figure 4.3 illustrates the workflow of capturing behavioral biometrics.

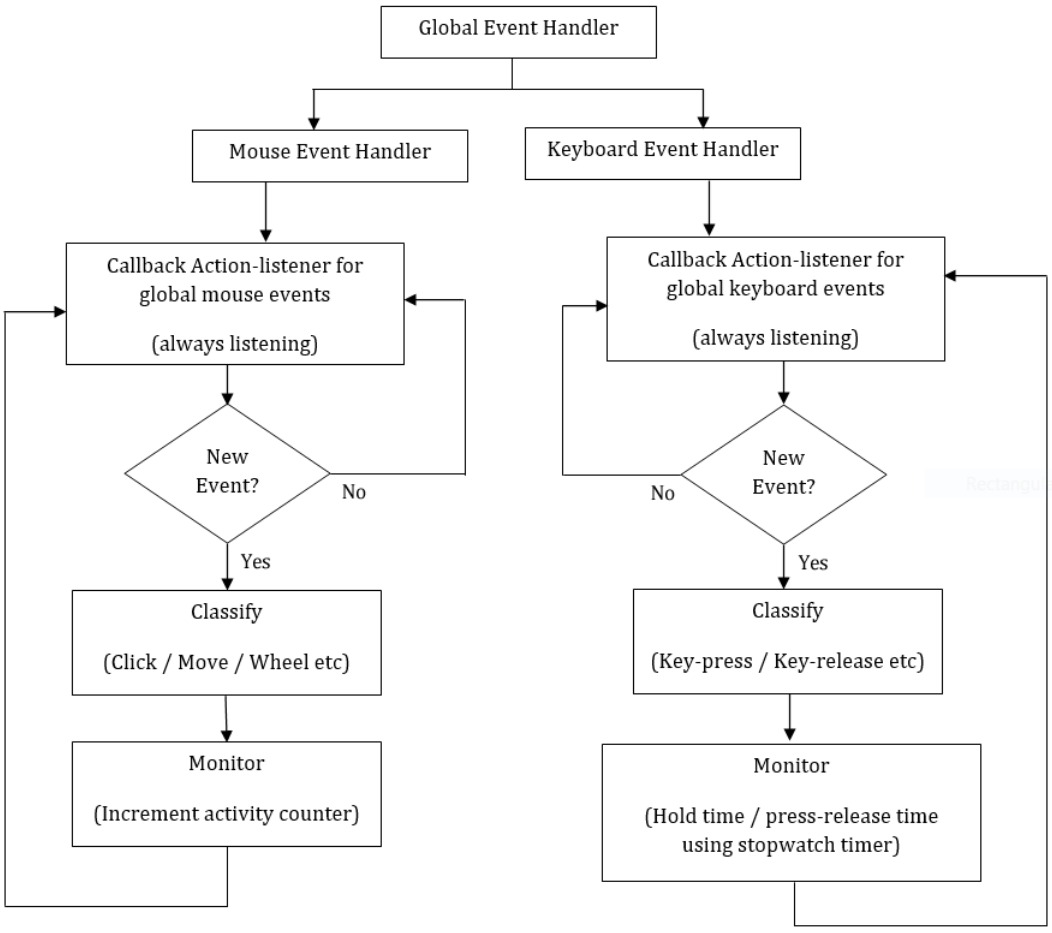


Figure 4.3: Workflow of capturing behavioral biometrics

Therefore, we have integrated a `HookManager` class [45] to process global mouse and keyboard hooks in our application. Here, we have written our own `HookManager_MouseDown`, `HookManager_MouseDoubleClick`, `HookManager_MouseMove` and `HookManager_MouseWheel` etc. event-handlers. The idea is that HookManager class always keeps listening to global mouse and keyboard actions, and depending on the type of action it triggers its corresponding event handler. Now due to the integration, it will also execute our defined event-handler as well. For example, when there is a movement of mouse, the `HookManager.MouseMove` event-handler is activated and at the same time, it will also trigger our own `HookManager_MouseMove` event-handler to do necessary measurements. Please note that, in the context of our application, `BehavioralBiometricsMonitorStart` and `BehavioralBiometricsMonitorStop` method are responsible to enable and disable necessary Hook Services respectively. Details implementation (in C# language) is written in section C.2.1.

4.5 Summary

Based on the study of this chapter, we have discussed about the following things:

- Fundamentals of behavioral biometrics (keystroke and mouse dynamics) and necessary measurements
- Relationship between employee tiredness and behavioral biometrics
- Workflow of keystroke and mouse dynamics measurement
- Integration and execution of individual event-handlers for various keyboard and mouse actions

Since we expect the state-of-the-art **tiredness assessment** result to be obtained by our designed vision-system (which is our primary objective), so our secondary objective is to analyze and correlate our result with captured keystroke dynamics (more specifically, hold-time and release-press delay) once a user is already identified as **tired**. In the context of our problem, **regression** might be the best algorithm to train our machine learning model with keystroke dynamics attributes (hold-time, release-press delay) as a set of selected features and the model can be used in future tests to deliver prediction-driven results.

Chapter 5

Proposed Methodology

5.1 Eye-status based Tiredness Detection Algorithm

This section proposes eye-status based tiredness detection algorithm and includes subsections with details information. Figure 5.1 illustrates the algorithm with necessary functional blocks.

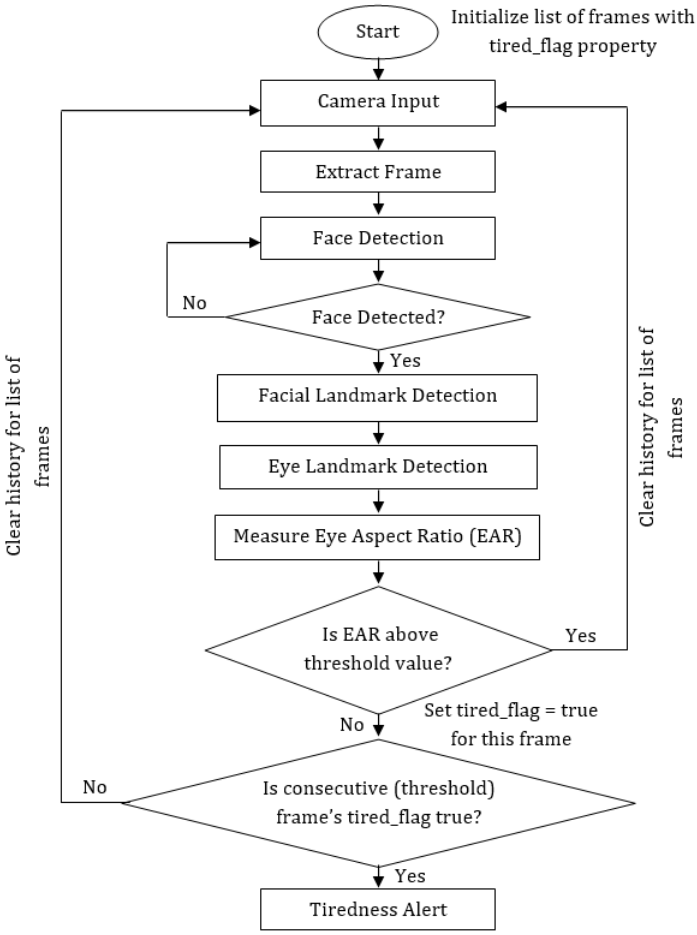


Figure 5.1: Eye-status based tiredness detection algorithm

Since we already acknowledged about the face detection algorithm (explained in chapter 3), so we are interested to understand about the **facial landmark detection**.

5.1.1 Facial Landmark Detection

OpenCV's facial landmark is known as **Facemark**, which has three different implementations of landmark detection, given below:

1. FacemarkKazemi [46]
2. FacemarkAAM [47]
3. FacemarkLBF [48]

Though all three implementations follow similar patterns, but we will use **FacemarkLBF** for our prototype application, because a trained model is available for this particular implementation only. So we download the facemark trained model `lbfmodel.yaml` [49] and use it in our application. Figure 5.2 illustrates the facial landmark detection algorithm in details.

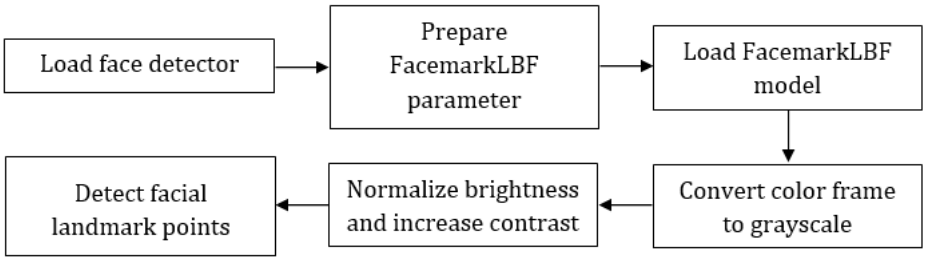


Figure 5.2: Facial landmark detection algorithm

`InitializeLandmarkIdentificationParameters` and `FacialLandmarkIdentification` are responsible to initialize and identify facial landmark points respectively. Details implementation (in C# language) is written in section C.3.1. Figure 5.3 shows facial landmark detection with necessary point indexes.

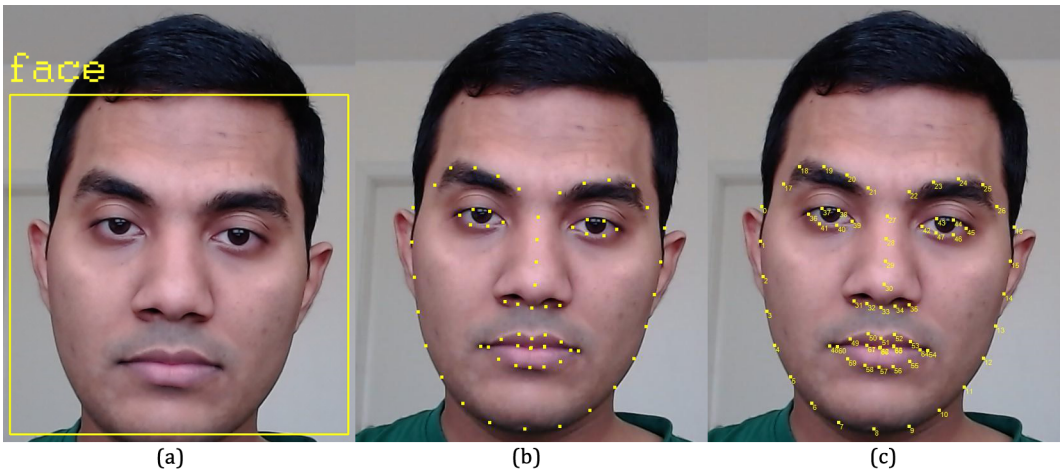


Figure 5.3: Facial landmark detection (a) face detection, (b) facial landmark points, (c) facial landmark points with respective indexes

Now our next objective is to filter out the eye landmark points. In the next section, we will discuss about the **eye landmark detection**.

5.1.2 Eye Landmark Detection

From figure 5.3, we have already identified the landmark points for both eyes, which are listed in the table below:

Facial Component	**Landmark Point Index**
Left Eye	$36, 37, 38, 39, 40, 41$
Right Eye	$42, 43, 44, 45, 46, 47$

Table 5.1: List of eye landmark points

In order to filter out the eye landmark points, we implement necessary changes in our C# application. Details implementation (in C# language) is written in section C.3.2. Figure 5.4 shows eye landmark detection with necessary point indexes.

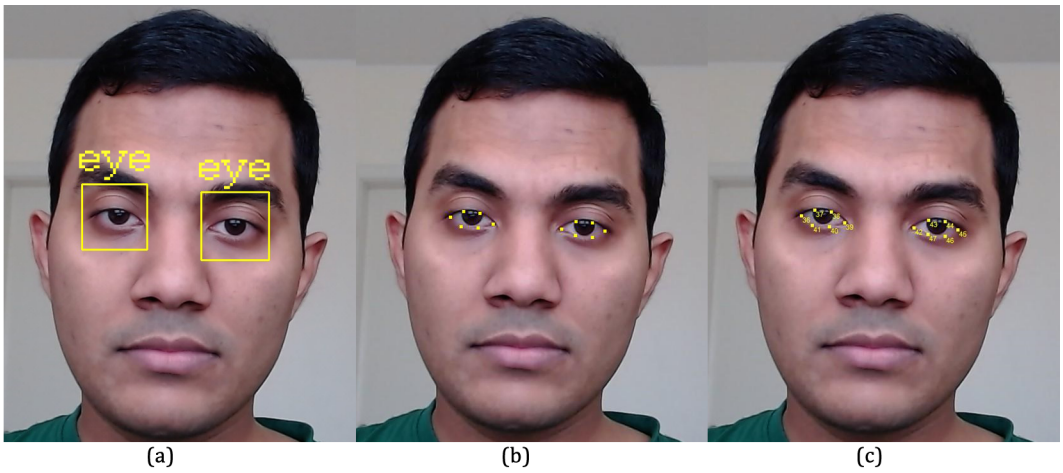


Figure 5.4: Eye landmark detection (a) input image, (b) eye landmark points, (c) eye landmark points with respective indexes

Now our next objective is to measure eye aspect ratio (**EAR**). In the next section, we will discuss about the **eye aspect ratio**.

5.1.3 Eye Aspect Ratio

Aspect ratio is an image projection attribute that can describe the proportional relationship between the width and height of that image [50]. In this case, we are interested to measure the aspect ratio of an eye. Eye aspect ratio is a single scalar quantity that can characterize the eye-opening in a frame. In principle, EAR measurement is fully invariant to a uniform scaling of image and in-plane rotation of face as well. It is partially person and head pose insensitive. As we see from figure 5.4, each eye is represented by $6(x, y)$ coordinates - starting from the left corner of the eye and then traversing clock-wise towards the remaining points. Figure 5.5 illustrates the measurements associated with calculation of eye aspect ratio.

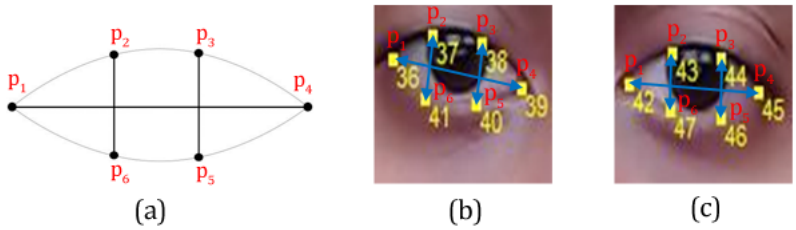


Figure 5.5: Eye aspect ratio (a) eye region with reference coordinate points, (b) left eye, (c) right eye

Based on the paper titled **Real-Time Eye Blink Detection using Facial Landmarks** [51], eye aspect ratio can be written as a mathematical equation, like below:

$$\text{EAR} = \frac{|p_2 - p_6| + |p_3 - p_5|}{2 * |p_1 - p_4|} \tag{5.1}$$

where EAR is known as the eye aspect ratio and $p_1, p_2, p_3, \ldots, p_6$ are 6 landmark points associated with eye. **Euclidean distance**[1] is calculated to measure the distance between two landmark points. The expressions used in equation 5.1 can be explained as follows:

$|p_2 - p_6|$ = Absolute value of Euclidean distance between points p_2 and p_6

$|p_3 - p_5|$ = Absolute value of Euclidean distance between points p_3 and p_5

$|p_1 - p_4|$ = Absolute value of Euclidean distance between points p_1 and p_4

In the next sub-sections, we are interested to see the calculation of EAR of both eyes and necessary equations in details.

Eye Aspect Ratio (EAR) of Left Eye

In terms of 2D coordinates, the 6 landmark points associated with left eye can be defined as: $p_1(x_{36}, y_{36})$, $p_2(x_{37}, y_{37})$, $p_3(x_{38}, y_{38})$, $p_4(x_{39}, y_{39})$, $p_5(x_{40}, y_{40})$ and $p_6(x_{41}, y_{41})$. From equation 5.1, we can calculate the eye aspect ratio (EAR) of left eye, like below:

$$\text{EAR}_{left} = \frac{\sqrt{(x_{37} - x_{41})^2 + (y_{37} - y_{41})^2} + \sqrt{(x_{38} - x_{40})^2 + (y_{38} - y_{40})^2}}{2 * \sqrt{(x_{36} - x_{39})^2 + (y_{36} - y_{39})^2}} \tag{5.2}$$

Eye Aspect Ratio (EAR) of Right Eye

In terms of 2D coordinates, the 6 landmark points associated with right eye can be defined as: $p_1(x_{42}, y_{42})$, $p_2(x_{43}, y_{43})$, $p_3(x_{44}, y_{44})$, $p_4(x_{45}, y_{45})$, $p_5(x_{46}, y_{46})$ and $p_6(x_{47}, y_{47})$. From equation 5.1, we can calculate the eye aspect ratio (EAR) of right eye, like below:

$$\text{EAR}_{right} = \frac{\sqrt{(x_{43} - x_{47})^2 + (y_{43} - y_{47})^2} + \sqrt{(x_{44} - x_{46})^2 + (y_{44} - y_{46})^2}}{2 * \sqrt{(x_{42} - x_{45})^2 + (y_{42} - y_{45})^2}} \tag{5.3}$$

[1]Euclidean distance refers to straight line distance between any two points in a Cartesian or Euclidean space. For example, Euclidean distance d between two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ is calculated using the following formula: $d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Section C.3.3 explains the method of eye aspect ratio (EAR) calculation in C# programming language. In the next section, we will explain how our application will do **tiredness detection** by evaluating eye aspect ratio (EAR) in a continuous monitoring approach.

5.1.4 Tiredness Detection

Generally, an eye can have two possible states - 1) open, 2) closed. When an eye is open, EAR remains mostly constant. On the other hand, while an eye is getting closed, the EAR starts decreasing and becomes close to zero. So we introduce a threshold value of EAR (which we indicate by $\text{EAR}_{threshold}$) that will help us to identify the current eye state. If the EAR of a frame falls below $\text{EAR}_{threshold}$, the eye is considered to be closed.

$$\text{Eye State} = \begin{cases} \text{Open}, & \text{EAR} > \text{EAR}_{threshold} \\ \text{Closed}, & \text{EAR} \leq \text{EAR}_{threshold} \end{cases} \tag{5.4}$$

$\text{EAR}_{threshold}$ is determined by trial and error method, monitoring EAR value of 10 different persons with different eye states, because we want our system to be able to classify an instance of open and closed eyes correctly. Table 5.2 shows the list of EAR values of 10 different persons that we measured for consecutive 30 frames.

Eye Aspect Ratio (EAR)					
Candidate	**Frame 1**	**Frame 2**	**Frame 3**	**Frame ...**	**Frame 30**
Person $1_{open-eye}$	0.2324	0.2258	0.2227	...	0.2505
Person $2_{open-eye}$	0.2615	0.2459	0.2417	...	0.2568
Person $3_{open-eye}$	0.2728	0.2723	0.2751	...	0.2637
Person $4_{open-eye}$	0.2495	0.2338	0.2435	...	0.2486
Person $5_{open-eye}$	0.2514	0.2505	0.2541	...	0.2601
Person $6_{closed-eye}$	0.1929	0.1916	0.1903	...	0.1367
Person $7_{closed-eye}$	0.1898	0.1827	0.1768	...	0.1362
Person $8_{closed-eye}$	0.1938	0.1640	0.1527	...	0.1219
Person $9_{closed-eye}$	0.1956	0.1932	0.1558	...	0.1402
Person $10_{closed-eye}$	0.1983	0.1972	0.1842	...	0.1586

Table 5.2: List of EAR values of 10 different persons

From table 5.2, we determine the optimal value of $\text{EAR}_{threshold}$ is 0.20. Now the main challenge is to distinguish between normal eye-blink activity and eye-close activity due to micro-sleep (tiredness). The frame-rate of our webcam is quite standard (30 FPS) - which means it takes 33.33 milliseconds to retrieve a frame. Scientists found that average duration of a single eye blink is in between 100-400 milliseconds [52]. So, if we notice that eye is closed in consecutive 3-12 frames in a long-time window of frames, we can easily make decision that an individual performs a single eye blink. Now we can utilize this concept to identify micro-sleep as well. In order to avoid conflict between normal eye-blink activity and micro-sleep, we take 12 as the **threshold** frame-number. We keep monitoring the sequence of frames in a real-time window and assign necessary tag based on the observed EAR value. If EAR value is equal or less than $\text{EAR}_{threshold}$ (0.20, determined in table 5.2), we mark this frame as a **tired** frame and increase the tired-frame counter by 1. If this counter becomes equal or greater than the threshold frame-number, we declare the individual as **tired** and trigger an alarm. With the help of trained SVM classifier, we can analyze the measured data (eye aspect ratio values and corresponding tiredness flag information) and plot necessary graph

to study the duration and frequency of micro-sleep effectively. In the following section, we discuss the process of **verification of open-eye and closed-eye detection** that helps us to validate our proposed algorithm.

Verification of Open-Eye and Closed-Eye Detection

When we notice EAR value greater than $EAR_{threshold}$ then we consider that eye state is open. On the other hand, when we find EAR value less than $EAR_{threshold}$ then we agree that eye state is closed. But it is important for us to verify this theory before applying the algorithm in our prototype application.

In order to verify the current state of eye, we utilize the pupil detection algorithm (already discussed in section 3.3.5), develop a small application using C# programming language and test random frames with different EAR values. Figure 5.6 shows the result of eye state verification for both open-eye and closed-eye.

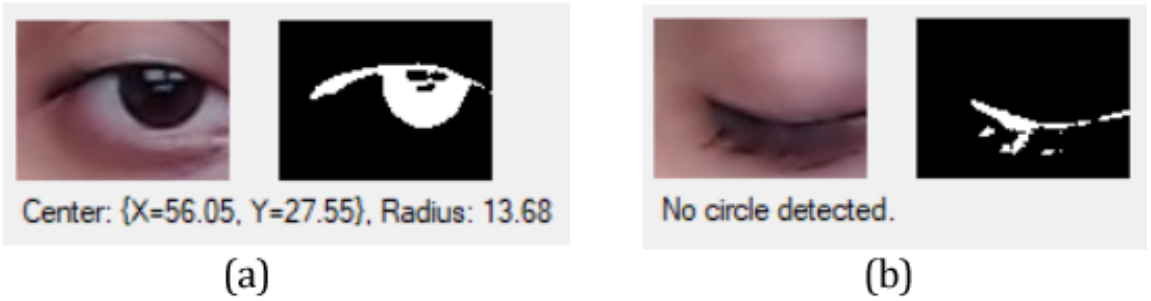


Figure 5.6: Eye state verification (a) open-eye verification, (b) closed-eye verification

In a short summary, the verification process is done in a step-by-step approach. Firstly, the image is transferred from BGR to GRAYSCALE colorspace. Then the eye region is made smoother by applying downsampling and upsampling steps of Gaussian pyramid decomposition sequentially. Finally, binary threshold operation takes place so that that if a pixel value is greater than the threshold value then it will become WHITE; otherwise it will become BLACK. So, it makes everything cleaned up except the pupil area. Finally, the circle-finding algorithm helps to detect the pupil. In case of open-eye, we can easily find a circle and do necessary measurements. On the other hand, for closed-eye, there will be no circle at all.

After analyzing the result, it is clear that our proposed algorithm's theory (related to EAR) is justified by the pupil detection algorithm.

5.2 Mouth-status based Tiredness Detection Algorithm

This section proposes mouth-status based tiredness detection algorithm and includes subsections with details information. Figure 5.7 illustrates the algorithm with necessary functional blocks.

