A Major Project Report on

ADVANCED DRIVERS DROWSINESS DETECTION AND WARNING SYSTEM FOR CRITICAL INFRASTRUCTURES

Submitted In partial fulfillment of the Requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

BY

SHAIK MANEESHA	**(17UJ1A0521)**
GURUBANI ANJALI	**(17UJ1A0534)**
K. SAI SATYAVATHI	**(17UJ1A0508)**
PAVAN RASHITH	**(17UJ1A0533)**
MANOJ KUMAR	**(18UJ5A0503)**

Under the Esteemed Guidance of
Mrs. K. ARCHANA, M.Tech
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MALLA REDDY COLLEGE OF ENGINEERING AND MANAGEMENT SCIENCE
(Approved by AICTE New Delhi & Affiliated to JNTU Hyderabad)Kistapur, Medchal, Medchal Dist- 501401.

2017-2021

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD,KUKATPALLY, HYDERABAD-85.

MALLA REDDY ENGINEERING COLLEGE AND MANAGEMENT SCIENCES
(Approved by AICTE New Delhi & Affiliated to JNTU Hyderabad)
Kistapur, Medchal, Medchal Dist- 501401.

CERTIFICATE

This is to certify that the major project report entitled "Advanced Drivers' Drowsiness Detection and Warning System for Critical Infrastructures " being submitted by

SHAIK MANEESHA	**(17UJ1A0521)**
GURUBANI ANJALI	**(17UJ1A0534)**
K. SAI SATYAVATHI	**(17UJ1A0508)**
PAVAN RASHITH	**(17UJ1A0533)**
MANOJ KUMAR	**(18UJ5A0503)**

in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the Jawaharlal Nehru Technological University , Hyderabad, during the academic year 2017-2021 is a record of bonafied work carried out under my guidance and supervision.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Internal Guide
Mrs. K. ARCHANA M.Tech.
Assistant Professor
Departmen of CSE.

Head of The Department
Dr. D . MAHAMMAD RAFI, M.Tech, Ph.D
Professor & Head of CSE
Department of CSE

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First and foremost, We express my deep debt of gratitude to the Chairperson and Managing Trustee **Mr.V. Malla Reddy** for their immense contribution in making this organization grow and providing me the state of the art facilities to do this project work.

Our heartfelt gratefulness to Director of MREM **Mr.L. Venugopal Reddy** for their deep commitment and dedication, to bring this institution to the peak in terms of discipline and values.

We owe my full satisfaction in our project work to the extensive hands of cooperation of Principal Academics **Dr.CNV.SRIDHAR , M.Tech,Ph.D.**

We would like to express my deep appreciation and sincere gratitude to my project internal guide , **Mrs. K. Archana**, Assistant Professor,Department of Computer Science & Engineering, Malla Reddy Engineering College and Management Sciences, for her guidance, support, encouragement, understanding and patience. We have been honored to work under her supervision and learn from her advice and useful insights throughout our project work.

We extend my gratitude to **Dr. D.Mahammad Rafi**, Professor and Head, Department of Computer Science and Engineering for his constant support to complete the project work.

We express my sincere thanks to all the teaching and non-teaching staffs of Malla Reddy Engineering College and Management Sciences who cooperated with me, which helped me to realize my dream.We