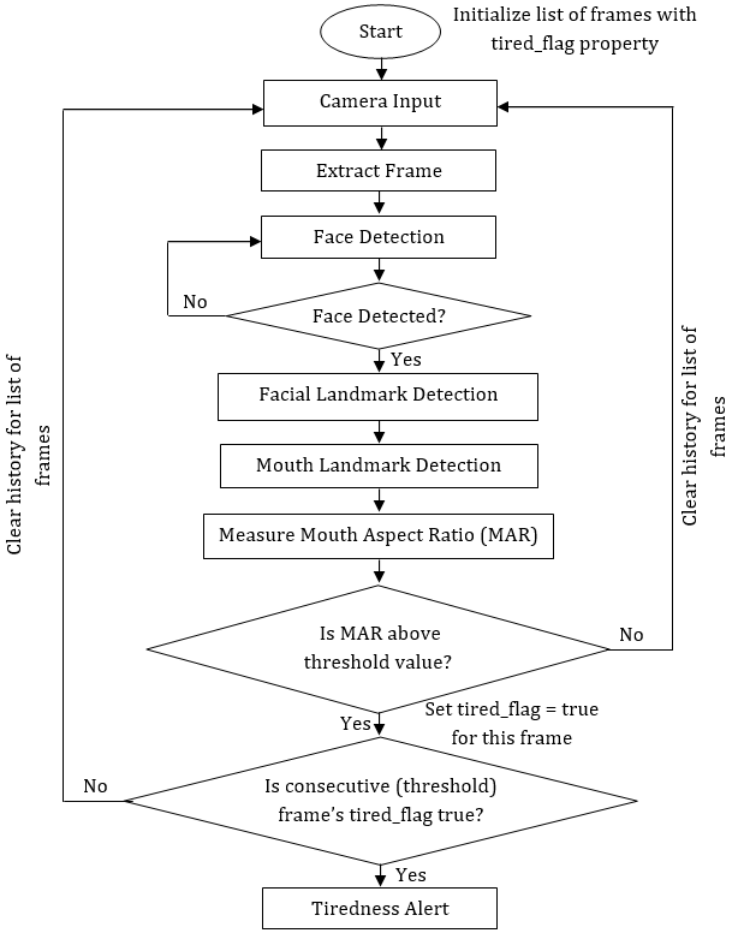


Figure 5.7: Mouth-status based tiredness detection algorithm

Since we already acknowledged about the face detection algorithm (explained in chapter 3) and facial landmark detection (explained in figure 5.2), so we are interested to understand about the **mouth landmark detection**.

5.2.1 Mouth Landmark Detection

From figure 5.3, we have already identified the landmark points for mouth. We divide these landmark points into two following groups: 1) outer edge of lips, 2) inner edge of lips - which are listed in table 5.3.

Based on the study of our algorithm, we would like to trace the outer edge of lips. In order to filter out the landmark points of outer edge of lips, we implement necessary changes in our C# application. Details implementation (in C# language) is written in section C.3.4. Figure 5.8 shows landmark detection of inner edge of lips with necessary point indexes.

Facial Component	Landmark Point Index
Outer edge of lips	48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59
Inner edge of lips	60, 61, 62, 63, 64, 65, 66, 67

Table 5.3: List of mouth landmark points

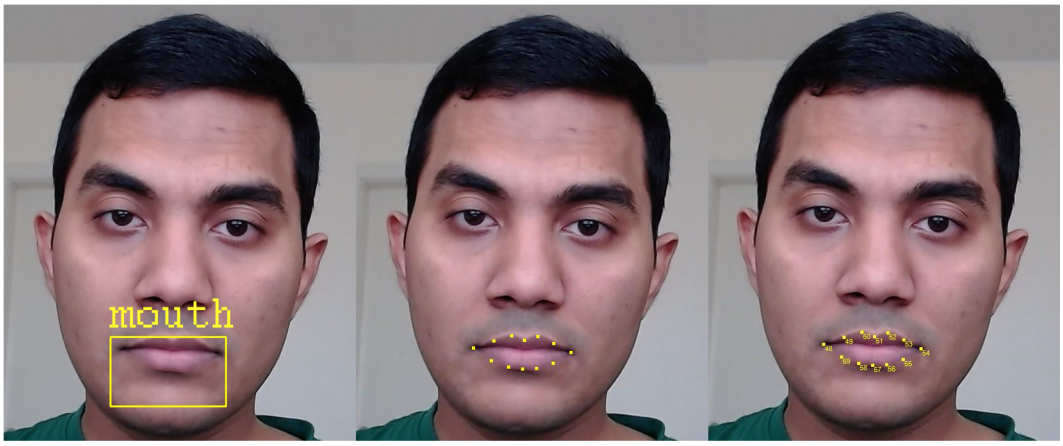


Figure 5.8: Landmark detection of outer edge of lips (a) input image, (b) landmark points of outer edge of lips, (c) landmark points of outer edge of lips with respective indexes

Now our next objective is to do measure mouth aspect ratio **MAR** in real time.

5.2.2 Mouth Aspect Ratio

Mouth aspect ratio is a single scalar quantity that can characterize the yawning state of an individual in a frame. As we see from figure 5.8, outer edge of lip is represented by $12(x, y)$ coordinates - starting from the left corner of the outer edge of lip and then traversing clockwise towards the remaining points. Figure 5.9 illustrates the measurements associated with calculation of mouth aspect ratio.

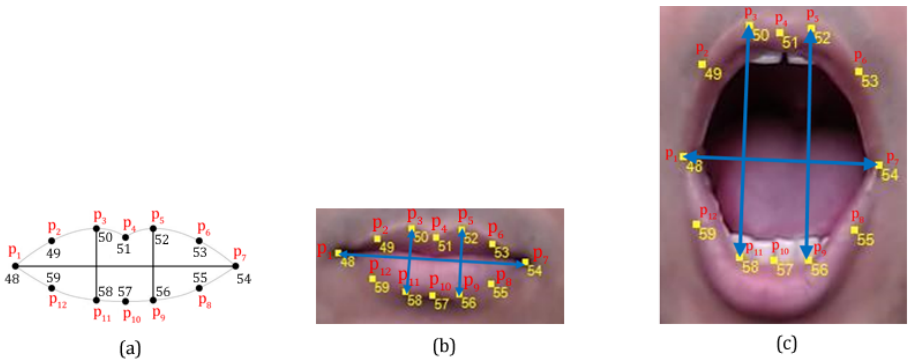


Figure 5.9: Mouth aspect ratio (a) mouth region with reference coordinate points, (b) outer edge of lip (closed state), (c) outer edge of lip (yawn state)

Based on the paper titled **Driver Fatigue Detection and Alert System using Non-Intrusive Eye and Yawn Detection** [53], mouth aspect ratio can be written as a mathematical equation, like below:

$$\text{MAR} = \frac{|p_3 - p_{11}| + |p_5 - p_9|}{2 * |p_1 - p_7|} \tag{5.5}$$

where MAR is known as the mouth aspect ratio and $p_1, p_2, p_3, \ldots, p_{12}$ are 12 landmark points associated with outer edge of lip. So, the expressions used in equation 5.5 can be explained as follows:

$|p_3 - p_{11}|$ = Absolute value of Euclidean distance between points p_3 and p_{11}

$|p_5 - p_9|$ = Absolute value of Euclidean distance between points p_5 and p_9

$|p_1 - p_7|$ = Absolute value of Euclidean distance between points p_1 and p_7

In terms of 2D coordinates, the 12 landmark points associated with outer edge of lip can be defined as: $p_1(x_{48}, y_{48})$, $p_2(x_{49}, y_{49})$, $p_3(x_{50}, y_{50})$, $p_4(x_{51}, y_{51})$, $p_5(x_{52}, y_{52})$, $p_6(x_{53}, y_{53})$, $p_7(x_{54}, y_{54})$, $p_8(x_{55}, y_{55})$, $p_9(x_{56}, y_{56})$, $p_{10}(x_{57}, y_{57})$, $p_{11}(x_{58}, y_{58})$ and $p_{12}(x_{59}, y_{59})$. So we can rewrite the equation 5.5 like below:

$$\text{MAR} = \frac{\sqrt{(x_{50} - x_{58})^2 + (y_{50} - y_{58})^2} + \sqrt{(x_{52} - x_{56})^2 + (y_{52} - y_{56})^2}}{2 * \sqrt{(x_{48} - x_{54})^2 + (y_{48} - y_{54})^2}} \tag{5.6}$$

Section C.3.5 explains the method of mouth aspect ratio (MAR) calculation in C# programming language. In the next section, we will explain how our application will do **tiredness detection** by evaluating mouth aspect ratio (MAR) in a continuous monitoring approach.

5.2.3 Tiredness Detection

Before doing tiredness detection, we need to classify the possible states of mouth in our daily life. In general, when a person is working in front of a computer, mouth remains mostly closed and MAR maintains nearly a constant low value. When a person is speaking to another person (e.g, voice calling due to some business purpose), mouth becomes slightly opened and MAR increases slightly. But when a person yawns, mouth becomes widely opened with maximum possible area and MAR increases significantly. So we introduce a threshold value of MAR (which we indicate by $\text{MAR}_{threshold}$) that will help us to identify the current mouth state. If the MAR of a frame exceeds over $\text{MAR}_{threshold}$, the person is considered to be yawning.

$$\text{Mouth State} = \begin{cases} \text{Closed/Speaking}, & \text{MAR} < \text{MAR}_{threshold} \\ \text{Yawn}, & \text{MAR} \geq \text{MAR}_{threshold} \end{cases} \tag{5.7}$$

$\text{MAR}_{threshold}$ is determined by trial and error method, monitoring MAR value of 10 different persons with different mouth states, because we want our system to be able to classify an instance of regular mouth (closed/speaking) and yawn correctly. Table 5.4 shows the list of MAR values of 10 different persons that we measured for consecutive 30 frames.

From table 5.4, we determine the optimal value of $\text{MAR}_{threshold}$ is 0.80. Now the main challenge is to distinguish between normal mouth activity (closed/speaking) and yawn (due to tiredness). Scientists found that the average duration of the yawn is 5 seconds [54]. But the actual duration can vary in between 3.33 to 6.67 seconds as far as the upper and lower margin are concerned. Therefore, in ideal case (30 FPS frame-rate, zero frame-loss and zero noise), we should monitor $100 - 200$ consecutive frames to make a decision of tiredness. In real life environment, it should be sufficient to inspect 100 consecutive frames, because we have possibilities to have frame-loss and noise. So we do prefer to take lower margin as the

Mouth Aspect Ratio (MAR)					
Candidate	**Frame 1**	**Frame 2**	**Frame 3**	**Frame ...**	**Frame 30**
Person $1_{closedmouth}$	0.2869	0.2993	0.2853	...	0.3281
Person $2_{closedmouth}$	0.3103	0.2885	0.3217	...	0.3400
Person $3_{closedmouth}$	0.3537	0.3731	0.3925	...	0.3601
Person $4_{speaking}$	0.4719	0.4963	0.5271	...	0.5695
Person $5_{speaking}$	0.5468	0.5471	0.5265	...	0.5528
Person $6_{speaking}$	0.5832	0.5884	0.5716	...	0.5796
Person 7_{yawn}	0.8190	0.8326	0.8405	...	0.9167
Person 8_{yawn}	0.9381	0.9750	0.9696	...	0.9481
Person 9_{yawn}	0.9573	0.9890	0.9909	...	0.9559
Person 10_{yawn}	0.9892	0.9947	1.0207	...	0.9954

Table 5.4: List of MAR values of 10 different persons

limit. Now, we keep an eye on MAR value being measured for all the captured frames. If measured MAR value is equal or greater than the threshold MAR value (0.80, determined in table 5.4) in a frame, then we mark this frame as a **tired** frame and increase the tired-frame counter by 1. If this counter becomes equal or greater than the threshold frame-number, we declare the individual as **tired** and trigger an alarm. With the help of trained SVM classifier, we can analyze the measured data (mouth aspect ratio values and corresponding tiredness flag information) and plot necessary graph to study the duration and frequency of yawn effectively. In the following section, we discuss the process of **verification of yawn detection** that helps us to validate our proposed algorithm.

Verification of Yawn Detection

Yawning is a common human behavior which consists of an involuntary wide opening of mouth with maximal widening of jaw, then a long & deep inhalation take place through mouth & nose, followed by a slow expiration and a feeling of comfort [54]. From vision system perspective, we can verify a detected yawn in two main steps: in first step, we measure the distance between two center points of inner lip. If this distance is equal or greater than pre-defined threshold distance, then in final step we do skin segmentation in the detected mouth region and expect a circular hole with maximal area [55] - which represents yawning state. Now figure 5.10 shows how to measure the distance between two center points of inner lip.

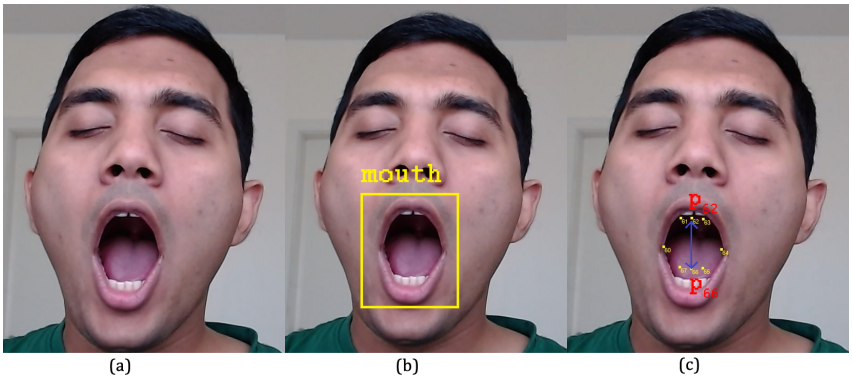


Figure 5.10: Measurement of distance between two center points of inner lip (a) input image, (b) mouth detection, (c) distance measurement between two center points of inner lip

From figure 5.10, we understand that $|p_{62} - p_{66}|$ is the Euclidean distance between two center points of inner lip, where $|p_{62} - p_{66}| = \sqrt{(x_{62} - x_{66})^2 + (y_{62} - y_{66})^2}$

Section C.3.6 explains the method of measuring the distance between two center points of inner lip in C# programming language. Now figure 5.11 shows comparison between obtained outputs after applying skin segmentation algorithm in detected mouth region.

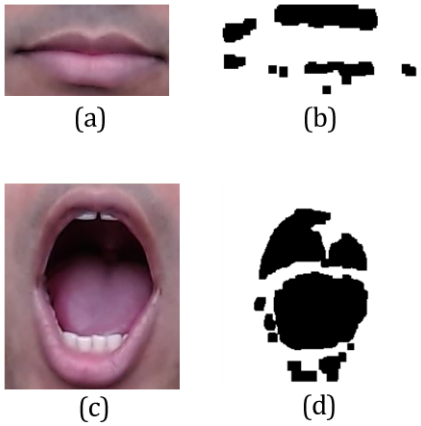


Figure 5.11: Comparison between obtained outputs after applying skin segmentation algorithm in detected mouth region (a) closed mouth, (b) obtained output after applying skin segmentation algorithm on closed mouth, (c) yawning mouth, (d) obtained output after applying skin segmentation algorithm on yawning mouth

Thus we can verify any frame for yawn detection, especially those frames which are already marked as tired-frame (due to high MAR value).

Based on the proposed algorithms in section 5.1.4 and 5.2.3, we expect tiredness detection result based on eye-status and mouse-status respectively. As we recall our secondary task objective, we need to capture behavioral biometrics (keystroke dynamics) when a person is already identified as tired. So we can do **ML model training for keystroke dynamics analysis** for future use.

5.3 ML Model Training for Keystroke Dynamics Analysis

This section proposes an approach of ML model training with keystroke dynamics data when a person is already identified as tired. Figure 5.12 shows the workflow with a block diagram.

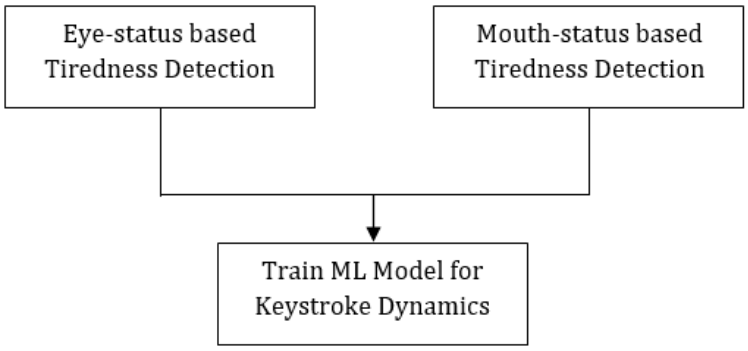


Figure 5.12: Workflow of ML model training with keystroke dynamics

According to our proposed outline in section 4.5, we consider hold-time and release-press-delay as the set of selected features, because there exists a mathematical relationship between this two attributes of keystroke dynamics. So a linear regression ML model should be able to predict release-press-delay using hold-time and release-press-delay data. Figure 5.13 represents the block diagram of the iterative process of ML model development.

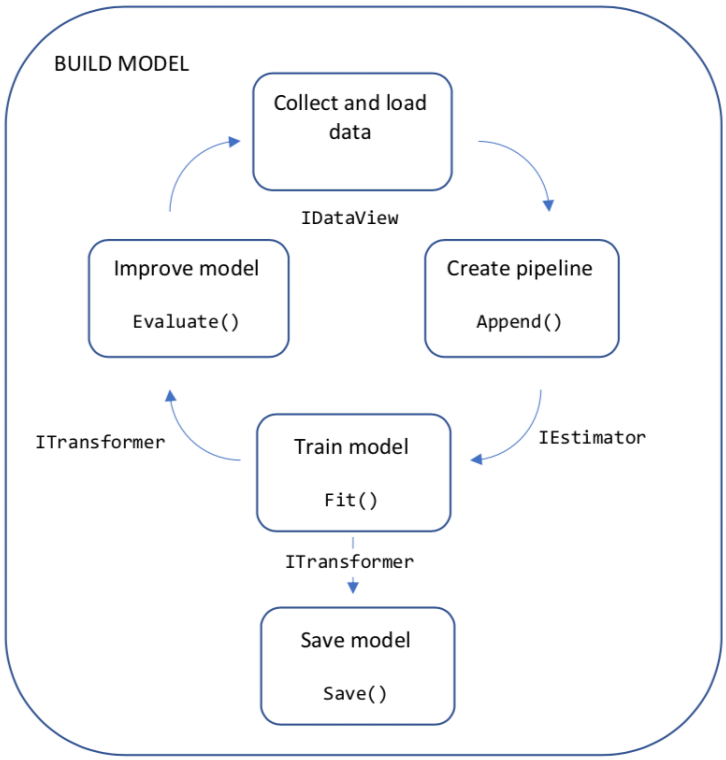


Figure 5.13: Block diagram of iterative process of ML model development [56]

In order to train a ML model, the first step is to prepare dataset of chosen features. Since hold-time and release-press-delay are the set of selected features, we are interested to collect the data when a person is already identified as tired. We have already defined `database.mdb` as our local database where we created a table named `ActivityTable` in which we have two columns named `AverageHoldTime` and `AverageReleasePressDelay`. We use a timer in our code which controls the insertion into the database. Section C.3.7 explains ML model training procedure in step-by-step approach in C# programming language.

5.4 Summary

Based on the study of this chapter, we have discussed about the following things:

- Eye-status based tiredness detection algorithm (using eye aspect ratio)
- Mouth-status based tiredness detection algorithm (using mouth aspect ratio)
- ML model training for keystroke dynamics analysis

Since our primary objective is to design and develop a vision-based system that delivers the state-of-the-art **tiredness assessment** result, so we would like to implement the proposed algorithms (eye-status and mouth-status based) with necessary hardware and software.

Chapter 6

Implementation

This chapter describes the necessary experimental activities in a step-by-step approach in order to achieve the state-of-the-art **tiredness detection** result. Since our target prototype application is a vision system, we divide the implementation activities into two groups:

1. Hardware Setup
2. Software Development

In the following section, we would like to discuss about the necessary **hardware setup** for our application.

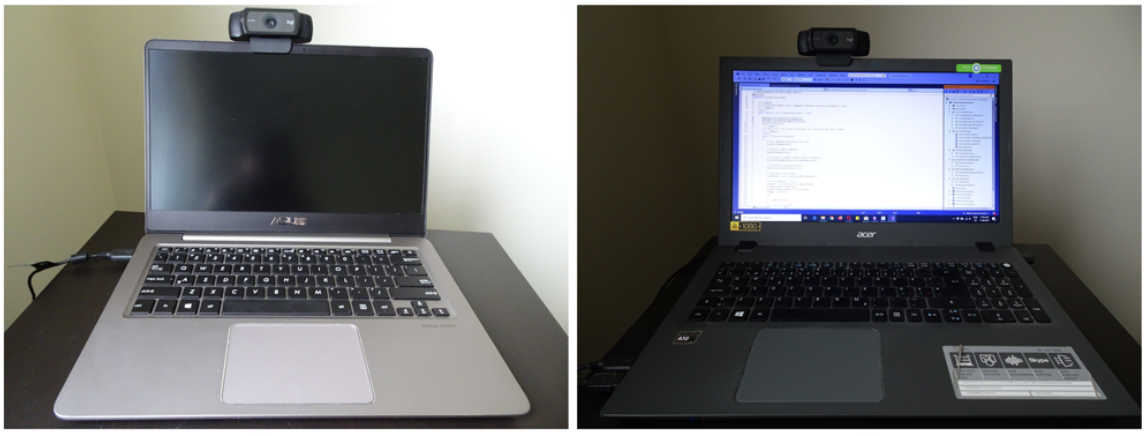
6.1 Hardware Setup

6.1.1 Webcam Configuration

In principle, image sensor is the backbone of our vision system. At the initial phase of the development, we tried to use the built-in camera of the laptop, but unfortunately the result was extremely poor. So it was important for us to install a professional HD quality webcam that could deliver our expected result effectively. Therefore, before we bought the webcam for our prototype application, we considered the following aspects:

- High optical resolution
- High frame rate (e.g, 30 FPS should be standard)
- Ability of auto-focus
- Ability of low-light correction
- Low power consumption
- Affordable price
- Plug-n-play support (USB 3.0 ready support will be preferable)
- Universal clip mount support

So we tried to use Logitech C170 webcam (details specification is available in section B.1) for our system and the outcome was far better than the built-in camera. The result was satisfactory, however in low-light condition the camera failed to identify facial components. So, we bought Logitech C920 webcam (details specification is available in section B.2), did necessary camera parameter calculation (described in section B.2) and installed in our system. Now the result was very good compared to the other webcams that we tried. In addition, the universal clip mount support was useful to mount the webcam to the screen with proper angle and alignment with individual's face. Since this webcam is already recognized as a business-purpose camera (often used for video-calling in workplace), so we understand that our solution is cost-effective, because we can utilize the same webcam for both business-purpose and our application. Figure 6.1 shows the hardware setup for our designed vision system.



(a) (b)

Figure 6.1: Hardware setup (Logitech C920 webcam mounted to two different laptops using universal clip mount support)

6.1.2 System Configuration

Meanwhile, we are highly concerned about the system configuration of our computer (desktop or laptop) as well, because of the following reasons:

- The system is expected to do real-time video streaming for a long session.
- To extract necessary features, real-time image processing ability MUST be supported.
- Multi-threading support is required.
- Installation, improvement and maintenance should not be difficult.
- Migration to other platforms and feature extension should be possible.
- Power consumption and memory consumption should be reasonable.

We could manage to test our prototype application in three different systems (having different configuration). Based on the comparative study of table 3.2, we understand that highly-configured systems (more specifically, RAM should be as maximum as possible) are the good candidates for our prototype application. At the same time, it is important to mention that we executed the necessary testing activities in both day-light and artificial light in a bright room (please see section 3.11, 3.12 and 3.14 for details). So, it is expected that the prototype application should be tested in a sufficient lighting condition.

6.1.3 Setup Activities

In principle, hardware setup activities can be summarized in a step-by-step approach by the following points:

1. The webcam must be mounted on top of the laptop screen with the help of universal clip mount support. **USB** 3.0 **port** should be used in order to ensure better connectivity and speed.
2. The subject (individual) should be located in front of the laptop, maintaining a standard sitting posture and the distance between the webcam & the subject's eyes should be in between $40 - 60$ cm. The subject's face and facial components must be visible to the mounted webcam.
3. The prototype application should be started once step 1 and 2 are accomplished.

Figure 6.2: USB 3.0 port in laptop where webcam should be connected

Now we will discuss about the necessary **software development** activities in details.

6.2 Software Development

The entire software is developed in C# programming language by using .NET framework (version 4.7.2). This development platform is chosen because of the following reasons:

1. The initial target operating system (OS) is Windows. So necessary drivers should be available over internet.
2. A cross platform .NET wrapper of OpenCV library (known as EmguCV) has necessary implementations of most of the OpenCV functions.
3. Built-in C# namespaces (libraries) are already rich in various useful features (e.g, forms, graphs, charts etc.) to analyze the measurements and easy to develop & debug. In case, if built-in libraries are not sufficient, it is possible to install additional NuGet packages at any time in order to develop application with required features.
4. Migration process to other object-oriented-programming (OOP) platforms (for example, Java) will be hassle-free for developers.
5. For future improvement and maintenance, engineers will find it easy to understand the syntax and logic behind the code.

6.2.1 Vision System Architecture

In the context of our prototype application, first of all, it is important to define the vision system architecture with necessary elements. So we make an attempt to draw a logical diagram in order to understand how the vision system will interact with the user, how the collaboration between each process will work, how the tiredness detection algorithm will execute and how the status reports will be generated and so on. Let's start to design the vision system architecture for our application.

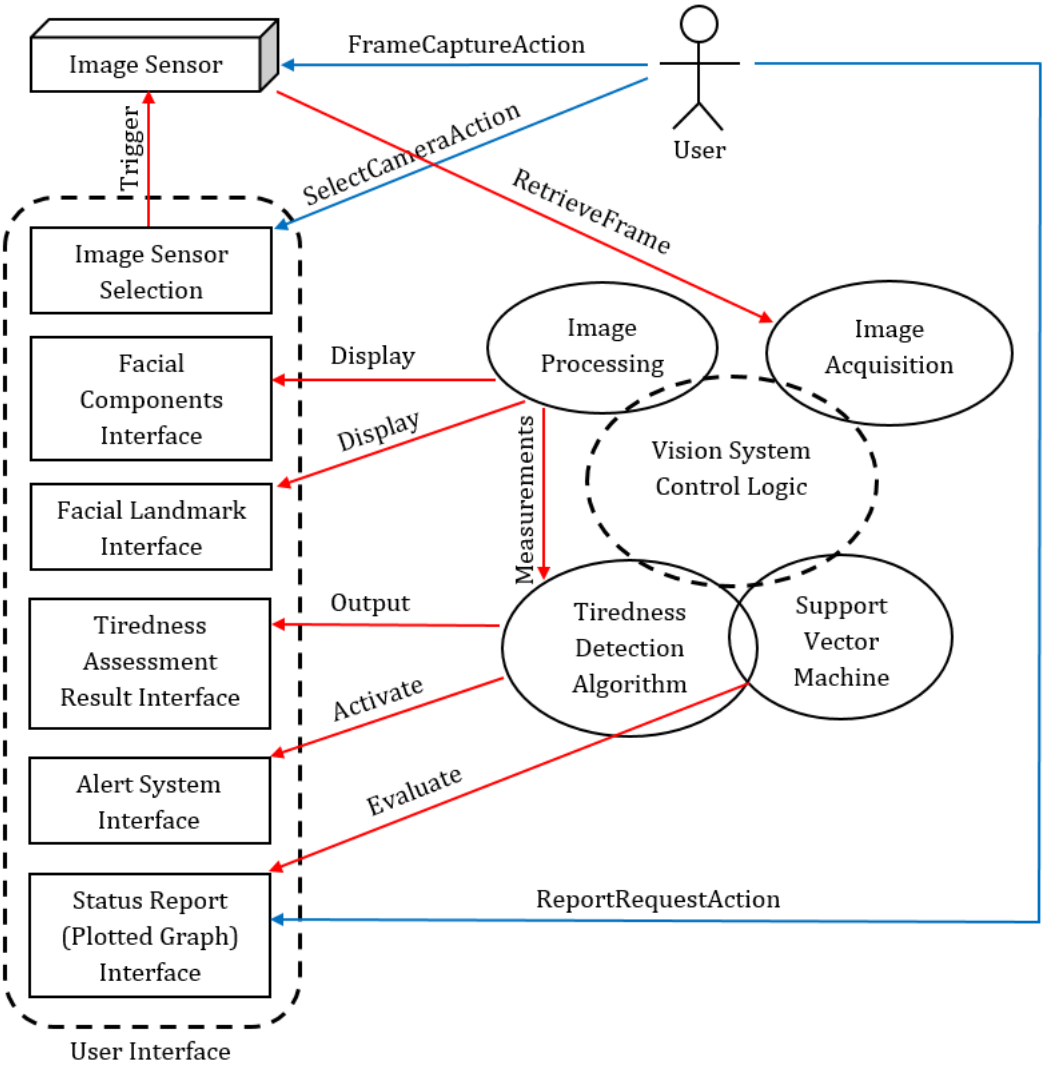


Figure 6.3: Overview of vision system architecture

Figure 6.3 is the overview of vision system architecture designed for this thesis. The dashed rounded rectangle is the **User Interface** and its interior rectangular boxes represent different interface modules that interact with the user. The dashed elliptical shape is the **Vision System Control Logic** which consists of different control modules that collaborate with different interface modules. Thick blue lines represent different interactions associated with the user. On the other hand, thick red lines represent different interactions between different system modules. Image sensor is represented by a 3D rectangular box.

When the application is executed, the image sensor starts to capture the raw image data of the user. **Image Acquisition** module is responsible to retrieve the frame from the image sensor and collaborates with **Image Processing** module for each frame. Image processing module

is a part of vision system control, which applies necessary algorithms for facial component and landmark detection. Then it propagates the obtained information to the respective interface modules. At the same time, it provides necessary measurement information to the **Tiredness Detection Algorithm** module for further processing. Tiredness detection algorithm module is responsible for assessing the tiredness level and delivering the result in real-time. If the user is identified as tired, it activates the alert system immediately. The status reports (plotted graphs) are available only if the application is not running. **Support Vector Machine** module is responsible to analyze the measurements and predict the user status as well. Table 6.1 provides the list of necessary libraries and packages that are used.

Library and Package	**Field of Application**
`Accord`	To implement support vector machine (SVM) features
`DirectShowLib`	To fetch hardware (image sensor) information
`EmguCV`	To call OpenCV functions from .NET compatible platform
`Microsoft.ML`	To support ML model training, prediction and evaluation
`System.Data.OleDb`	To perform database related activities
`System.Diagnostics`	To use stopwatch (timer)
`System.Drawing`	To access different drawing objects (point, rectangle, circle etc.)
`System.IO`	To execute file related operations
`System.Media`	To play the alarm
`System.Threading`	To facilitate multi-threading support
`System.Windows.Forms`	To create the windows forms

Table 6.1: Libraries and packages used

6.2.2 Prototype Application Architecture

Before creating the project in our selected IDE, we study our vision system architecture (already described in section 6.2.1) very carefully and design the prototype application architecture with necessary functional blocks. Figure 6.4 is the overview of prototype application architecture used for our research work.

The rectangular boxes represent the necessary functional blocks to be implemented in our project. Each black arrow represents the hierarchical-calling relationship between two connected blocks. In case, if a single block calls multiple blocks then the execution takes place from left to right. The constructor of **TirednessAssessment** class initiates the test environment with required parameters (e.g, available camera-list, camera-capture property, cascade classifiers, landmark model etc.) and **ProcessFrame** method is invoked every time a single frame is captured. This method is responsible to call two important methods, which are - **FaceIdentification** and **LandmarkIdentification**. In principle, **LandmarkIdentification** method drives the tiredness evaluation procedure by calling **EyeTirednessMeasurement** and **MouthTirednessMeasurement** methods through the respective aspect ratio measurement methods. The final two blocks (**EyeStatusReport** and **MouthStatusReport**) represent the feature of necessary status reports. In reality, we have divided them into multiple functions in order to analyze the measured data and show graphs with proper legends. It is expected that, all these functional blocks should execute continuously as long as the application is running (not stopped by the user) and the system MUST deliver the state-of-the-art result in real time.

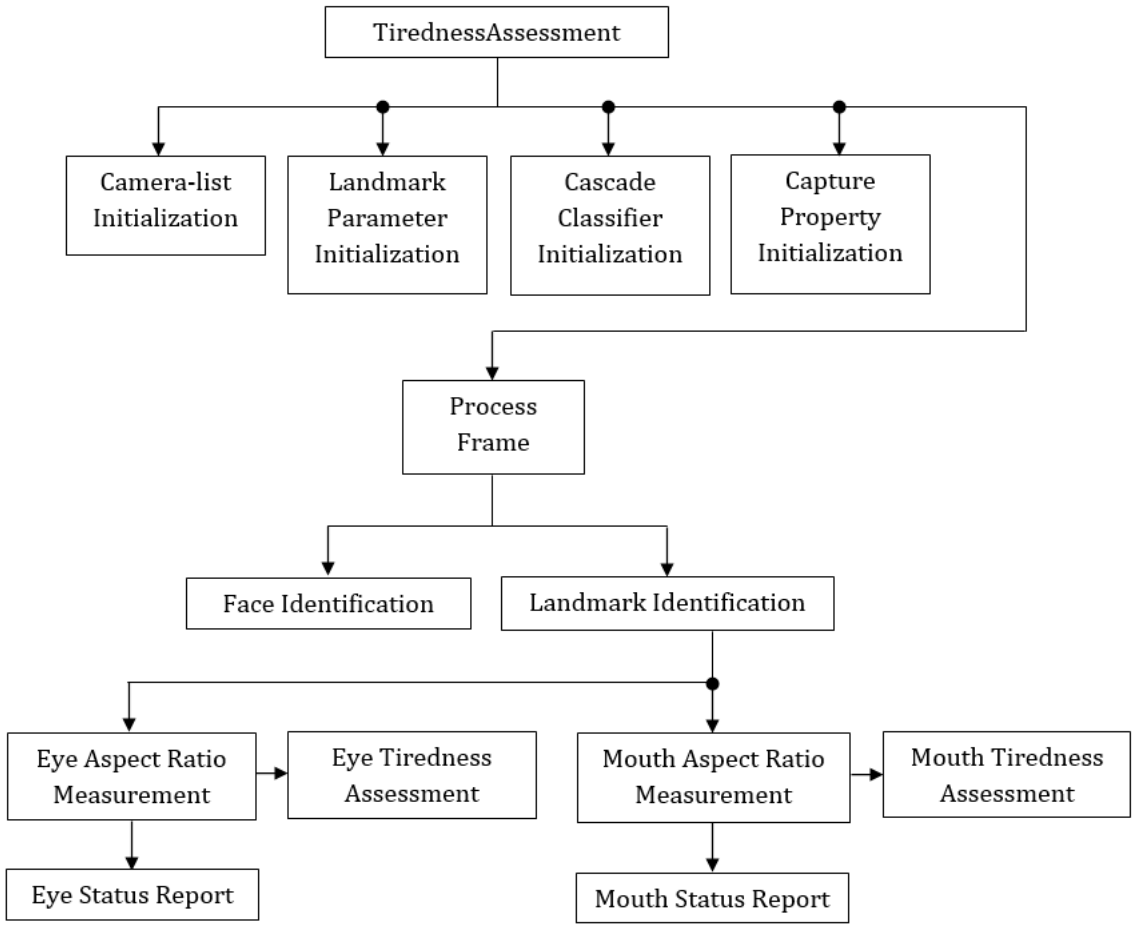


Figure 6.4: Overview of prototype application architecture

6.2.3 User Interface

As we study the prototype application architecture (already discussed in section 6.2.2), it is clear that our application will be a **Windows Forms Application (.NET Framework)** where the **main( )** function will instantiate the user interface. So we create a WFA project named `TirednessAssessment` which automatically generates a form with the same name. In the context of our application, we need few more forms that will help to analyze the result as well. Table 6.2 provides the list of necessary windows forms used in our prototype application.

Windows Form Name	Functionality
`TirednessAssessment`	To display the main user interface
`EARHistoryGraph`	To display current session's history of eye aspect ratio (EAR) vs frame-number of subject in a plotted graph
`EyeStatusGraph`	To display current session's eye status (eye-state vs frame-number) of subject using trained SVM classifier
`MARHistoryGraph`	To display current session's history of mouth aspect ratio (MAR) vs frame-number of subject in a plotted graph
`MouthStatusGraph`	To display current session's mouth status (mouth-state vs frame-number) of subject using trained SVM classifier

Table 6.2: Windows forms used

Figure 6.5 shows the user interface in development environment with necessary annotations. Figure 6.6, 6.7 and 6.8 show the user interface for different cases in real-time.

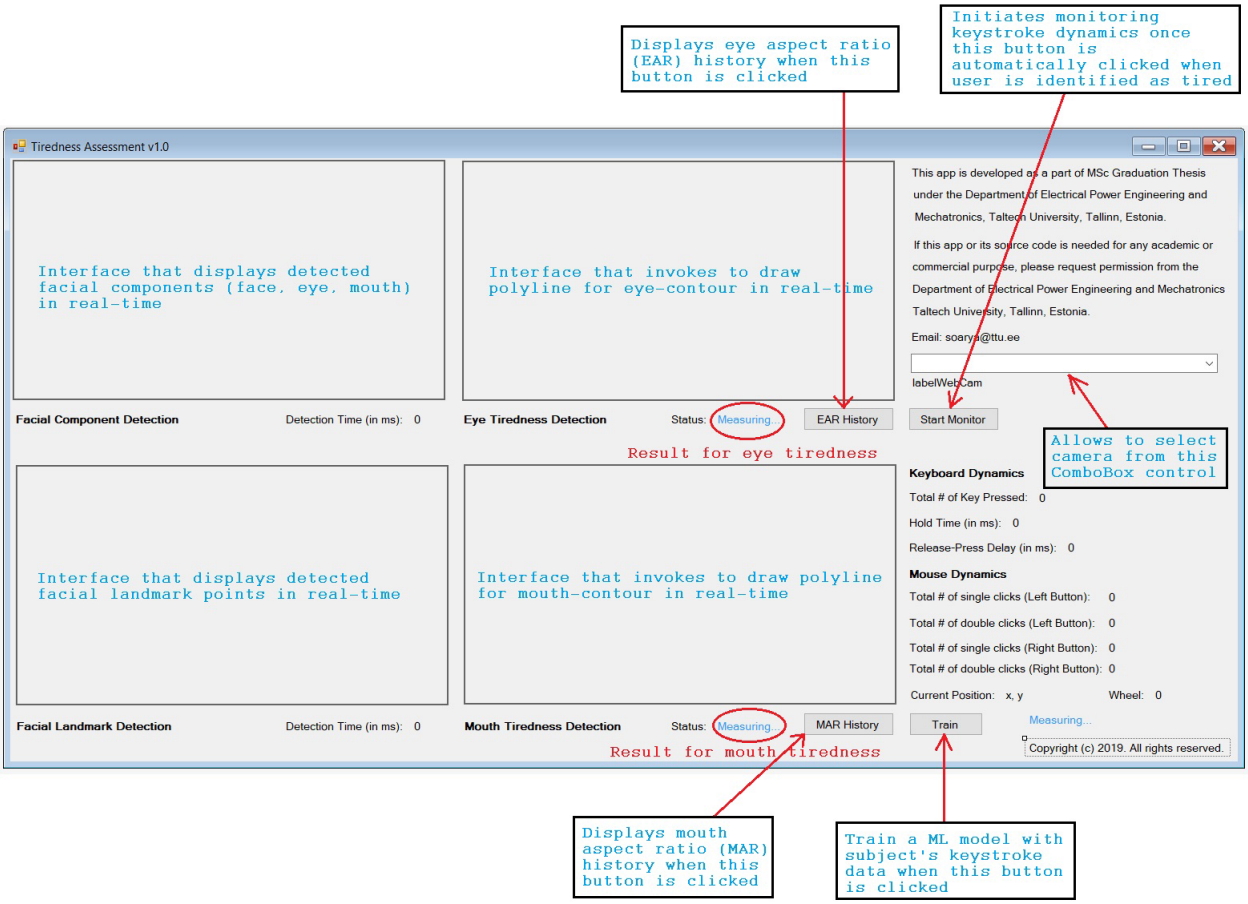


Figure 6.5: User interface in development environment

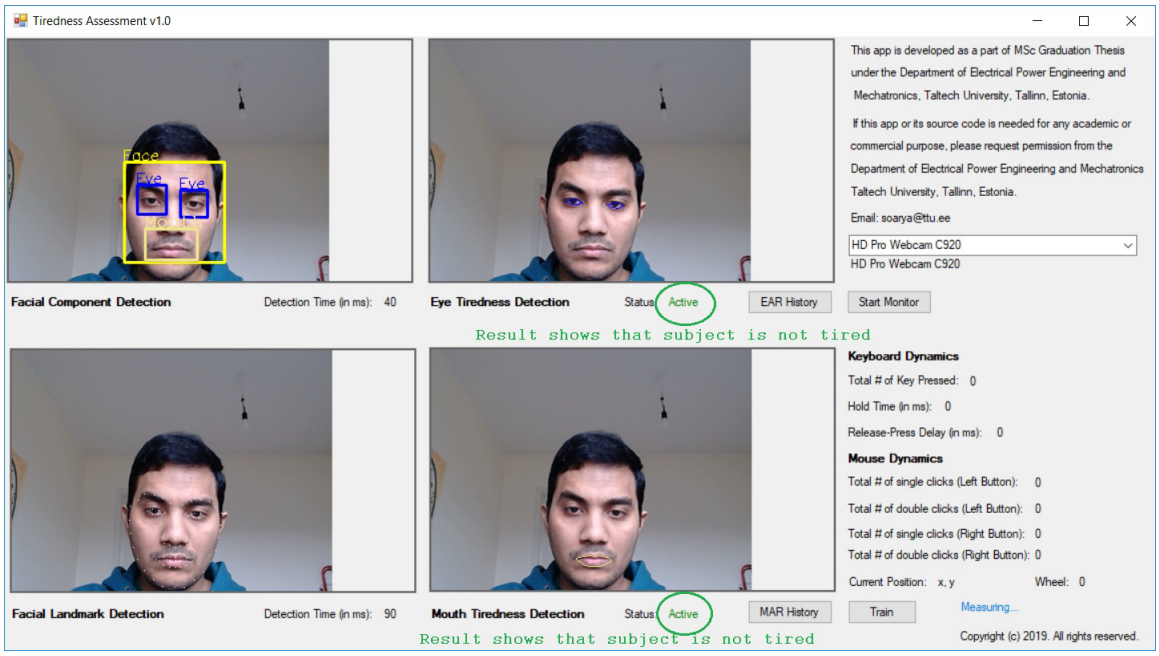


Figure 6.6: User interface in real-time when subject is not tired

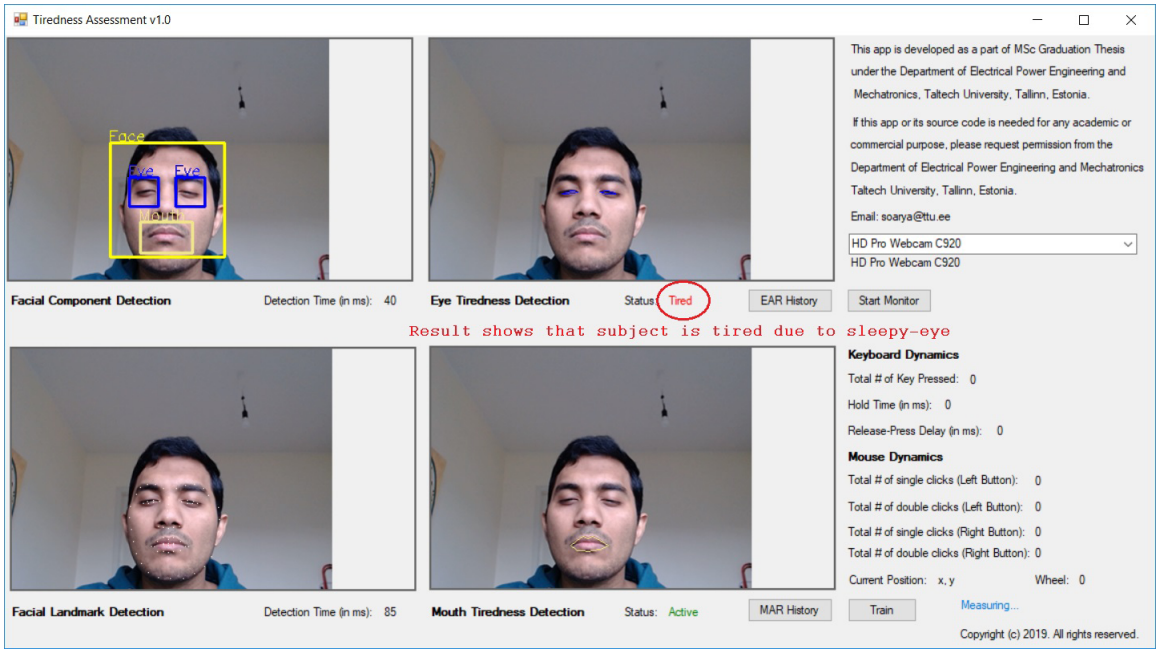


Figure 6.7: User interface in real-time when subject is tired due to sleepy-eye

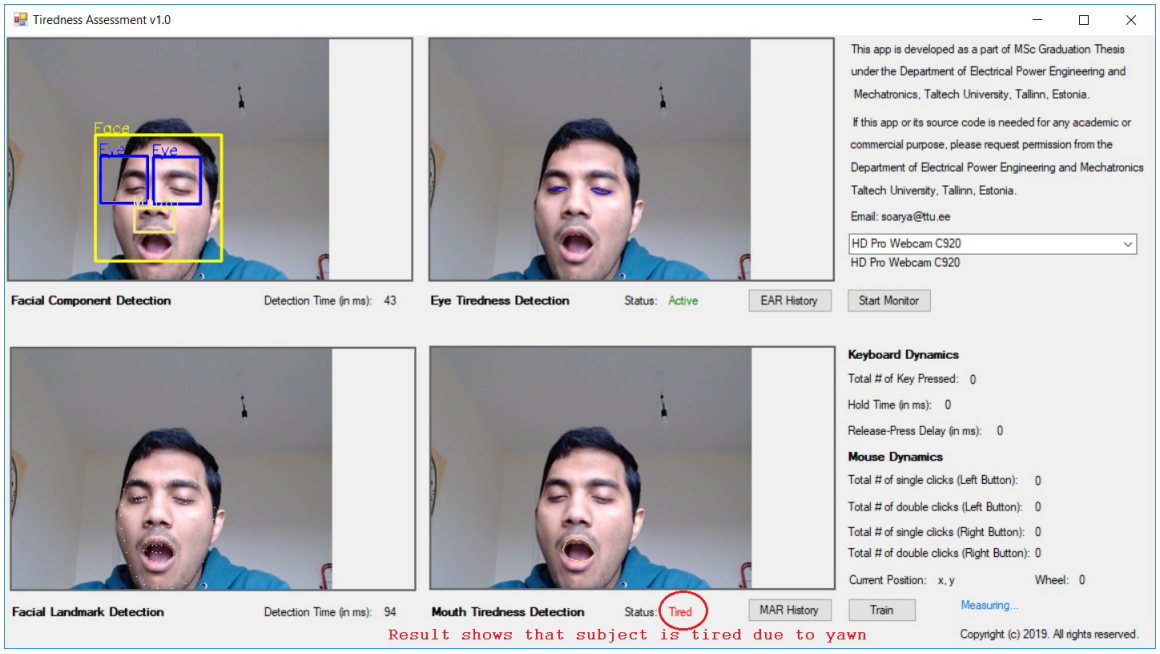


Figure 6.8: User interface in real-time when subject is tired due to yawn

6.3 Summary

The proposed implementation is cost-effective, because only one webcam is used and no additional hardware are required. Libraries & packages are selected based on the flexibility and possibility of future development. Standard C# programming rules are strictly maintained throughout the source code and proper documentation is done. In a summary, the implementation is done in an iterative approach and sufficient testing & optimization are performed.

Chapter 7

Results

The results are presented by both eye-status based and mouth-status based tiredness detection algorithm. Necessary measurements are done in real-time and trained SVM classifiers are used to deliver the state-of-the-art results. Please note that, before testing any of the tiredness detection algorithm, it must be made sure that all the necessary pre-requisite activities are done as mentioned in section 6.1.3.

7.1 Eye-status based Tiredness Detection

In order to test the eye-status based tiredness detection algorithm, we consider 2 subjects (1 with and 1 without glasses) in a bright room with constant light. For 1st subject, we capture about 330+ frames in real-time and apply our algorithm. Table 7.1 shows the measured EAR data (important frames only) of subject 1.

Eye Aspect Ratio (EAR) Data (Subject 1)					
Frame Nr.	**Left EAR**	**Right EAR**	**Frame Nr.**	**Left EAR**	**Right EAR**
1	0.1930	0.2011	271	0.2457	0.2337
2	0.2170	0.2205	272	0.1920	0.1871
3	0.2115	0.2064	273	0.1750	0.1669
4	0.1823	0.1829	274	0.1807	0.1712
5	0.2011	0.1926	275	0.1600	0.1511
6	0.2088	0.2067	276	0.1942	0.1897
7	0.2873	0.2854	277	0.1929	0.1812
8	0.3257	0.3148	278	0.1806	0.1665
9	0.2974	0.3097	279	0.1822	0.1706
10	0.3367	0.3258	280	0.1846	0.1705
11	0.3325	0.3422	281	0.1990	0.1831
12	0.3306	0.3233	282	0.1745	0.1648
13	0.3059	0.3040	283	0.1777	0.1651
14	0.1697	0.1750	284	0.1697	0.1553
...	...	...	...	...	...

Table 7.1: Eye aspect ratio (EAR) data (Subject 1)

From frame 272 to frame 283, we can see that both left EAR and right EAR values are below the threshold EAR value - which matches with our proposed algorithm's condition of micro-sleep. Figure 7.1 shows the mentioned frames along with their respective EAR values.

Figure 7.1: Tiredness detection based on eye-status (Subject 1)

In the plotted graph of EAR vs frame-number at figure 7.2, we notice that EAR started to fall below threshold EAR value from frame 272 and remained low up to next 30 frames. So all these frames are marked as **tired-frame** by the application.

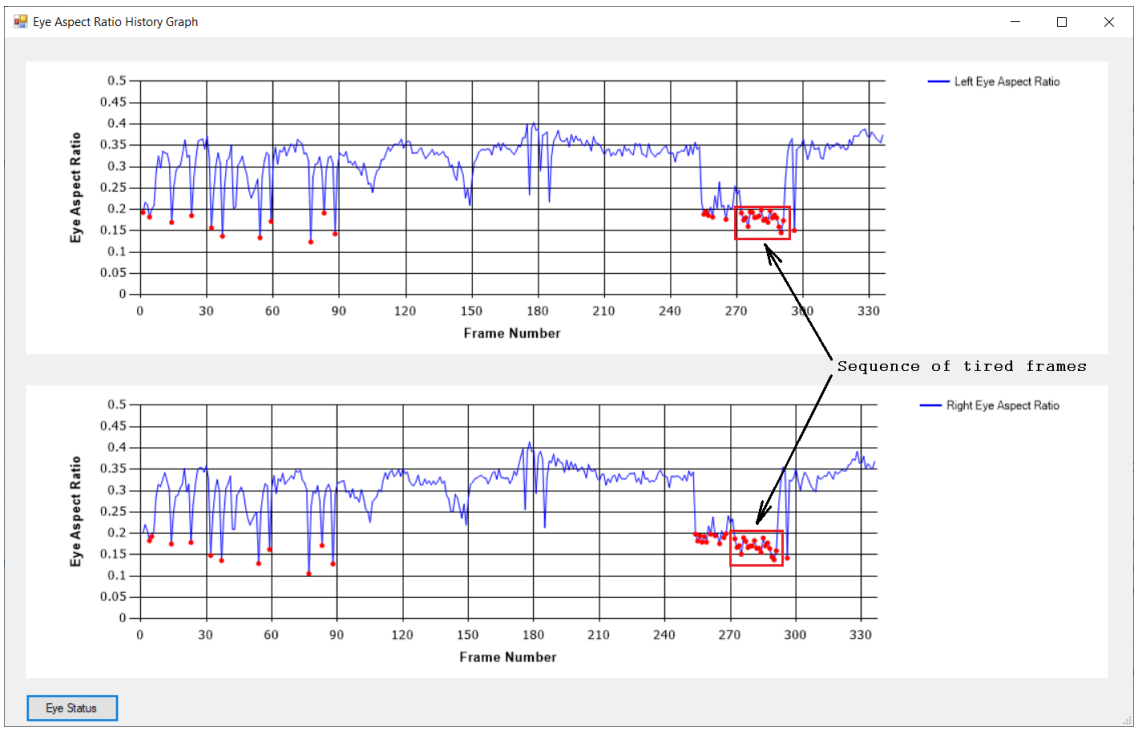


Figure 7.2: Eye aspect ratio history graph (Subject 1)

Now we applied a trained SVM classifier to analyze this data to know if there was any micro-sleep or not. Figure 7.3 shows the plotted graph for details. Please note that, 0 refers to "closed eye" and 1 refers to "open eye" in the eye status graph.

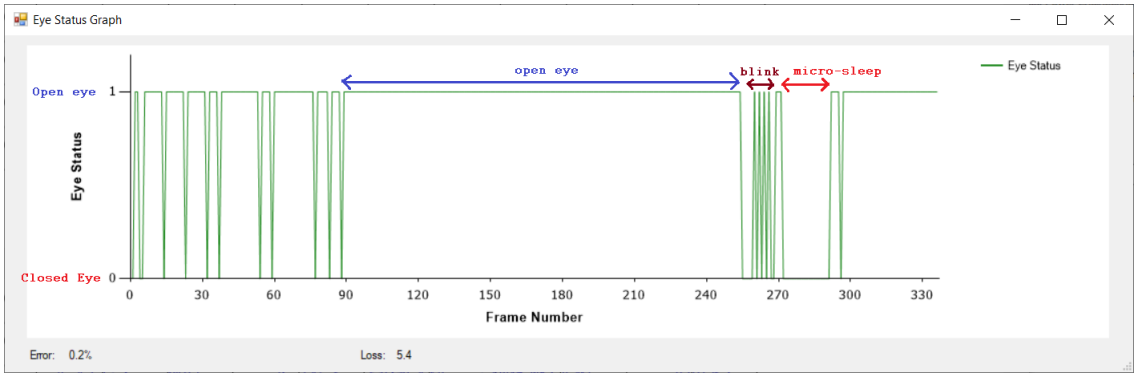


Figure 7.3: Eye status graph (Subject 1)

From figure 7.3, we notice all possible eye states including open-eye, closed-eye and eye-blink. Our algorithm ignored the eye-blink activity, but it could identify the clear presence of micro-sleep successfully in real time. And the aftermath was really interesting, because an immediate eye-opening activity from the subject was witnessed. The automated alarm (triggered by the application itself) might be responsible for this sudden change.

Similarly, for 2nd subject, we capture about 340+ frames in real-time and apply our algorithm. Table 7.2 shows the measured EAR data (important frames only) of subject 2.

Eye Aspect Ratio (EAR) Data (Subject 2)					
Frame Nr.	**Left EAR**	**Right EAR**	**Frame Nr.**	**Left EAR**	**Right EAR**
1	0.2949	0.2857	280	0.1771	0.1744
2	0.2536	0.2746	281	0.1699	0.1727
3	0.2584	0.2835	282	0.1686	0.1689
4	0.2717	0.3013	283	0.1874	0.1748
5	0.2888	0.3044	284	0.1745	0.1735
6	0.2924	0.3013	285	0.1681	0.1713
7	0.2975	0.3159	286	0.1756	0.1741
8	0.3073	0.3121	287	0.1631	0.1653
9	0.3015	0.3040	288	0.1869	0.1763
10	0.2958	0.3101	289	0.1605	0.1676
11	0.2900	0.3096	290	0.1664	0.1689
12	0.2785	0.2946	291	0.1579	0.1528
13	0.2862	0.3000	292	0.1753	0.1829
14	0.2822	0.3040	293	0.1847	0.1811
...	...	...	...	...	...

Table 7.2: Eye aspect ratio (EAR) data (Subject 2)

From frame 280 to frame 291, we can see that both left EAR and right EAR values are below the threshold EAR value - which matches with our proposed algorithm's condition of micro-sleep. Figure 7.4 shows the mentioned frames along with their respective EAR values.

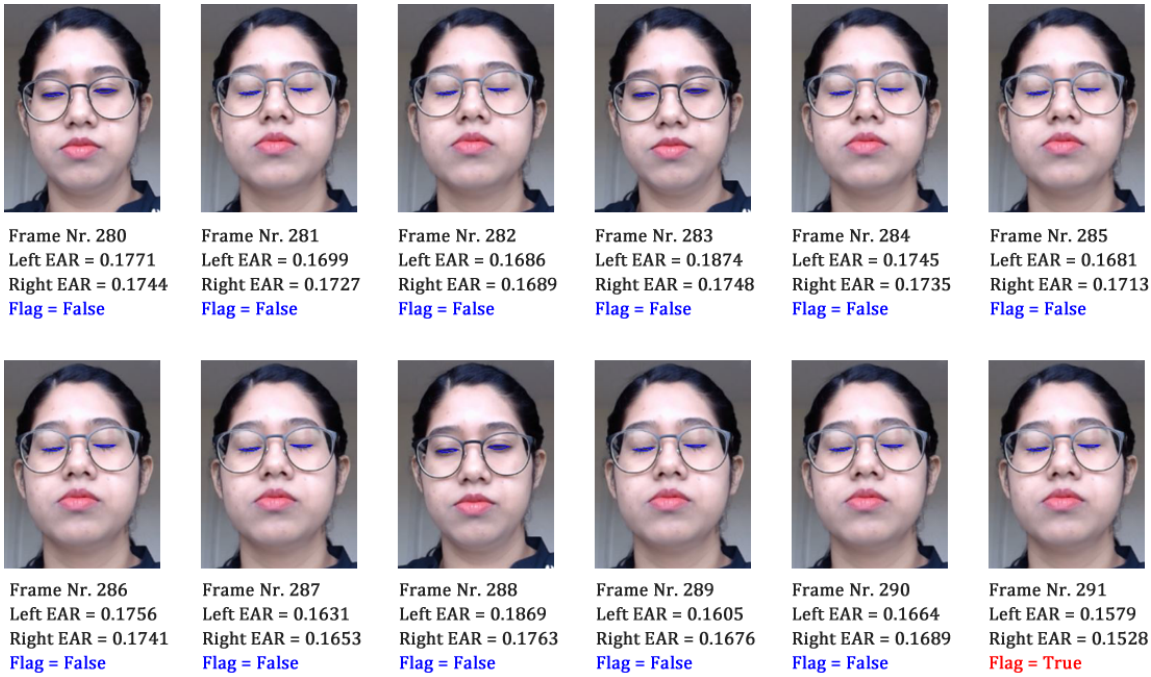


Figure 7.4: Tiredness detection based on eye-status (Subject 2)

In the plotted graph of EAR vs frame-number at figure 7.5, we notice that EAR started to fall below threshold EAR value from frame 280 and remained low up to next 20 frames. So all these frames are marked as **tired-frame** by the application.

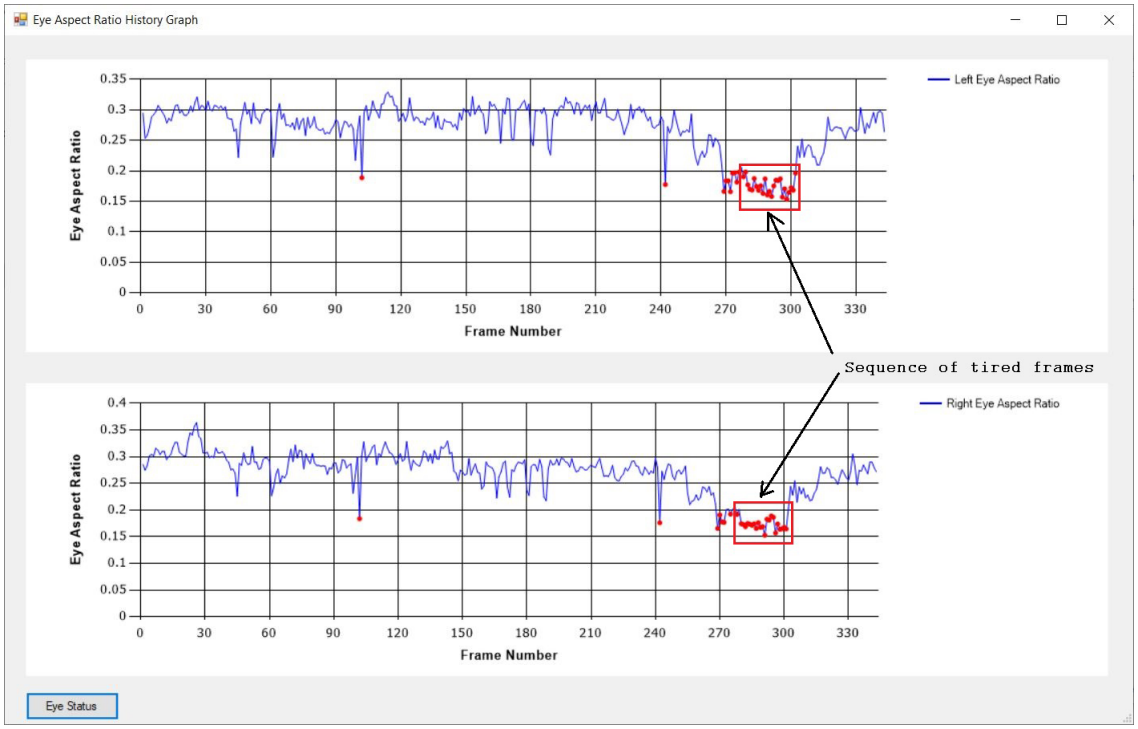


Figure 7.5: Eye aspect ratio history graph (Subject 2)

Like previous inspection, we applied a trained SVM classifier to analyze this data to know if there was any micro-sleep or not. Figure 7.6 shows the plotted graph for details. Please note that, 0 refers to “closed eye” and 1 refers to “open eye” in the eye status graph.

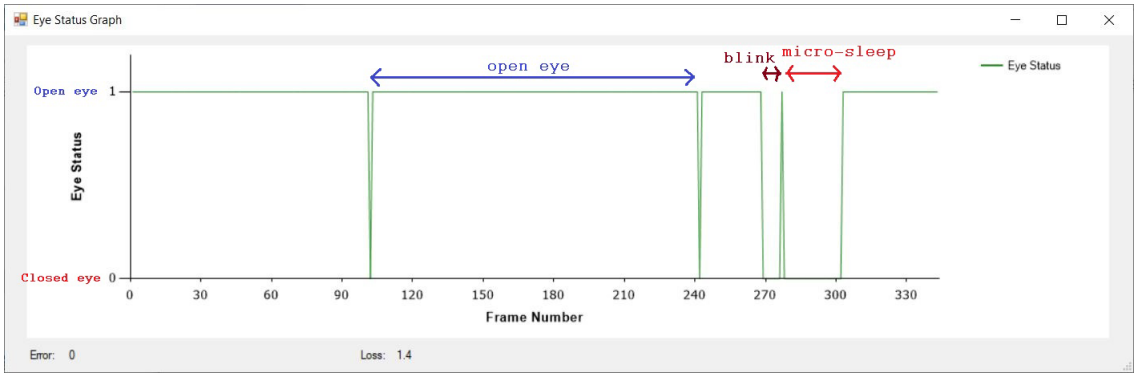


Figure 7.6: Eye status graph (Subject 2)

From figure 7.6, we notice all possible eye states including open-eye, closed-eye & eye-blink. Our system ignored the eye-blink activity, but it could identify the clear presence of micro-sleep successfully in real time and it triggered the automated alarm immediately.

7.2 Mouth-status based Tiredness Detection

In order to test the mouth-status based tiredness detection algorithm, we consider the same subjects in the same lighting condition (as described in section 7.1). For 1st subject, we capture about 670+ frames in real-time and apply our algorithm. Table 7.3 shows the measured MAR data (important frames only) of subject 1.

Mouth Aspect Ratio (MAR) Data (Subject 1)					
Frame Nr.	**MAR**	**Frame Nr.**	**MAR**	**Frame Nr.**	**MAR**
445	0.8624	485	0.7720	525	0.7986
446	0.8715	486	0.8220	526	0.8559
447	0.9093	487	0.8350	527	0.8519
448	0.8448	488	0.8690	528	0.8813
449	0.8209	489	0.7844	529	0.8474
...	...	...	...	...	...
465	0.9001	505	0.7961	545	0.8261
466	0.8952	506	0.7356	546	0.8225
467	0.8363	507	0.8578	547	0.7835
468	0.8905	508	0.8343	548	0.8472
469	0.7047	509	0.8731	549	0.7853
...	...	...	...	...	...

Table 7.3: Mouth aspect ratio (MAR) data (Subject 1)

From frame 445 to frame 560, we can see that most of the MAR values (except few noises) are above the threshold MAR value - which matches with our proposed algorithm's condition of yawn. Figure 7.7 shows the mentioned frames along with their respective MAR values. In the plotted graph of MAR vs frame-number at figure 7.8, we notice that MAR started to rise above threshold MAR value from frame 445 and remained high up to next 100+ frames (except few noises). So all these frames are marked as **tired-frame** by the application. Then we applied a trained SVM classifier to predict if there was any yawn or not. In the plotted graph of figure 7.9 0 refers to “closed-mouth/speaking” and 1 refers to “yawn”.

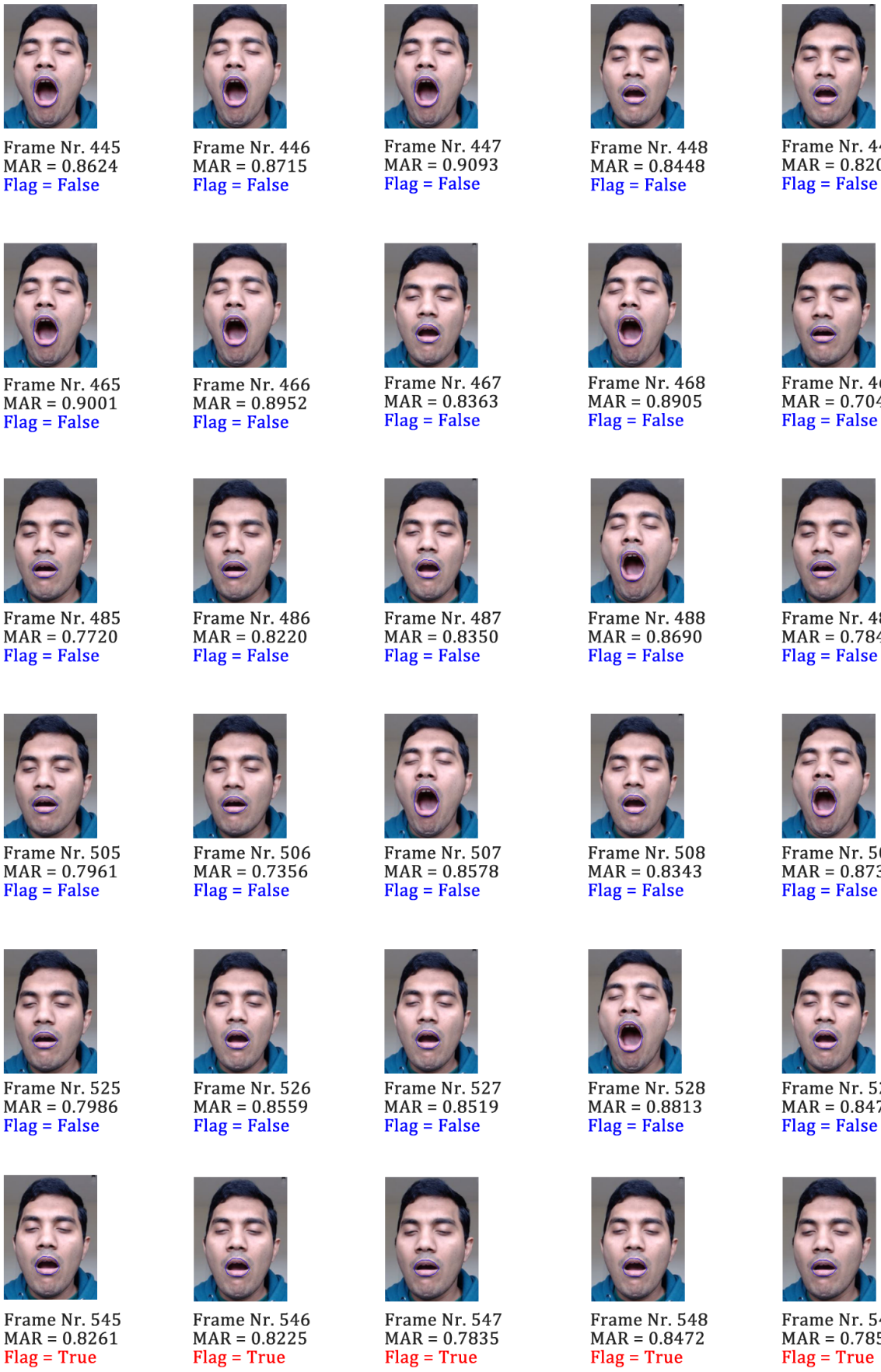


Figure 7.7: Tiredness detection based on mouth-status (Subject 1)

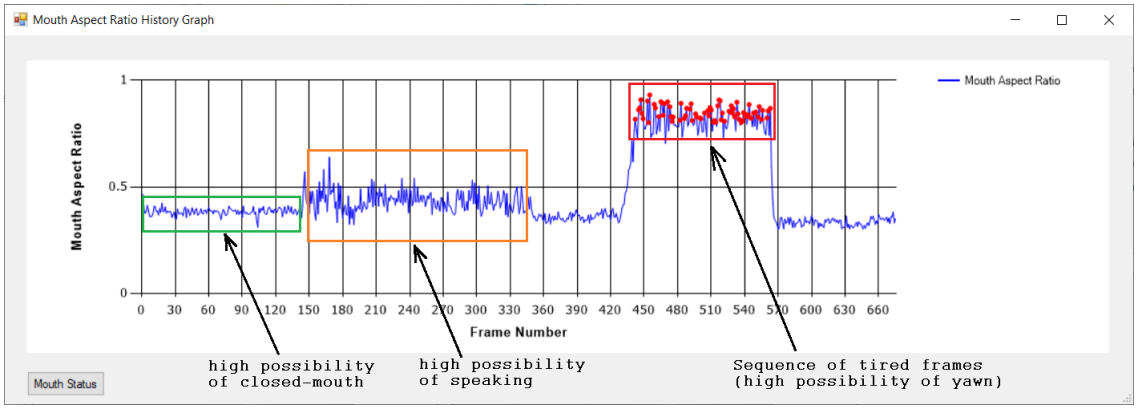


Figure 7.8: Mouth aspect ratio history graph (Subject 1)

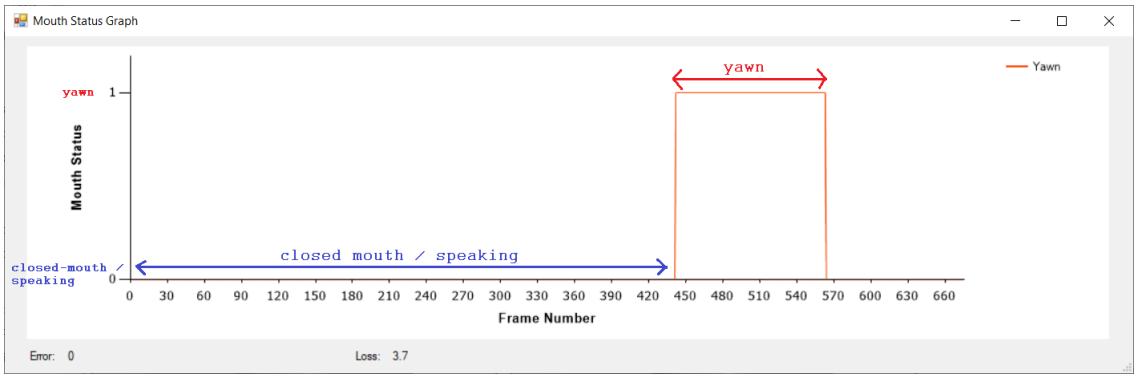


Figure 7.9: Mouth status graph (Subject 1)

Similarly, for 2nd subject, we capture about 490+ frames in real-time and apply our algorithm. Table 7.4 shows the measured MAR data (important frames only) of subject 2.

Mouth Aspect Ratio (MAR) Data (Subject 2)					
Frame Nr.	**MAR**	**Frame Nr.**	**MAR**	**Frame Nr.**	**MAR**
320	0.9515	360	0.9590	400	0.9372
321	0.9676	361	0.8723	401	0.9549
322	0.9650	362	0.9245	402	0.9191
323	0.8906	363	0.9183	403	0.9311
324	0.9370	364	0.9475	404	0.8552
...	...	...	...	...	...
340	0.9310	380	0.9433	420	0.8780
341	0.9411	381	0.8883	421	1.0067
342	0.9260	382	0.9354	422	0.8927
343	0.9233	383	0.9185	423	0.9225
344	1.0499	384	0.9304	424	0.8714
...	...	...	...	...	...

Table 7.4: Mouth aspect ratio (MAR) data (Subject 2)

From frame 320 to frame 428, we can see that all of the MAR values are above the threshold MAR value - which matches with our proposed algorithm's condition of yawn. Figure 7.10 shows the mentioned frames along with their respective EAR values.

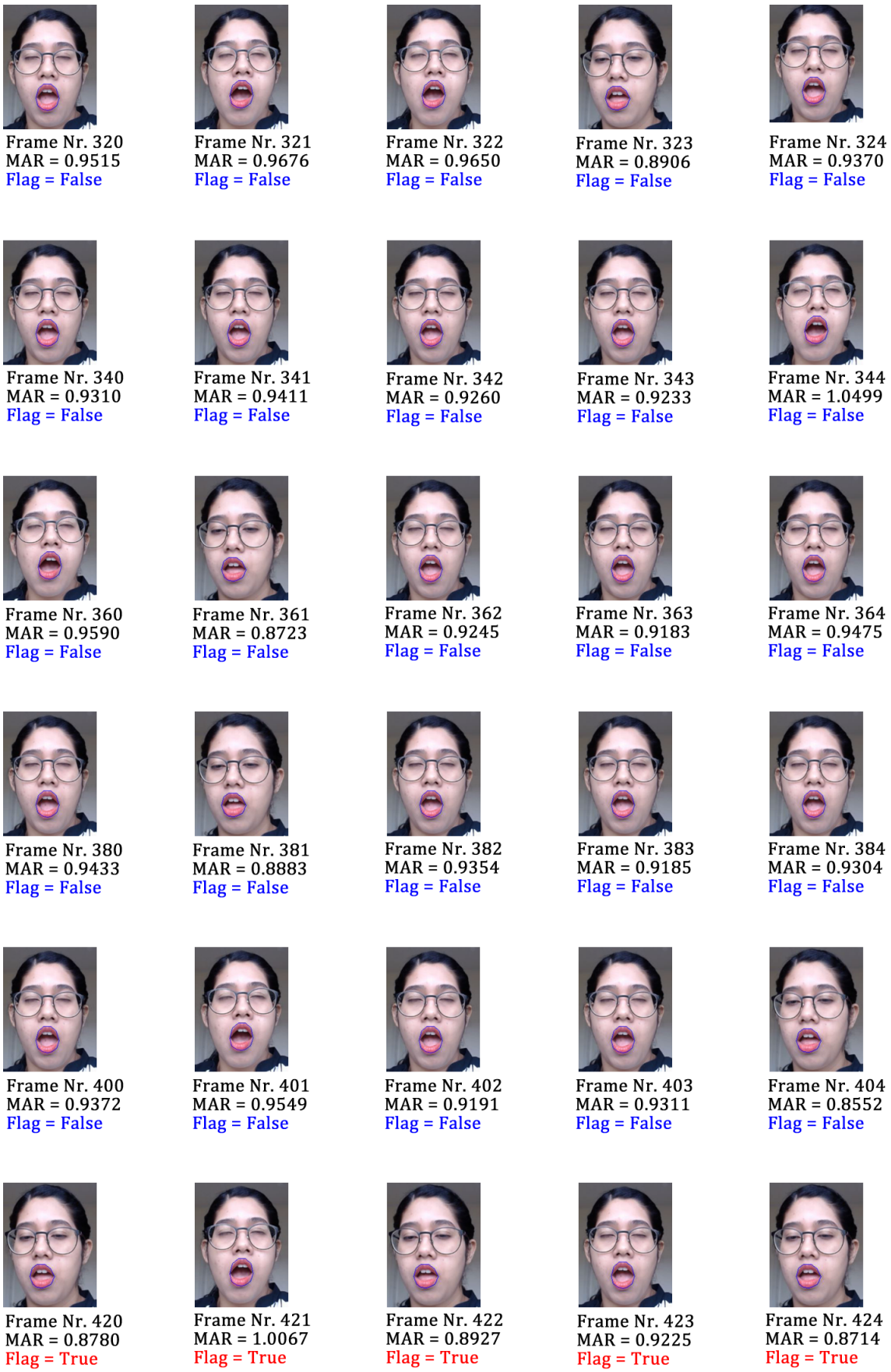


Figure 7.10: Tiredness detection based on mouth-status (Subject 2)

In the plotted graph of MAR vs frame-number at figure 7.11, we notice that MAR started to rise above threshold MAR value from frame 320 and remained high up to next 100+ frames. So all these frames are marked as **tired-frame** by the application.

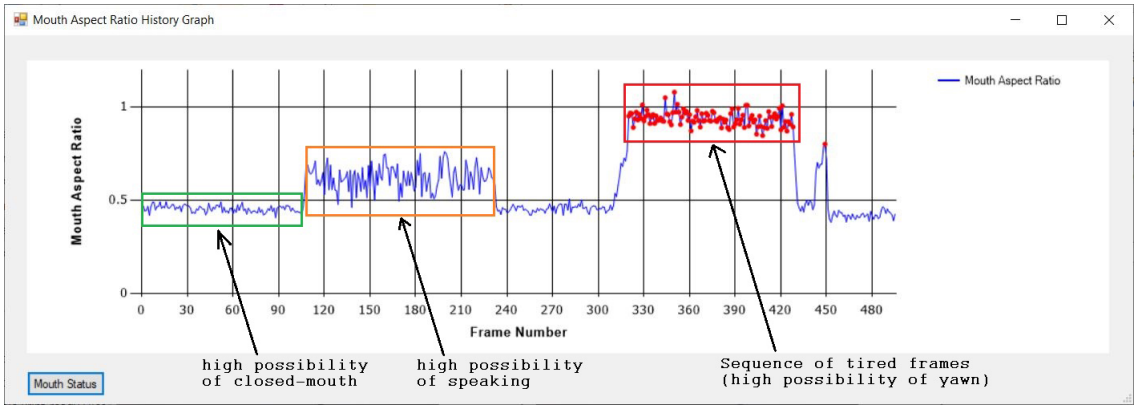


Figure 7.11: Mouth aspect ratio history graph (Subject 2)

Now we applied a trained SVM classifier to predict if there was any yawn or not. In the plotted graph of figure 7.12 0 refers to "closed-mouth/speaking" and 1 refers to "yawn".

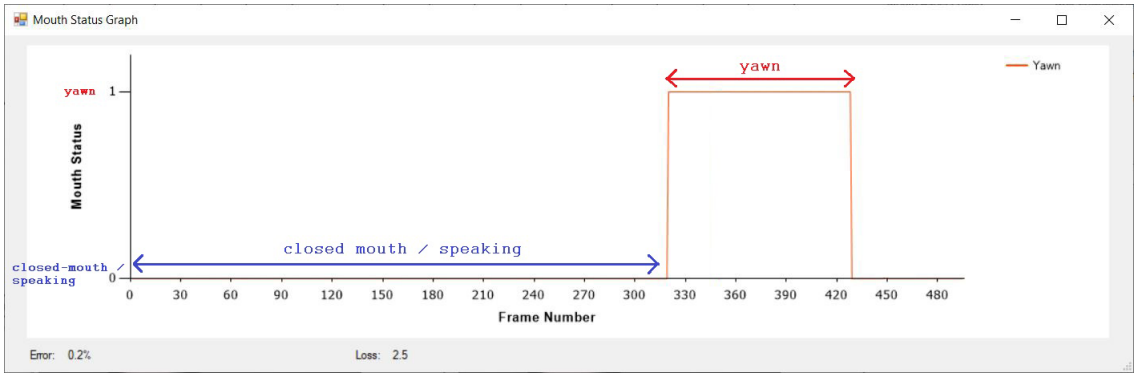


Figure 7.12: Mouth status graph (Subject 2)

7.3 Summary

Based on the obtained test results, it is clear that the system delivers the state-of-the-art **tiredness assessment** result under the considered condition. It is important to mention that, low light will not cause any adverse effect, because it is possible to install IR camera to the system and the application allows to select the camera using ComboBox control. Use of glasses by one subject also does not hinder the performance of the system, because the eye region is constantly visible to the camera. The classification error was in between $0 - 0.2\%$ - which justifies that the SVM classifiers were well-trained to predict the results. In a summary, the test results of the system were satisfactory, even though there were very few random noises and false-positives observed during the test time.

Chapter 8

Discussion

In this chapter, we define, in brief, concept of confusion matrix to evaluate our system's performance. In section 8.2, a comparative study with other existing methods is done. Finally, we discuss about the limitation of the system and future improvements.

8.1 System Performance Analysis

Necessary tests with both subjects were done in the same room in the presence of day-light and artificial light. Now, it is important to analyze the observed results and compare with the expected results. Hence, a confusion matrix can help to evaluate our system's performance. Basically, it presents a matrix-formatted table that displays the frequency distribution of the variables where each column represents the predicted class and each row represents the actual class. In the context of our system, there exist four possibilities which are as follows:

- True positive: Predicted class is “tired” and actual class is also “tired”
- False positive: Predicted class is “tired” but actual class is “not tired”
- True negative: Predicted class is “not tired” and actual class is also “not tired”
- False negative: Predicted class is “not tired” but actual class is “tired”

	Positive (“tired”)	Negative (“not tired”)
True	True positive (**TP**)	True negative (**TN**)
False	False positive (**FP**)	False negative (**FN**)

Table 8.1: Table of the confusion matrix

Since we are interested to study the system's performance, so it is important to calculate the accuracy and precision of the system using the following equations:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{8.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8.2}$$

8.1.1 Performance Analysis for Eye-status based Tiredness Detection

Confusion matrix for sleepiness detection (Subject 1)						
Session Nr.	**TP**	**FP**	**TN**	**FN**	**Accuracy**	**Precision**
1.	5	1	22	1	93.10%	83.33%
2.	7	2	37	2	91.67%	77.78%
3.	17	6	79	4	90.57%	73.91%

Table 8.2: Performance analysis for eye-status based tiredness detection (Subject 1)

Confusion matrix for sleepiness detection (Subject 2)						
Session Nr.	**TP**	**FP**	**TN**	**FN**	**Accuracy**	**Precision**
1.	3	1	26	1	93.55%	75%
2.	9	4	59	3	90.67%	69.23%
3.	15	7	97	5	90.32%	68.18%

Table 8.3: Performance analysis for eye-status based tiredness detection (Subject 2)

8.1.2 Performance Analysis for Mouth-status based Tiredness Detection

Confusion matrix for yawn detection (Subject 1)						
Session Nr.	**TP**	**FP**	**TN**	**FN**	**Accuracy**	**Precision**
1.	7	0	13	6	76.92%	100%
2.	11	0	17	13	68.29%	100%
3.	19	0	29	26	64.87%	100%

Table 8.4: Performance analysis for mouth-status based tiredness detection (Subject 1)

Confusion matrix for yawn detection (Subject 2)						
Session Nr.	**TP**	**FP**	**TN**	**FN**	**Accuracy**	**Precision**
1.	9	0	15	7	77.42%	100%
2.	13	0	19	15	68.09%	100%
3.	23	1	31	29	64.29%	95.83%

Table 8.5: Performance analysis for mouth-status based tiredness detection (Subject 2)

8.1.3 Summary

Some observations that can be made from the analysis in section 8.1.1 and 8.1.2:

1. Eye-status based tiredness detection works fine without fail in all of the sessions, but struggles with the problem of detecting good number of false positives; thus it reduces system's precision. Figure 8.1(a) illustrates an example of precision loss of the system. This might be happening due to the limitations of Haar-cascade classifier used to identify face region in a frame.

2. On the other hand, mouth-status based tiredness detection suffers to detect actual yawn properly in many cases. The reason might be that the user opens his/her mouth too wide while yawning - which disturbs the basic expected structure of a face and not identified by the facial landmark detector; thus it reduces system's accuracy significantly. Figure 8.1(b) illustrates an example of accuracy loss of the system.

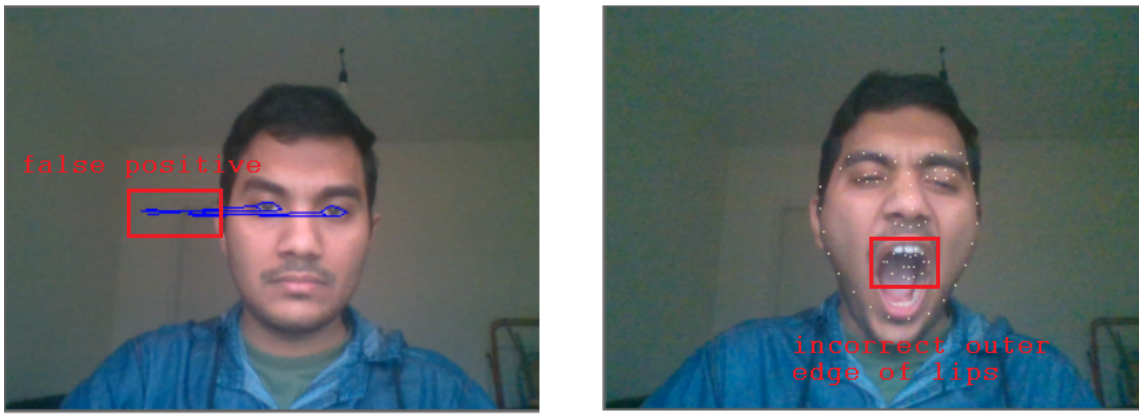


Figure 8.1: Observed performance Issues (a) precision loss, (b) accuracy loss

8.2 Comparative Study with Other Existing Methods

It is evident that still now there exists no vision-based employee tiredness assessment system. But several vision-based driver fatigue monitoring systems have been proposed in recent years. So a brief comparative study with these existing methods has been done in table 8.6 considering several important factors.

Factors	**Method 1**[5]	**Method 2**[7]	**Method 3**[8]	**Our method**
Hardware	Night-vision camera & LED	IR camera	Raspberry Pi camera	Any webcam
Library	OpenCV	OpenCV	OpenCV	OpenCV
Classifier	LBP & Haar cascade	LBP & Haar cascade	Haar cascade only	Haar cascade only
Detection approach	Open-eye detection	Eye-state classification	Eye-template matching	Landmark detection
Target ROI	Eye region only	Eye region only	Eye region only	Eye and mouth region
Calculation	Blink rate	Eye-state (open or closed)	Blink rate & duration of micro-sleep	Duration of micro-sleep & yawn
Computation	High	High	Medium	Medium
Speed	Average	Average	Good	Good
Performance	Average	Good	Good	Average
Cost	Average	Average	Average	Low

Table 8.6: Comparison with other existing methods

Based on the comparative study of table 8.6, it is clear that our proposed method offers a cost-effective setup and delivers more reliable result than other existing methods in real time. In section 8.3, the limitations of the system will be described. And finally, in section 8.4, we will discuss about possible future recommendations for our system.

8.3 Limitations of the System

Our proposed system was experimented in a bright room with constant light and initially we were using Logitech C170 webcam (specification is available in section B.1). So, the result was satisfactory in a stable good lighting condition. Although, when we tried to test in a low-light condition, it could not even identify individual's face. In order to solve this problem, we replaced our previous webcam with Logitech C920 PRO HD (specification is available in section B.2). Since this webcam has auto low-light correction feature, so it improved the result better than before, especially in low-light condition. But sometimes we used to notice a significant percentage of false-positive detection in extreme low-light condition. The possible solution would be to install a high-quality infrared webcam; however, in that case the system will no longer remain as a cost-effective system.

Since the system handles real-time video streaming and delivers result in run-time, so it requires to have sufficient amount of RAM to be installed before running the application. External GPU is not mandatory, but strongly suggested.

While the application is running in background, other applications are not allowed to access webcam. This can be a big drawback in such a condition when an employee needs to do frequent video-calling for business purpose.

Again, the prototype application expects the user's face to be always visible by the webcam, which means the head pose can have certain DOF. As a result, the system puts some constraints to individual, whereas in real-life a person can have many different posture while working in front of computer.

Finally, we expected to obtain result from the trained ML model (trained with keystroke dynamics dataset) for all users. But we understood that the trained ML model works with the individual only (for whom the dataset was prepared and the ML model was trained). So, we had to build individually-trained ML models for individual users respectively which would take significant amount of time to prepare the required big dataset. Since we did not have sufficient time to get the big dataset ready, so keystroke tiredness prediction module was not included in this current version of the application.

8.4 Future Recommendations

In spite of a few minor limitations, the proposed algorithms are proven working effectively in different lighting conditions of both laboratory and official environments. Still there exist plenty of opportunities to improve the system's applicability and performance. Since we have developed the prototype application using .NET framework (maintaining standard object-oriented design and architecture with proper documentation), so it is very much possible to develop a mobile application for smartphones (Android OS, iOS or Windows Phone) replicating the same concept with advanced features. As a result, drivers will find the application very useful. All they will have to do is to turn on the application in their mobile phone and place it in a position where facial components are fully visible. Most importantly, the system will be cost-effective too.

As we mentioned in the motivation, this research work intends to bring more intelligence to vision system, for example, computer control through individual's facial expression (e.g, eye blink, motion of pupil etc.) and possible motion(s). Thus handicapped people will be highly beneficiary. From technical perspective, a significant portion of existing software's source-code can be reused to implement required features.

In the context of regular computer users, we can utilize the trained ML model (having keystroke dynamics data) in order to analyze day-time fatigue with less margin of error. Even the existing software's source-code can be used to implement the necessary changes.

In addition, we can try to integrate the measurements of wearable sensors (e.g, Fitbit) in our existing application. Hence it will add more variables to the system (e.g, blood pressure, heart rate etc.) and the internal logic will become more complicated. But it will not only improve the result, but also increase the reliability of the system. In a word, it is possible to evaluate tiredness in real-time with more robust features by extending our existing system.

Chapter 9

Conclusion

9.1 Conclusion (English)

In this paper, a unique method for employee tiredness detection was proposed by monitoring eye status and frequency of yawning in real-time. Eye aspect ratio (EAR) and mouth aspect ratio (MAR) were used to monitor eye status and mouth status respectively in sequences of frames. It is obligatory to mention that OpenCV library functions have been used significantly (via EmguCV wrapper class) throughout the program and the prototype application was developed using C# programming language to ensure possibility of future improvement, maintenance and migration to other platforms (if necessary).

The concept of AdaBoost algorithm, Haar-like features, Integral image and CNN were studied extensively in order to detect face and necessary facial components. Once face was identified by the machine-learning based algorithm, it was set as ROI while trace corresponding landmark points. High quality webcam was used throughout the experiment, so that each individual frame can be utilized to detect employee tiredness which would trigger the alarm to alert him/her effectively. Other necessary biometric data (e.g, keystroke dynamics data) were stored in database to train a ML model for future utilization.

The proposed method was tested in lab environment as well as in official environment too. The system delivers satisfactory results in official environment with minimal computational complexity. The future work will focus on utilizing small variations not only from eye and mouth regions, but also from motion activities (e.g, employee head motion) with minimal computation overhead.

9.2 Conclusion (Estonian)

Käesolevas diplomitöös esitati unikaalne meetod mõõtmaks töötajate väsimustaset kasutades silmade seisundit ja haigutamise sagedust reaalajas. Silmade kuvasuhe (EAR) ja suude kuvasuhe (MAR) kasutati silmade ja suu seisund jälgimiseks vastavalt kaadritele. On vajalik ka ära mainida, et OpenCV teegikogu funktsioone on kasutatud suures osas (EmguCV wrapper klassi kaudu) läbi terve programmi ja rakenduse prototüüp oli arendatud kasutates C# programeerimiskeelt, et tagada lihtsam valmidus lisaarenduseks tulevikus, tehniliseks hoolduseks ja vajaduse korral migratsiooniks teistele platformidele.

AdaBoosti algortimi kontseptsiooni, Haari sarnaste omaduste, tervikliku kuvandi ja CNN uurimine olid vajalikud,et kinnistada teadmised näo erinevate aspektide jälgimiseks. Peale seda kui näo aspektid olid jälgitavad masinõppe algoritmi abil, kasutati ROI (Region of Interest, kitsendatud otsinguala) süsteemi erivevate punktide määramiseks. Kogu eksperimendi vältel kasutati kõrge kaadrisagedusega veebikaamerat, et iga üksikut kaadrit saaks võimalikult efektiivselt ära kasutada töötajate kurnatuse taseme jälgimiseks ning saaks seadistada alarmi vastavalt detekteeritud tasemele. Ülejäänud vajalikud biomeetrilised andmed salvestati andmebaasi, et treenida ML mudelit tulevaseks kasutuseks.

Välja pakutud meetodit testiti laborikeskonnas ja ka kontoris -päriselus. Süsteemi kasutamise tulemusena saavutati rahuldavad tulemused kontori keskkonnas vähese süsteemikeerukusega. Tulevased arendused keskenduvad mitte ainult variatsioonide kasutamisele suu ja silmade alal, vaid lahendusele lisanduvad ka väikseked liikumistegevuse nagu näiteks pea liigutamine ning säilib ka programmi lihtsus.

Bibliography

[1] V. Pille, V. Tuulik, and A. Hazak, "Sitting at a desk at work makes creative employees tired," *TTU Economic Research Series*, 2017.

[2] Y. S. Can, B. Arnrich, and C. Ersoy, "Stress detection in daily life scenarios using smart phones and wearable sensors: A survey," *Journal of biomedical informatics*, p. 103139, 2019.

[3] A. Pimenta, D. Carneiro, J. Neves, and P. Novais, "Analysis of mental fatigue and mood states in workplaces," in *Intelligent Distributed Computing IX*. Springer, 2016, pp. 111–119.

[4] M. A. Haque, R. Irani, K. Nasrollahi, and T. B. Moeslund, "Facial video-based detection of physical fatigue for maximal muscle activity," *IET Computer Vision*, vol. 10, no. 4, pp. 323–330, 2016.

[5] G. L. R. Clavijo, J. O. Patino, and D. M. Leon, "Detection of visual fatigue by analyzing the blink rate," in *2015 20th Symposium on Signal Processing, Images and Computer Vision (STSIVA)*. IEEE, 2015, pp. 1–5.

[6] E. Abdulin and O. Komogortsev, "User eye fatigue detection via eye movement behavior," in *Proceedings of the 33rd annual ACM conference extended abstracts on human factors in computing systems*. ACM, 2015, pp. 1265–1270.

[7] D. Ghimire, S. Jeong, S. Yoon, S. Park, and J. Choi, "Real-time sleepiness detection for driver state monitoring system," *Adv. Sci. Technol. Lett*, vol. 120, pp. 1–8, 2015.

[8] O. Khunpisuth, T. Chotchinasri, V. Koschakosai, and N. Hnoohom, "Driver drowsiness detection using eye-closeness detection," in *2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. IEEE, 2016, pp. 661–668.

[9] Y. Zhang and C. Hua, "Driver fatigue recognition based on facial expression analysis using local binary patterns," *Optik*, vol. 126, no. 23, pp. 4501–4505, 2015.

[10] J. Kumari, R. Rajesh, and K. M. Pooja, "Facial expression recognition: A survey," *Procedia Computer Science*, vol. 58, pp. 486–491, 2015.

[11] T. M. Abhishree, J. Latha, K. Manikantan, and S. Ramachandran, "Face recognition using gabor filter based feature extraction with anisotropic diffusion as a pre-processing technique," *Procedia Computer Science*, vol. 45, pp. 312–321, 2015.

[12] L. Liu, P. Fieguth, G. Zhao, M. Pietikäinen, and D. Hu, "Extended local binary patterns for face recognition," *Information Sciences*, vol. 358, pp. 56–72, 2016.

[13] S. Agrawal and P. Khatri, "Facial expression detection techniques: based on viola and jones algorithm and principal component analysis," in *2015 Fifth International Conference on Advanced Computing & Communication Technologies*. IEEE, 2015, pp. 108–112.

[14] A. Painsky, S. Rosset, and M. Feder, "Linear independent component analysis over finite fields: Algorithms and bounds," *IEEE Transactions on Signal Processing*, vol. 66, no. 22, pp. 5875–5886, 2018.

[15] A. Tharwat, T. Gaber, A. Ibrahim, and A. E. Hassanien, "Linear discriminant analysis: A detailed tutorial," *AI communications*, vol. 30, no. 2, pp. 169–190, 2017.

[16] M. Ulinskas, R. Damaševičius, R. Maskeliūnas, and M. Woźniak, "Recognition of human daytime fatigue using keystroke data," *Procedia computer science*, vol. 130, pp. 947–952, 2018.

[17] E. V. E. Centers, "Lazy eye," https://epicvisioneyecenters.com/lazy-eye, accessed: 2019-10-12.

[18] B. Systems, "Facial emg & startle response," https://www.biopac.com/, accessed: 2019-07-24.

[19] M. Pro, "How to digitize human emotions for virtual reality applications," https://maker.pro/, accessed: 2019-10-03.

[20] A. Dasgupta, A. George, S. Happy, and A. Routray, "A vision-based system for monitoring the loss of attention in automotive drivers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 4, pp. 1825–1838, 2013.

[21] R. Shah, "Image sensors explained: How CCD and CMOS sensors works? CCD vs CMOS," 2017. [Online]. Available: https://www.youtube.com/watch?v=FKJFIzDfUNE

[22] D. Bode, "Fundamentals of photography: What is a digital image?" 2013. [Online]. Available: https://www.youtube.com/watch?v=TVTn7mYKegY

[23] GeeksforGeeks, "Digital image processing basics," https://www.geeksforgeeks.org/, accessed: 2019-08-14.

[24] P. Chakravorty, "What is a signal? [lecture notes]," *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 175–177, 2018.

[25] TutorialPoint, "Digital image processing," http://www.tutorialspoint.com/dip/, accessed: 2019-08-14.

[26] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *CVPR (1)*, vol. 1, no. 511-518, p. 3, 2001.

[27] J.Brownlee, "A gentle introduction to computer vision," https://machinelearningmastery.com/, accessed: 2019-03-19.

[28] J. Starmer, "Adaboost, clearly explained," 2019. [Online]. Available: https://www.youtube.com/watch?v=LsK-xG1cLYA

[29] E. Emer, "Boosting, adaboost algorithm," http://math.mit.edu/, accessed: 2019-08-14.

[30] OpenCV, "Face detection using haar cascades," https://docs.opencv.org/, accessed: 2019-08-27.

[31] A. Harvey, "Opencv face detection: Visualized," 2010. [Online]. Available: https://vimeo.com/12774628

[32] Badgerati, "Computer vision – the integral image," https://computersciencesource.wordpress.com/, accessed: 2019-08-28.

[33] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. international conference on image processing*, vol. 1. IEEE, 2002, pp. I–I.

[34] S. Hameed, "Haarcascade for eyes," http://umich.edu/~shameem/, accessed: 2019-09-26.

[35] M. Castrillón-Santana, O. Déniz-Suárez, M. Hernández-Tejera, and C. Guerra-Artal, "Face and facial feature detection evaluation," in *Third International Conference on Computer Vision Theory and Applications, VISAPP08*, January 2008.

[36] H. S. Chatterjee, "A basic introduction to convolutional neural network," https://medium.com/@himadrisankarchatterjee/, accessed: 2019-07-16.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[38] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[39] J.Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[40] A. Poddar, M. Sahidullah, and G. Saha, "Speaker verification with short utterances: a review of challenges, trends and opportunities," *IET Biometrics*, vol. 7, no. 2, pp. 91–101, 2018.

[41] A. Harilal, F. Toffalini, I. Homoliak, J. Castellanos, J. Guarnizo, S. Mondal, and M. Ochoa, "The wolf of sutd (twos): A dataset of malicious insider threat behavior based on a gamified competition." *JoWUA*, vol. 9, no. 1, pp. 54–85, 2018.

[42] A. A. E. Ahmed and I. Traore, "Mouse dynamics biometric technology," in *Behavioral Biometrics for Human Identification: Intelligent Applications*. IGI Global, 2010, pp. 207–223.

[43] M. Antal and E. Egyed-Zsigmond, "Intrusion detection using mouse dynamics," *IET Biometrics*, 2019.

[44] M. R. Rosekind, K. B. Gregory, M. M. Mallis, S. L. Brandt, B. Seal, and D. Lerner, "The cost of poor sleep: workplace productivity loss and associated costs," *Journal of Occupational and Environmental Medicine*, vol. 52, no. 1, pp. 91–98, 2010.

[45] G. Mamaladze, "Processing global mouse and keyboard hooks in c#," https://www.codeproject.com/Articles/7294/Processing-Global-Mouse-and-Keyboard-Hooks-in-C, accessed: 2019-10-13.

[46] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1867–1874.

[47] G. Tzimiropoulos and M. Pantic, "Optimization problems for fast aam fitting in-the-wild," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 593–600.

[48] S. Ren, X. Cao, Y. Wei, and J. Sun, "Face alignment at 3000 fps via regressing local binary features," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1685–1692.

[49] L. Kurnianggoro, "Facemark api for opencv," https://github.com/kurnianggoro/GSOC2017, accessed: 2019-10-13.

[50] A. Singh, C. Chandewar, and P. Pattarkine, "Driver drowsiness alert system with effective feature extraction," *International Journal for Research in Emerging Science and Technology*, vol. 5, no. 4, pp. 26–31, 2018.

[51] T. Soukupova and J. Cech, "Eye blink detection using facial landmarks," in *21st Computer Vision Winter Workshop, Rimske Toplice, Slovenia*, 2016.

[52] H. R. Schiffman, *Sensation and perception: An integrated approach*. John Wiley & Sons, 1990.

[53] P. Awasekar, M. Ravi, S. Doke, and Z. Shaikh, "Driver fatigue detection and alert system using non-intrusive eye and yawn detection," *International Journal of Computer Applications*, vol. 975, p. 8887.

[54] S. Gupta and S. Mittal, "Yawning and its physiological significance," *International Journal of Applied and Basic Medical Research*, vol. 3, no. 1, p. 11, 2013.

[55] S. Abtahi, B. Hariri, and S. Shirmohammadi, "Driver drowsiness monitoring based on yawning detection," in *2011 IEEE International Instrumentation and Measurement Technology Conference*. IEEE, 2011, pp. 1–4.

[56] Microsoft, "What is ml.net and how does it work?" https://docs.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work#machine-learning-model, accessed: 2019-11-07.

[57] C. Reis, C. Mestre, H. Canhão, D. Gradwell, and T. Paiva, "Sleep complaints and fatigue of airline pilots," *Sleep Science*, vol. 9, no. 2, pp. 73–77, 2016.

[58] L. Reinke, Y. Özbay, W. Dieperink, and J. E. Tulleken, "The effect of chronotype on sleepiness, fatigue, and psychomotor vigilance of icu nurses during the night shift," *Intensive care medicine*, vol. 41, no. 4, pp. 657–666, 2015.

[59] C. Nikulin, G. Lopez, E. Piñonez, L. Gonzalez, and P. Zapata, "Nasa-tlx for predictability and measurability of instructional design models: case study in design methods," *Educational Technology Research and Development*, vol. 67, no. 2, pp. 467–493, 2019.

[60] C. M. Davis, P. G. Roma, and R. D. Hienz, "A rodent model of the human psychomotor vigilance test: Performance comparisons," *Journal of neuroscience methods*, vol. 259, pp. 57–71, 2016.

[61] A. Nanduri and L. Sherry, "Anomaly detection in aircraft data using recurrent neural networks (rnn)," in *2016 Integrated Communications Navigation and Surveillance (ICNS)*. IEEE, 2016, pp. 5C2–1.

[62] E. Corporation, "Cascadeclassifier.detectmultiscale method," http://www.emgu.com, accessed: 2019-09-01.

List of Figures

List of Tables

Appendix A

Concepts and Theories

A.1 Theory of Tiredness

A.1.1 Subjective Assessment of Tiredness

Tool	Acronym	Description
Fatigue Severity Scale	FSS	It is a self-response questionnaire consists of 9 items, referring to the previous week, rated on a 7-point Likert scale, ranging between "1: strongly disagree" and "7: strongly agree". The total score is calculated by adding all items and then dividing the sum by 9. The scale assesses the level of perceived fatigue in daily situations. Results at or above 4 indicate a clinically significant level of fatigue [57]. This fatigue scale is inspired from Samn Perelli's 7 point checklist.
Karolinska Sleepiness Scale	KSS	It is a subjective scale used to measure sleepiness on a scale ranging from 1 to 9, with "1: very alert", "3: alert", "5: neither alert nor sleepy", "7: sleepy, but no effort to keep awake" and "9: very sleepy great effort to keep awake" [58].
NASA Task Load Index	NASA TLX	It is a multi-dimensional rating procedure that assigns a total workload score based on a weighted average of six sub-scales: mental demand (mental and perceptive activity); physical demand (degree of physical effort); temporal demand (temporal perception); performance (degree of goal accomplishment); effort (amount of physical and mental effort); and frustration level (feeling of pressure, discouragement, and insecurity during execution) [59].

Table A.1: Description of subjective assessment of tiredness

A.1.2 Objective Assessment of Tiredness

Tool	Acronym	Description
Psychomotor Vigilance Task	PVT	It is a computer-based risk assessment tool to quantify fatigue and sustained attention in laboratory, clinical, and operational settings – where individuals are instructed to respond to a digital signal by pressing a key. Then errors of omission and commission are recorded [60].
Flight Operational Quality Assurance	FOQA	It collects time-series data from flight that identifies events and trends that helps to reduce safety margins [61].

Table A.2: Description of objective assessment of tiredness

A.2 Theory of Computer Vision

A.2.1 What is Computer Vision?

Computer vision (CV) is a field of study that concerns with development of techniques that help computers in order to extract information from digital images (e.g, photographs, videos) and solve practical problems. It is a multidisciplinary field that can be defined as a subfield of artificial intelligence and machine learning that involves usage of specialized methods and general learning algorithms [27].

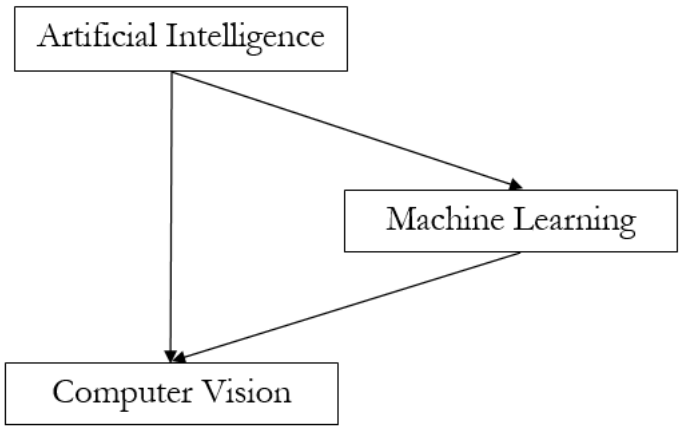


Figure A.1: Overview of the relationship of AI and CV [27]

In our study, we use computer vision in order to detect faces, particularly tired faces.

A.2.2 Theory of Image Sensor

An image sensor converts light energy (focused beam of photons) into an electrical output modulated by graphic information. It is used in electronic imaging devices including digital cameras, medical imaging equipment, thermal imaging devices, radar, sonar etc.

Currently two types of image sensors are used.

- CCD (Charged Coupled Device)
- CMOS (Complementary Metal Oxide Semiconductor)

Both CCD and CMOS image sensor consist of millions of photosites (pixels). These photosites convert the incoming light into charge. In terms of mechanism, some common steps are done by both type of image sensors. But depending on the type of sensors, these steps might vary in terms of sequence [21].

- Light to Charge Conversion: Photosites (pixels) are exposed to the light for a certain amount of time.
- Charge Accumulation: Charge will be collected in the photosites (pixels).
- Transfer: These pixels are transferred for the further processing.
- Charge to Voltage Conversion: After transfer, this charge is converted into voltage.
- Amplification: This voltage is amplified by the amplifiers.

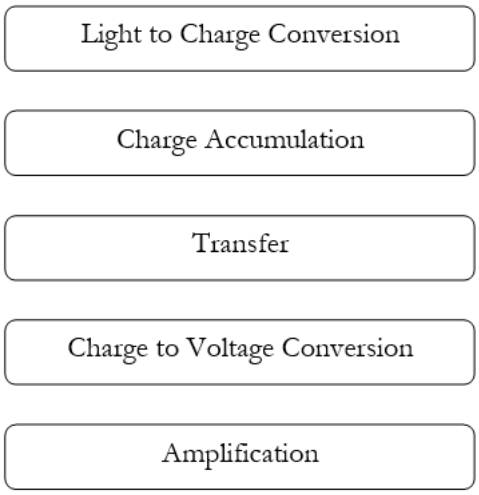


Figure A.2: Common steps of image sensors [21]

A.2.3 Comparison between CCD and CMOS Sensor

- **System Integration**: CCD is very old technology. In this technology, it is not possible to integrate the peripheral components like Timers, ADC to the main sensor. So for integrating the peripheral components, you need additional chip. So the overall size of CCD sensor will get large. In case of CMOS sensors, since the fabrication procedure is same as the fabrication procedure of integrated circuits, so it is possible to integrate these peripheral components (Timers, ADC) into the single chip. So in case of CMOS sensor, it is possible to have a camera-on-chip or system-on-chip. Because of that, the CMOS sensor is quite compact [21].
- **Power Consumption**: In case of CCD sensors, we require different power supplies for different timing clocks. Typical voltage required is 7V to 10V. So overall power consumption in CCD sensors will be high. In case of CMOS sensors, we require single power supply. Typical voltage required is 3.3V to 5V, which is relatively low. So overall power consumption in CMOS sensors will be lesser compared to CCD sensors. So the applications where power consumption is the main criteria, CMOS sensors are preferred over CCD sensors [21].
- **Processing Speed**: In case of CCD sensors, the charge that is generated in each pixel is converted into the voltage one-by-one. So overall processing speed of CCD sensors is lesser compared to CMOS sensors. Now the processing speed can be improved by using

multiple vertical shift registers. In that case, we will require additional hardware. In case of CMOS sensors, the charge to voltage conversion takes place in the same pixel. So the processing speed is higher compared to the CCD sensors. Now the processing speed can be improved by using multiple column select lines. So in this way, by doing parallel processing we can increase the processing speed of this CMOS sensor [21].

- **Noise and Sensitivity**: In the case of CMOS sensors, charge-to-voltage converter circuit as well as the amplification circuit is integrated in the same pixel. So overall fill-factor of the CMOS sensor is less compared to the CCD sensor. And because of that, the sensitivity of the CMOS sensor will be less compared to the CCD sensor. And because of that, the Dynamic range of CCD sensor is quite high compared to the CMOS sensor. Not only that, in case of the CMOS sensor, the amplifier which has been used in each pixel is not identical. So because of that, you will see the non-uniform amplification. This will act as an additional noise. But nowadays using different techniques like microlens on each pixel of CMOS sensor, the sensitivity can be increased [21].
- **Image distortion**: In case of CCD sensor, if you expose this sensor for longer time, then you will notice the effect which is called "blooming". Now-a-days, using the anti-blooming technology, we can reduce this blooming. While in case of CMOS sensor, the most common type of distortion is known to be "rolling shutter". Since the pixels are read in line-by-line fashion, so whenever any fast-moving-object is captured by the CMOS sensor, then this rolling-shutter effect is quite noticeable. For example, if you try to capture the wing of helicopter, you will notice it as curvature. In case of CCD sensors, all pixels are getting exposed in the same time, so we do not notice this effect in CCD sensors. So, to remove this rolling-shutter effect in CMOS sensor, all the pixels need to be exposed in the same time, which is known as the "global-shutter". Now-a-days, many of the CMOS sensors are also coming with this global shutter [21].

A.2.4 Types of an image

Digital image can be categorized into following types [23].

- **Binary Image**: It contains only two pixel elements i.e 0 & 1, where 0 refers to black and 1 refers to white. This image is also known as Monochrome.
- **Black and White Image**: It consists of only black and white color is called Black and White Image.
- **8 bit Color Format**: It has 256 different shades of colors in it and commonly known as Grayscale Image. In this format, 0 stands for Black, and 255 stands for white, and 127 stands for gray.
- **16 bit Color Format**: It is a color image format. It has 65,536 different colors in it. It is also known as High Color Format. In this format the distribution of color is not as same as Grayscale image.

A.2.5 Preparation of Custom Haar Cascade

Step 1 - NEGATIVE IMAGES

We need to make a lot of negative images (let's say 200 images) in .jpg format. Please note that, the images must be grayscale images.

Step 2 - NEGATIVE INFO FILE

After putting all the negative images, now we need to prepare the list of negative images. By clicking `create_list.bat`, we create the list of negative images.

Step 3 - POSITIVE IMAGES

We need to make some positive images (let's say 20 images) in .bmp format and put them inside `positive\rawdata` directory.

Step 4 - POSITIVE INFO FILE

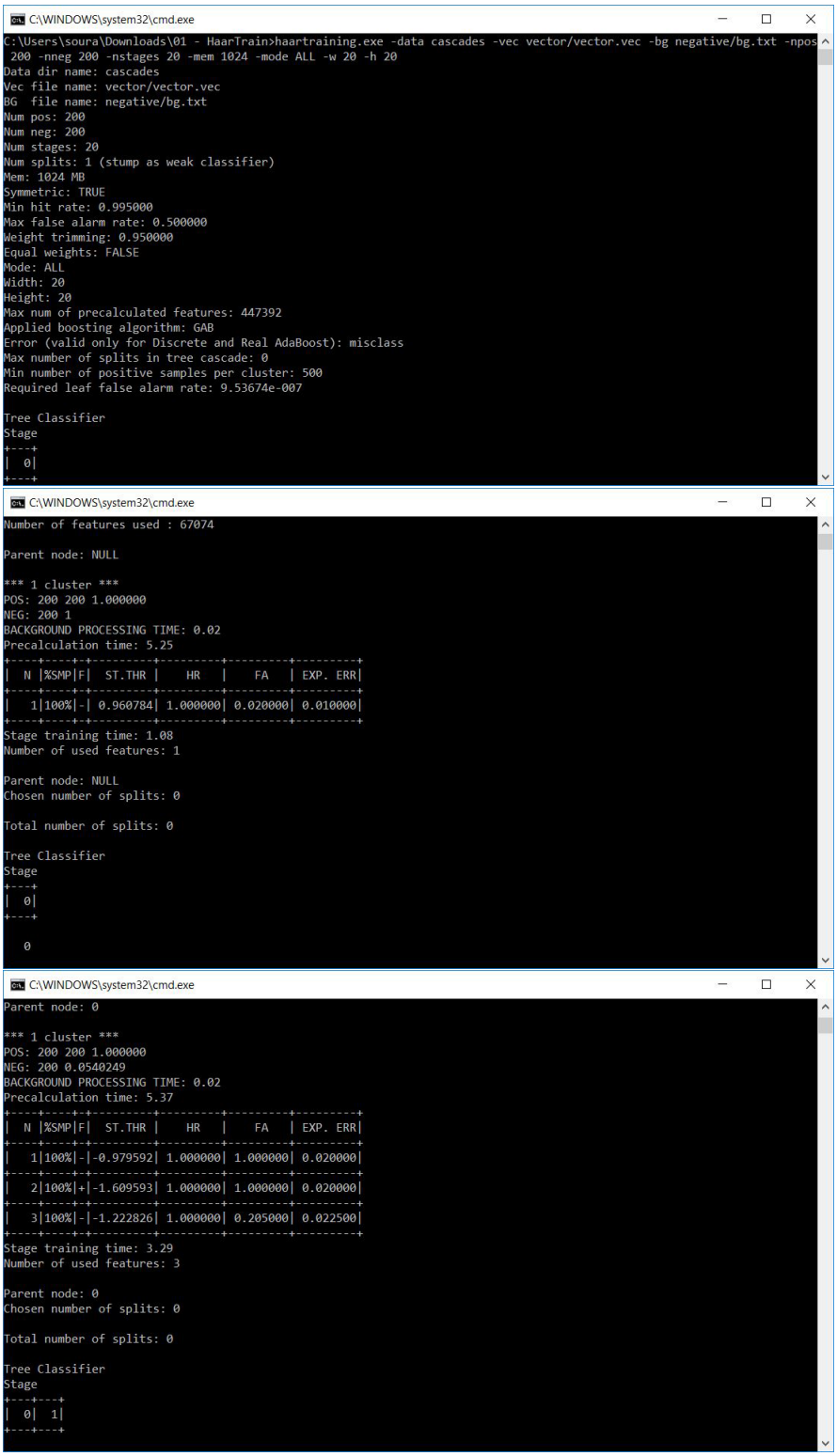
We need to open the `objectmarker.exe` and crop the positive images. After every crop activity, we need to press the spacebar to confirm the crop. Then we need to press Enter key to go to the next image.

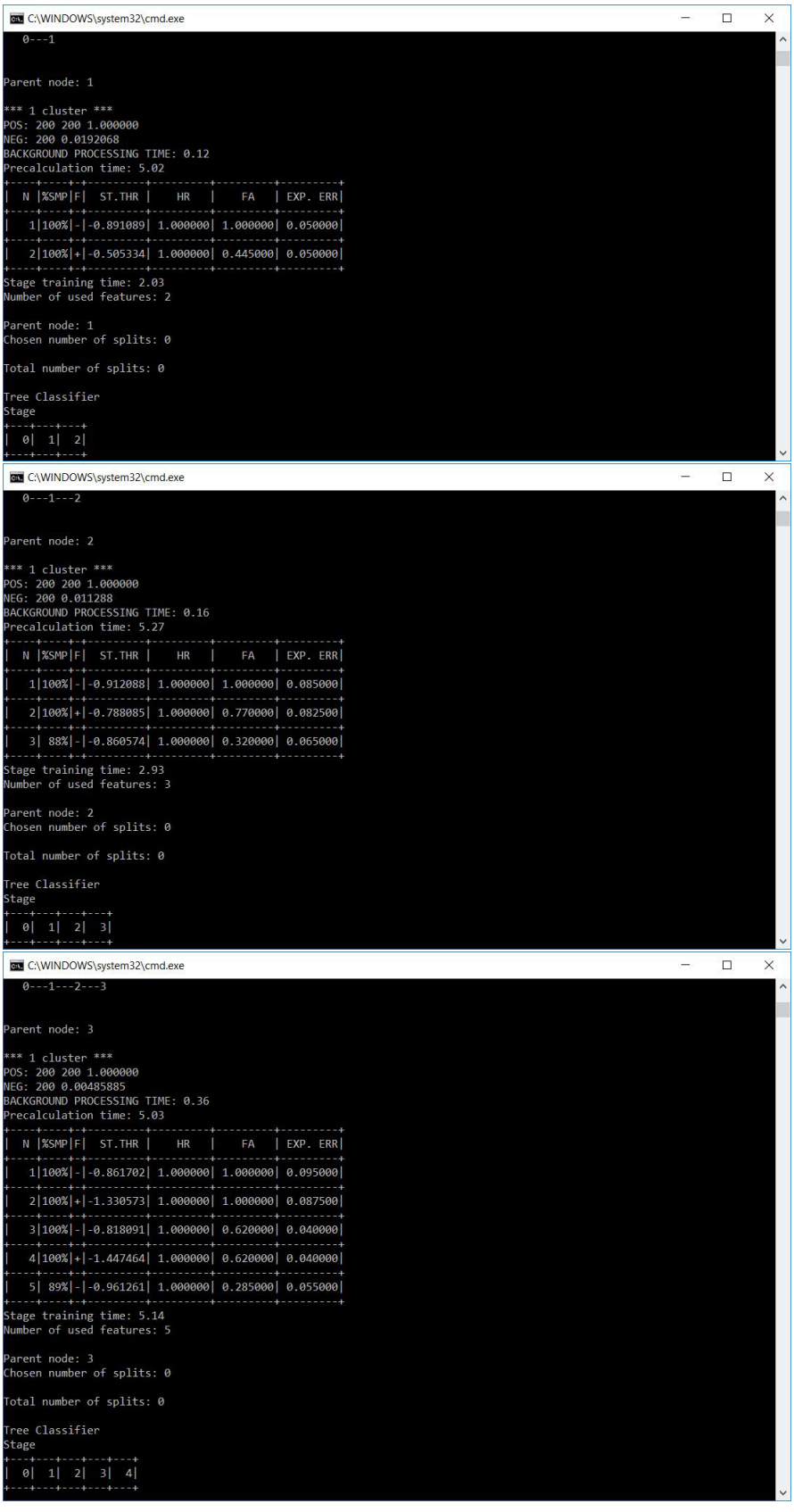
Step 5 - CREATING SAMPLES

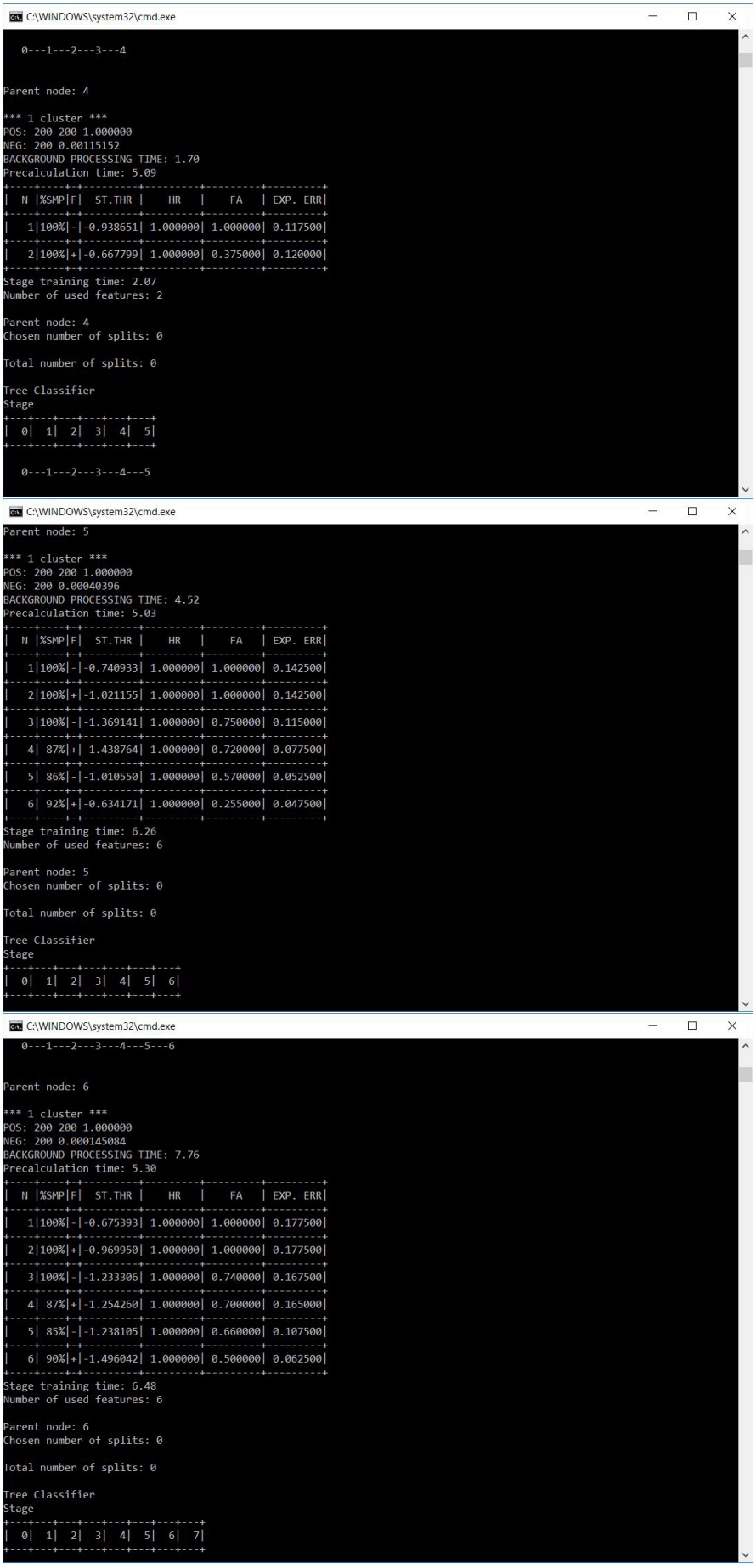
We need to create samples by clicking `samples_creation.bat`. This will generate a positive vector file (.vec) in `vectors` directory.

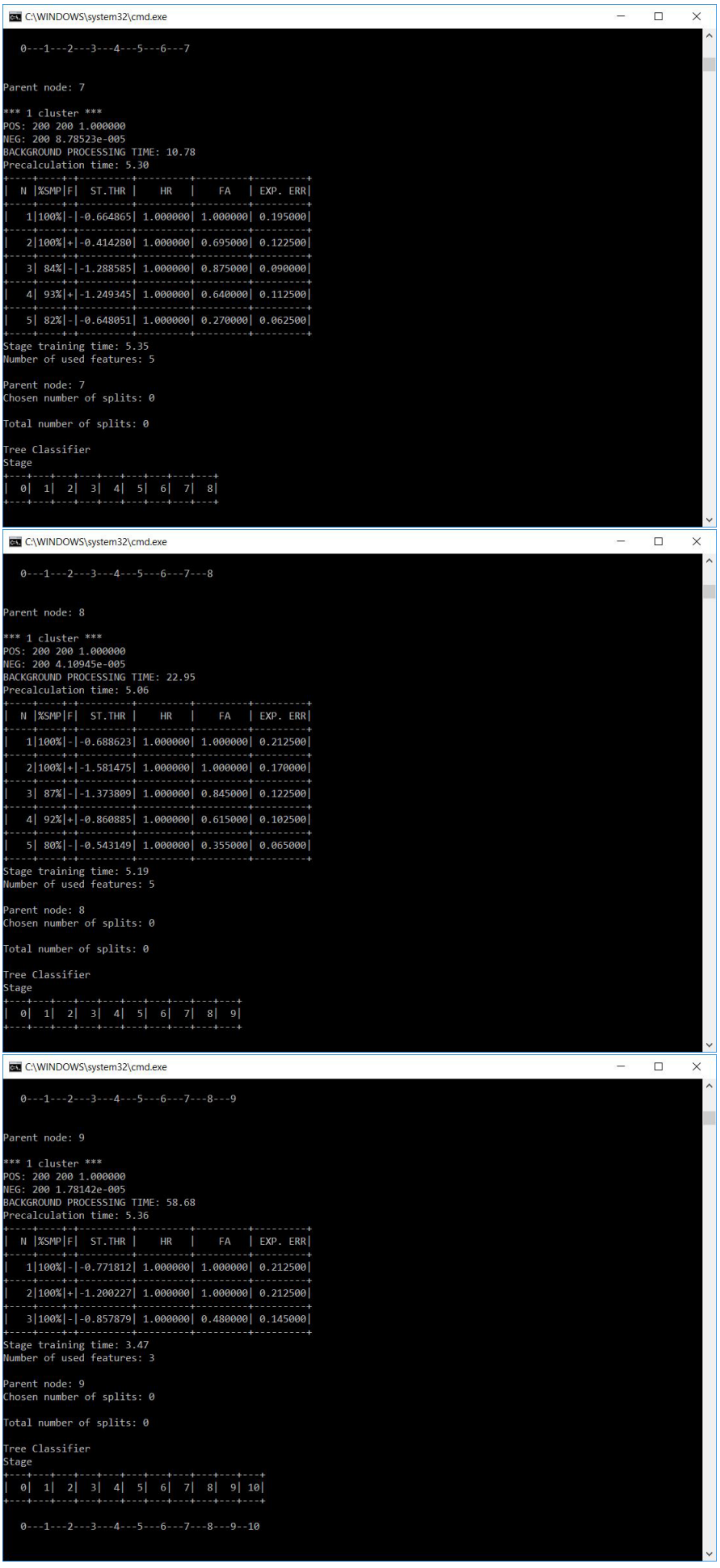
Step 6 - HAAR TRAINING

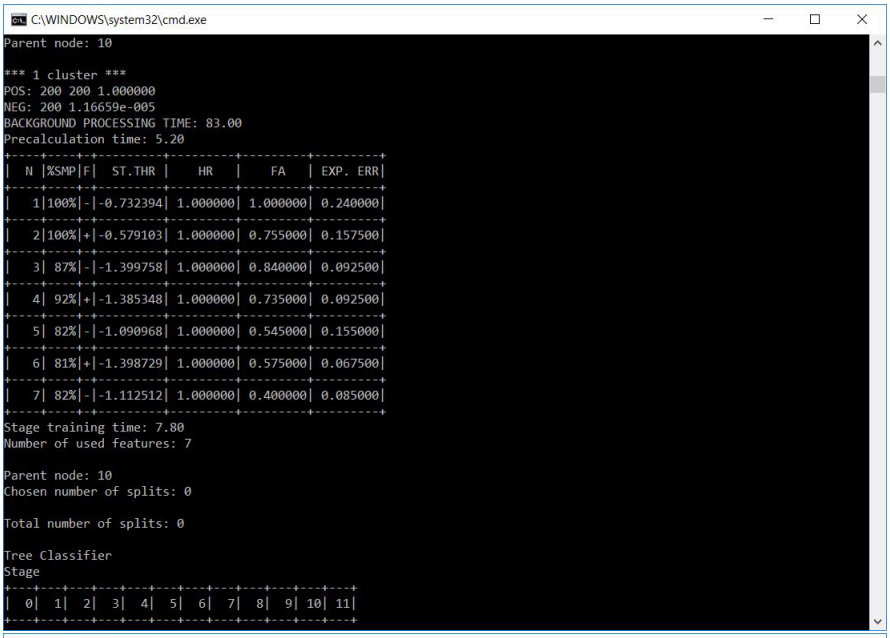
We need to open `cascades` directory and delete all the existing folders. Then we need to edit the `02 haarTraining.bat` file using Notepad++. We need to change the numbers of positive and negative images that are passed as parameter. Then we need to save it and run it. After some time, the Haar Cascade training will be completed. Please note that, the training time might vary in between several minutes to few hours, depending on the training parameters. The following figures illustrate the training process in a step-by-step approach.

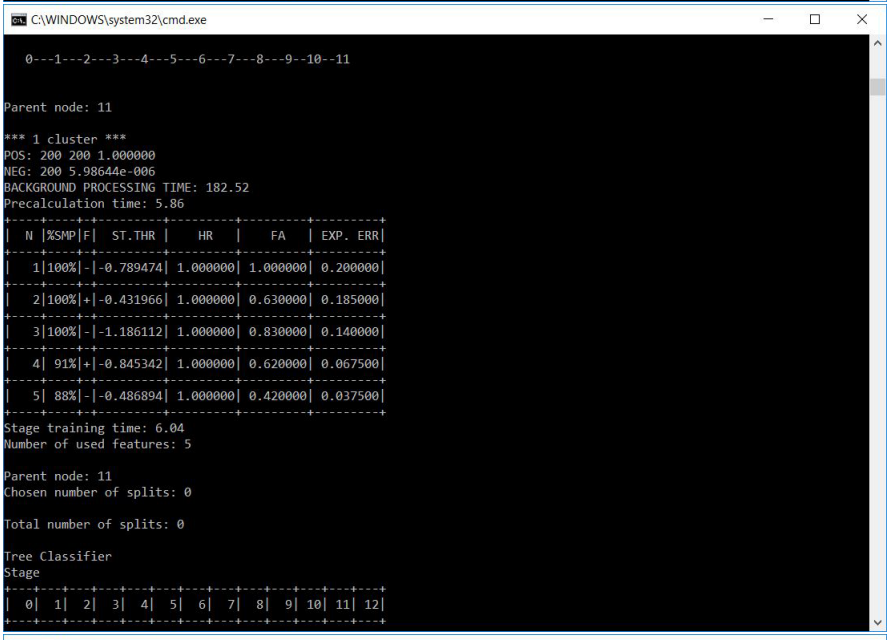

C:\WINDOWS\system32\cmd.exe
C:\Users\soura\Downloads\01 - HaarTrain>haartraining.exe -data cascades -vec vector/vector.vec -bg negative/bg.txt -npos 200 -nneg 200 -nstages 20 -mem 1024 -mode ALL -w 20 -h 20
Data dir name: cascades
Vec file name: vector/vector.vec
BG file name: negative/bg.txt
Num pos: 200
Num neg: 200
Num stages: 20
Num splits: 1 (stump as weak classifier)
Mem: 1024 MB
Symmetric: TRUE
Min hit rate: 0.995000
Max false alarm rate: 0.500000
Weight trimming: 0.950000
Equal weights: FALSE
Mode: ALL
Width: 20
Height: 20
Max num of precalculated features: 447392
Applied boosting algorithm: GAB
Error (valid only for Discrete and Real AdaBoost): misclass
Max number of splits in tree cascade: 0
Min number of positive samples per cluster: 500
Required leaf false alarm rate: 9.53674e-007
Tree Classifier
Stage
+---+
| 0|
+---+
C:\WINDOWS\system32\cmd.exe
Number of features used : 67074
Parent node: NULL
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 1
BACKGROUND PROCESSING TIME: 0.02
Precalculation time: 5.25
+----+----+-+---------+---------+---------+---------+
| N |%SMP|F| ST.THR | HR | FA | EXP. ERR|
+----+----+-+---------+---------+---------+---------+
| 1|100%|-| 0.960784| 1.000000| 0.020000| 0.010000|
+----+----+-+---------+---------+---------+---------+
Stage training time: 1.08
Number of used features: 1
Parent node: NULL
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
+---+
| 0|
+---+
0
C:\WINDOWS\system32\cmd.exe
Parent node: 0
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.0540249
BACKGROUND PROCESSING TIME: 0.02
Precalculation time: 5.37
+----+----+-+---------+---------+---------+---------+
| N |%SMP|F| ST.THR | HR | FA | EXP. ERR|
+----+----+-+---------+---------+---------+---------+
| 1|100%|-|-0.979592| 1.000000| 1.000000| 0.020000|
+----+----+-+---------+---------+---------+---------+
| 2|100%|+|-1.609593| 1.000000| 1.000000| 0.020000|
+----+----+-+---------+---------+---------+---------+
| 3|100%|-|-1.222826| 1.000000| 0.205000| 0.022500|
+----+----+-+---------+---------+---------+---------+
Stage training time: 3.29
Number of used features: 3
Parent node: 0
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
+---+---+
| 0| 1|
+---+---+

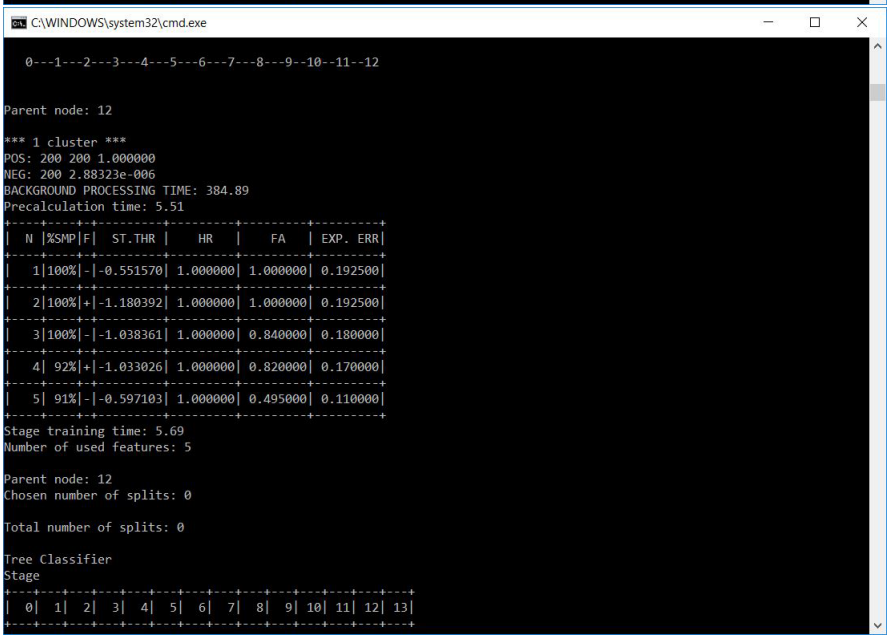

C:\WINDOWS\system32\cmd.exe
0---1
Parent node: 1
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.0192068
BACKGROUND PROCESSING TIME: 0.12
Precalculation time: 5.02
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.891089	1.000000	1.000000	0.050000
2	100%	+	-0.505334	1.000000	0.445000	0.050000
Stage training time: 2.03
Number of used features: 2
Parent node: 1
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2
C:\WINDOWS\system32\cmd.exe
0---1---2
Parent node: 2
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.011288
BACKGROUND PROCESSING TIME: 0.16
Precalculation time: 5.27
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.912088	1.000000	1.000000	0.085000
2	100%	+	-0.788085	1.000000	0.770000	0.082500
3	88%	-	-0.860574	1.000000	0.320000	0.065000
Stage training time: 2.93
Number of used features: 3
Parent node: 2
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3
C:\WINDOWS\system32\cmd.exe
0---1---2---3
Parent node: 3
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.00485885
BACKGROUND PROCESSING TIME: 0.36
Precalculation time: 5.03
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.861702	1.000000	1.000000	0.095000
2	100%	+	-1.330573	1.000000	1.000000	0.087500
3	100%	-	-0.818091	1.000000	0.620000	0.040000
4	100%	+	-1.447464	1.000000	0.620000	0.040000
5	89%	-	-0.961261	1.000000	0.285000	0.055000
Stage training time: 5.14
Number of used features: 5
Parent node: 3
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4

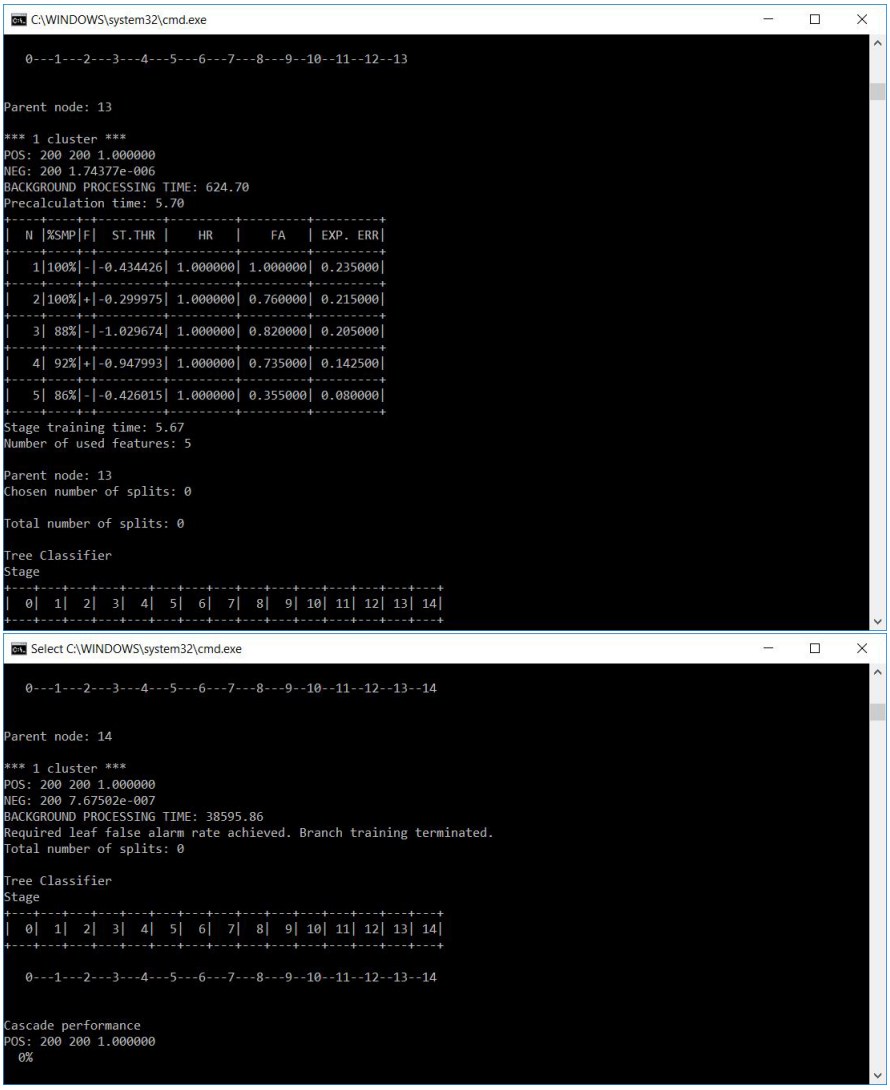

C:\WINDOWS\system32\cmd.exe
0---1---2---3---4
Parent node: 4
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.00115152
BACKGROUND PROCESSING TIME: 1.70
Precalculation time: 5.09
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.938651	1.000000	1.000000	0.117500
2	100%	+	-0.667799	1.000000	0.375000	0.120000
Stage training time: 2.07
Number of used features: 2
Parent node: 4
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5
0---1---2---3---4---5
C:\WINDOWS\system32\cmd.exe
Parent node: 5
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.00040396
BACKGROUND PROCESSING TIME: 4.52
Precalculation time: 5.03
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.740933	1.000000	1.000000	0.142500
2	100%	+	-1.021155	1.000000	1.000000	0.142500
3	100%	-	-1.369141	1.000000	0.750000	0.115000
4	87%	+	-1.438764	1.000000	0.720000	0.077500
5	86%	-	-1.010550	1.000000	0.570000	0.052500
6	92%	+	-0.634171	1.000000	0.255000	0.047500
Stage training time: 6.26
Number of used features: 6
Parent node: 5
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6
C:\WINDOWS\system32\cmd.exe
0---1---2---3---4---5---6
Parent node: 6
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 0.000145084
BACKGROUND PROCESSING TIME: 7.76
Precalculation time: 5.30
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.675393	1.000000	1.000000	0.177500
2	100%	+	-0.969950	1.000000	1.000000	0.177500
3	100%	-	-1.233306	1.000000	0.740000	0.167500
4	87%	+	-1.254260	1.000000	0.700000	0.165000
5	85%	-	-1.238105	1.000000	0.660000	0.107500
6	90%	+	-1.496042	1.000000	0.500000	0.062500
Stage training time: 6.48
Number of used features: 6
Parent node: 6
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7


C:\WINDOWS\system32\cmd.exe
0---1---2---3---4---5---6---7
Parent node: 7
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 8.78523e-005
BACKGROUND PROCESSING TIME: 10.78
Precalculation time: 5.30
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.664865	1.000000	1.000000	0.195000
2	100%	+	-0.414280	1.000000	0.695000	0.122500
3	84%	-	-1.288585	1.000000	0.875000	0.090000
4	93%	+	-1.249345	1.000000	0.640000	0.112500
5	82%	-	-0.648051	1.000000	0.270000	0.062500
Stage training time: 5.35
Number of used features: 5
Parent node: 7
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7	8
C:\WINDOWS\system32\cmd.exe
0---1---2---3---4---5---6---7---8
Parent node: 8
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 4.10945e-005
BACKGROUND PROCESSING TIME: 22.95
Precalculation time: 5.06
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.688623	1.000000	1.000000	0.212500
2	100%	+	-1.581475	1.000000	1.000000	0.170000
3	87%	-	-1.373809	1.000000	0.845000	0.122500
4	92%	+	-0.860885	1.000000	0.615000	0.102500
5	80%	-	-0.543149	1.000000	0.355000	0.065000
Stage training time: 5.19
Number of used features: 5
Parent node: 8
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7	8	9
C:\WINDOWS\system32\cmd.exe
0---1---2---3---4---5---6---7---8---9
Parent node: 9
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 1.78142e-005
BACKGROUND PROCESSING TIME: 58.68
Precalculation time: 5.36
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.771812	1.000000	1.000000	0.212500
2	100%	+	-1.200227	1.000000	1.000000	0.212500
3	100%	-	-0.857879	1.000000	0.480000	0.145000
Stage training time: 3.47
Number of used features: 3
Parent node: 9
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7	8	9	10
0---1---2---3---4---5---6---7---8---9--10


C:\WINDOWS\system32\cmd.exe
Parent node: 10
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 1.16659e-005
BACKGROUND PROCESSING TIME: 83.00
Precalculation time: 5.20
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.732394	1.000000	1.000000	0.240000
2	100%	+	-0.579103	1.000000	0.755000	0.157500
3	87%	-	-1.399758	1.000000	0.840000	0.092500
4	92%	+	-1.385348	1.000000	0.735000	0.092500
5	82%	-	-1.090968	1.000000	0.545000	0.155000
6	81%	+	-1.398729	1.000000	0.575000	0.067500
7	82%	-	-1.112512	1.000000	0.400000	0.085000
Stage training time: 7.80
Number of used features: 7
Parent node: 10
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7	8	9	10	11


C:\WINDOWS\system32\cmd.exe
0---1---2---3---4---5---6---7---8---9--10--11
Parent node: 11
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 5.98644e-006
BACKGROUND PROCESSING TIME: 182.52
Precalculation time: 5.86
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.789474	1.000000	1.000000	0.200000
2	100%	+	-0.431966	1.000000	0.630000	0.185000
3	100%	-	-1.186112	1.000000	0.830000	0.140000
4	91%	+	-0.845342	1.000000	0.620000	0.067500
5	88%	-	-0.486894	1.000000	0.420000	0.037500
Stage training time: 6.04
Number of used features: 5
Parent node: 11
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7	8	9	10	11	12


C:\WINDOWS\system32\cmd.exe
0---1---2---3---4---5---6---7---8---9--10--11--12
Parent node: 12
*** 1 cluster ***
POS: 200 200 1.000000
NEG: 200 2.88323e-006
BACKGROUND PROCESSING TIME: 384.89
Precalculation time: 5.51
N	%SMP	F	ST.THR	HR	FA	EXP. ERR
1	100%	-	-0.551570	1.000000	1.000000	0.192500
2	100%	+	-1.180392	1.000000	1.000000	0.192500
3	100%	-	-1.038361	1.000000	0.840000	0.180000
4	92%	+	-1.033026	1.000000	0.820000	0.170000
5	91%	-	-0.597103	1.000000	0.495000	0.110000
Stage training time: 5.69
Number of used features: 5
Parent node: 12
Chosen number of splits: 0
Total number of splits: 0
Tree Classifier
Stage
0	1	2	3	4	5	6	7	8	9	10	11	12	13



Step 7 - CONVERT TO XML

Once the training is completed, we need to run the `convert.bat` file in order to convert our data to a XML file.

A.2.6 Theory of Deep-Learning based approach

Workflow of Face Detection

Before we try to do face detection using YOLO framework, first we should make sure that we have installed OpenCV and Darknet successfully.

1. **Pre-trained model configuration update**: We take a pre-trained model configuration file (e.g, `yolov3.cfg`) and rename it to `yolo-obj.cfg`. Then we have to modify batch, subdivisions and classes (`batch=64`, `subdivisions=8` and `classes=1`) in several places. This `yolo-obj.cfg` file should be placed inside `\build\darknet\x64` directory.
2. **Class name preparation**: We have to create a new file `obj.names` which contains the class-name `face` and it should be put inside `\build\darknet\x64\data` directory.

3. **Class information preparation**: In this step, we need to create a new file `obj.data` inside `\build\darknet\x64\data` directory - which will contain the necessary class information for face like below:

```
classes= 1
train  = data/train.txt
valid  = data/test.txt
names = data/obj.names
backup = backup/
```

4. **Training image files preparation**: We have to put all the training image files (at least 2000 image files with face) inside `\build\darknet\x64\data\obj` directory. Then we need to generate annotation files by labeling each face on the training image files. Yolo_mark tool can be used for labeling purpose.

5. **Training information preparation**: In this step, we have to create a file `train.txt` inside `\build\darknet\x64\data` directory. Pre-trained weights (`darknet53.conv.74`) for the convolutional layers should be placed inside `\build\darknet\x64` directory.

6. **Training NN Model**: We initiate the training process by the entering the following command using Command Prompt window.

```
darknet.exe detector train data/obj.data yolo-obj.cfg
   darknet53.conv.74
```

   Usually 2000 iterations will be sufficient. The result (`yolo-obj_ final.weights`) will be obtained inside `\build\darknet\x64\backup` directory.

7. **Face Detection**: In order to detect face in an image (lets support `input.jpg`), we execute the following command in Command Prompt window.

```
darknet.exe detect yolo-obj.cfg backup/yolo-obj_final.weights
   input.jpg
```

Appendix B

Hardware Setup Requirements

B.1 Implementation of Designed System

B.1.1 Specifications of Webcams

Logitech C170

Figure B.1: Logitech C170

Camera parameter	**Value**
Optical resolution	True 640×480
Sensor type	CMOS
Sensor size	1.02 mm×0.76 mm
Diagonal field of view (FOV)	58°
Focal length	2.3 mm
Frame rate (max)	640×480 @ 30 FPS
Camera dimension	70.3 mm × 71 mm × 60.5 mm
Camera mass	75 g
Price	24.99 €

Table B.1: Specification of Logitech C170

Logitech C920 PRO HD

Figure B.2: Logitech C920 PRO HD

Camera parameter	**Value**
Optical resolution	True 1920×1080
Sensor type	CMOS
Sensor size	4.80 mm×3.60 mm
Diagonal field of view (FOV)	78°
Focal length	3.67 mm
Frame rate (max)	1080P @ 30 FPS
Camera dimension	126 mm × 45 mm × 73 mm
Camera mass	170 g
Price	83.90 €

Table B.2: Specification of Logitech C920

B.2 Camera Parameter Calculation

Haar-like feature dimension: 20×20 px (5.29×5.29 mm);

We consider **Logitech C920 PRO HD** webcam for necessary parameter calculation.

Focal length (**FL**): 3.67 mm;

Sensor dimension (**SD**): 4.80 mm × 3.60 mm;

Working distance (**WD**): 600 mm

We know,

$$\text{Field of view} = \frac{\text{Sensor dimension} \times \text{Working distance}}{\text{Focal length}} \tag{B.1}$$

$$\text{Surface coverage per pixel} = \frac{\text{Field of view}}{\text{Number of pixels}} \tag{B.2}$$

$$\text{Minimum required resolution} = \frac{\text{Feature dimension}}{\text{Surface coverage per pixel}} \tag{B.3}$$

So we calculate,

FOV (horizontal) $= \frac{4.80 \times 600}{3.67} = 784.74$ mm

Surface coverage per pixel (horizontal) $= \frac{784.74}{1920} = 0.4087$ mm

Minimum required resolution (horizontal) $= \frac{5.29}{0.4087} = 12.9435 \approx 48.92$ px

Again, we calculate,

FOV (vertical) $= \frac{3.60 \times 600}{3.67} = 588.56$ mm

Surface coverage per pixel (vertical) $= \frac{588.56}{1080} = 0.5449$ mm

Minimum required resolution (vertical) $= \frac{5.29}{0.5449} = 9.7082 \approx 36.69$ px

So, the required minimum resolution is 49×37 px approximately.

Appendix C

Software Development Activities

C.1 Computer Vision Programming

C.1.1 Facial Components Detection

Object Detection Method

Basically, **CascadeClassifier.DetectMultiScale** method finds rectangular regions in input image that possibly contains objects the cascade has been trained for and returns those regions as an array of rectangles. The method scans the input image several times at different scales. In each iteration, it considers overlapping regions in the image. It may also apply some heuristics to reduce the number of analyzed regions, such as Canny prunning. After it finishes and collects the passed rectangles, it groups them and returns an array of average rectangles for each large enough group [62]. The method's prototype syntax is written below:

```
public Rectangle[] DetectMultiScale(
        IInputArray image,
        double scaleFactor = 1.1,
        int minNeighbors = 3,
        Size minSize = null,
        Size maxSize = null
)
```

Pupil Detection Method

```
gray.ROI = eyeRect;
gray.PyrDown().PyrUp();
gray._ThresholdBinaryInv(new Gray(40), new Gray(255));
```

```
CircleF[] pupilCircles = gray.HoughCircles(new Gray(12), new
    Gray(26), 1.90, 10.0, 0, 0)[0];
```

C.1.2 Object Detection using YOLO Framework

Face Detection Method

The minimal implementation of face detection using `Alturos.Yolo` wrapper class can be written like below:

```
YoloWrapper yoloWrapper = new
    YoloWrapper("yolov3_face_config.cfg",
    "yolov3_face_weights.weights", "face.names");
MemoryStream memoryStream = new MemoryStream();
imagePictureBox.Image.Save(memoryStream, ImageFormat.Png);
yoloWrapper.Detect(memoryStream.ToArray());
```

Please note that `yolov3_face_config.cfg`, `yolov3_face_weights.weights` and `face.names` must be present in the same directory of the application.

The `Detect` method's prototype is written below:

```
public IEnumerable<YoloItem> Detect(byte[] imageData);
```

C.2 Behavioral Biometrics Programming

C.2.1 Behavioral Biometrics Monitoring Method

```
#region Behavioral Biometrics Monitor
/// <summary>
/// Method that enables necessary Hook Services for Mouse and
    Keystroke events.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BehavioralBiometricsMonitorStart(object sender,
    EventArgs e)
{
        // Enable the hook services for mouse activities
        HookManager.MouseDown += HookManager_MouseDown;
```

```
        HookManager.MouseDoubleClick +=
           HookManager_MouseDoubleClick;
        HookManager.MouseMove += HookManager_MouseMove;
        HookManager.MouseWheel += HookManager_MouseWheel;

        // Enable the hook services for keyboard activities
        HookManager.KeyDown += HookManager_KeyDown;
        HookManager.KeyUp += HookManager_KeyUp;
        HookManager.KeyPress += HookManager_KeyPress;
}
/// <summary>
/// Method that disables necessary Hook Services for Mouse and
   Keystroke events.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BehavioralBiometricsMonitorStop(object sender,
   EventArgs e)
{
        // Disable the hook services for mouse activities
        HookManager.MouseDown -= HookManager_MouseDown;
        HookManager.MouseDoubleClick -=
           HookManager_MouseDoubleClick;
        HookManager.MouseMove -= HookManager_MouseMove;
        HookManager.MouseWheel -= HookManager_MouseWheel;

        // Disable the hook services for keyboard activities
        HookManager.KeyDown -= HookManager_KeyDown;
        HookManager.KeyUp -= HookManager_KeyUp;
        HookManager.KeyPress -= HookManager_KeyPress;
}
#endregion
```

C.3 Proposed Methodology Programming

C.3.1 Facial Landmark Detection Method

```
/// <summary>
/// Initialization of FacemarkLBF parameters.
/// </summary>
public void InitializeLandmarkIdentificationParameters()
{
        _faceDetector = new
           CascadeClassifier(@"haarcascade_frontalface_default.xml");
        _fParams = new FacemarkLBFParams();
        _fParams.ModelFile = @"lbfmodel.yaml";
        _fParams.NLandmarks = 68;
        _fParams.InitShapeN = 10;
        _fParams.StagesN = 5;
        _fParams.TreeN = 6;
        _fParams.TreeDepth = 5;
        _facemark = new FacemarkLBF(_fParams);
        _facemark.LoadModel(_fParams.ModelFile);
}
/// <summary>
/// Identification of facial landmarks.
/// </summary>
private void FacialLandmarkIdentification()
{
        Image<Bgr, Byte> image = _frame.ToImage<Bgr, Byte>();
        Image<Gray, byte> grayImage = image.Convert<Gray, byte>();
        grayImage._EqualizeHist();
        VectorOfRect faces = new
           VectorOfRect(_faceDetector.DetectMultiScale(grayImage));
        VectorOfVectorOfPointF landmarks = new
           VectorOfVectorOfPointF();
        bool success = _facemark.Fit(grayImage, faces, landmarks);
        /*** Remaining code for this method ***/
```

```
}
```

C.3.2 Eye Landmark Detection Method

```
/// <summary>
/// Class containing all necessary constants for the application.
/// </summary>
static class Constants
{
        public const int LEFT_EYE_LANDMARK_POINT_RANGE_MIN = 36;
        public const int LEFT_EYE_LANDMARK_POINT_RANGE_MAX = 41;
        public const int RIGHT_EYE_LANDMARK_POINT_RANGE_MIN = 42;
        public const int RIGHT_EYE_LANDMARK_POINT_RANGE_MAX = 47;
        /*** Remaining constants of this class ***/
}
/*** Inside eye landmark identification method ***/
for (int index = 0; index < landmarks[i].Size; index++)
{
        if (index >= Constants.LEFT_EYE_LANDMARK_POINT_RANGE_MIN
           && index <= Constants.LEFT_EYE_LANDMARK_POINT_RANGE_MAX)
        {
                leftEyePointLandmark.Add(new
                   PointF(landmarks[i][index].X,
                   landmarks[i][index].Y));
        }
        if (index >= Constants.RIGHT_EYE_LANDMARK_POINT_RANGE_MIN
           && index <=
           Constants.RIGHT_EYE_LANDMARK_POINT_RANGE_MAX)
        {
                rightEyePointLandmark.Add(new
                   PointF(landmarks[i][index].X,
                   landmarks[i][index].Y));
        }
}
```

C.3.3 Eye Aspect Ratio Calculation Method

```
/// <summary>
/// Method that returns EAR (Eye Aspect Ratio) of a given eye
   landmark points.
/// </summary>
/// <param name="eyeLandmarkPoints">Eye Landmark Points.</param>
/// <returns>Eye Aspect Ratio.</returns>
public static double GetEyeAspectRatio(PointF[] eyeLandmarkPoints)
{
        PointF p1 = new PointF(eyeLandmarkPoints[0].X,
           eyeLandmarkPoints[0].Y);
        PointF p2 = new PointF(eyeLandmarkPoints[1].X,
           eyeLandmarkPoints[1].Y);
        PointF p3 = new PointF(eyeLandmarkPoints[2].X,
           eyeLandmarkPoints[2].Y);
        PointF p4 = new PointF(eyeLandmarkPoints[3].X,
           eyeLandmarkPoints[3].Y);
        PointF p5 = new PointF(eyeLandmarkPoints[4].X,
           eyeLandmarkPoints[4].Y);
        PointF p6 = new PointF(eyeLandmarkPoints[5].X,
           eyeLandmarkPoints[5].Y);

        double eyeAspectRatio = (GetDistance(p2.X, p2.Y, p6.X,
           p6.Y) + GetDistance(p3.X, p3.Y, p5.X, p5.Y))/(2 *
           GetDistance(p1.X, p1.Y, p4.X, p4.Y));
        return eyeAspectRatio;
}
```

C.3.4 Mouth Landmark Detection Method

```
/// <summary>
/// Class containing all necessary constants for the application.
/// </summary>
static class Constants
{
```

```
        public const int OUTER_EDGE_LIP_RANGE_MIN = 48;
        public const int OUTER_EDGE_LIP_RANGE_MAX = 59;
        /*** Remaining constants of this class ***/
}
/*** Inside mouth landmark identification method ***/
for (int index = 0; index < landmarks[i].Size; index++)
{
        if (index >= Constants.OUTER_EDGE_LIP_RANGE_MIN && index
           <= Constants.OUTER_EDGE_LIP_RANGE_MAX)
        {
                outerEdgeLipIntegerPointLandmark.Add(new
                   Point(Convert.ToInt32(landmarks[i][index].X),
                   Convert.ToInt32(landmarks[i][index].Y)));
                outerEdgeLipFloatingPointLandmark.Add(new
                   PointF(landmarks[i][index].X,
                   landmarks[i][index].Y));
        }
}
```

C.3.5 Mouth Aspect Ratio Calculation Method

```
/// <summary>
/// Method that returns Mouth (Mouth Aspect Ratio) of a given eye
   landmark points.
/// </summary>
/// <param name="outerLipLandmarkPoints">Outer lip landmark
   points.</param>
/// <returns>Mouth Aspect Ratio.</returns>
public static double GetMouthAspectRatio(PointF[]
   outerLipLandmarkPoints)
{
        PointF p1 = new PointF(outerLipLandmarkPoints[0].X,
           outerLipLandmarkPoints[0].Y);
        PointF p3 = new PointF(outerLipLandmarkPoints[2].X,
           outerLipLandmarkPoints[2].Y);
```

```
        PointF p5 = new PointF(outerLipLandmarkPoints[4].X,
            outerLipLandmarkPoints[4].Y);
        PointF p7 = new PointF(outerLipLandmarkPoints[6].X,
            outerLipLandmarkPoints[6].Y);
        PointF p9 = new PointF(outerLipLandmarkPoints[8].X,
            outerLipLandmarkPoints[8].Y);
        PointF p11 = new PointF(outerLipLandmarkPoints[10].X,
            outerLipLandmarkPoints[10].Y);
        double mouthAspectRatio = (GetDistance(p3.X, p3.Y, p11.X,
            p11.Y) + GetDistance(p5.X, p5.Y, p9.X, p9.Y)) / (2 *
            GetDistance(p1.X, p1.Y, p7.X, p7.Y));
        return mouthAspectRatio;
}
```

C.3.6 Inner Lip Distance Measurement Method

```
/// <summary>
/// Method that returns Inner Lip Distance.
/// </summary>
/// <param name="innerLipLandmarkPoints">Inner lip landmark
    points.</param>
/// <returns>Inner lip distance.</returns>
public static double GetInnerLipDistance(PointF[]
    innerLipLandmarkPoints)
{
        PointF p62 = new PointF(innerLipLandmarkPoints[2].X,
            innerLipLandmarkPoints[2].Y);
        PointF p66 = new PointF(innerLipLandmarkPoints[6].X,
            innerLipLandmarkPoints[6].Y);
        double innerLipDistance = GetDistance(p62.X, p62.Y, p66.X,
            p66.Y);
        return innerLipDistance;
}
```

C.3.7 ML Model (for Keystroke Dynamics Analysis) Training Method

```
/// <summary>
/// Action-listener for Button ("Train"). It starts ML model
   training for keystroke dynamics analysis.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void BtnTrain_Click(object sender, EventArgs e)
{
        MLContext mlContext = new MLContext();
        // Step 1: Collect and load data
        List<BiometricData> biometricDataList =
           BiometricDataActivity.Fetch();
        for(int index = 0; index < biometricDataList.Count;
           index++)
        {
              if (biometricDataList[index].averageHoldTime == 0
                 ||
                 biometricDataList[index].averageReleasePressDelay
                 == 0)
              {
                    biometricDataList.RemoveAt(index);
              }
        }
        KeystrokeDynamicsData[] keystrokeDynamicsData = new
           KeystrokeDynamicsData[biometricDataList.Count];
        for (int index = 0; index < biometricDataList.Count;
           index++)
        {
              keystrokeDynamicsData[index] = new
                 KeystrokeDynamicsData()
              {
                    AverageHoldTime = Convert.ToSingle
                    (biometricDataList[index].averageHoldTime),
                    AverageReleasePressDelay = Convert.ToSingle
```

```
                    (biometricDataList[index].averageReleasePressDelay)
            };
        }
        IDataView trainingData =
            mlContext.Data.LoadFromEnumerable(keystrokeDynamicsData);

        // Step 2: Create pipeline
        var pipeline =
            mlContext.Transforms.Concatenate("Features", new[] {
            "AverageHoldTime" })
                .Append(mlContext.Regression.Trainers.Sdca(labelColumnName:
                    "AverageReleasePressDelay",
                    maximumNumberOfIterations: 100));

        // Step 3: Train model
        var model = pipeline.Fit(trainingData);

        // Step 4: Save model
        if(File.Exists(@"model.zip"))
        {
            File.Delete(@"model.zip");
        }
        mlContext.Model.Save(model, trainingData.Schema,
            "model.zip");

        lblTrainingStatus.Text = "Training is successful!";
}
```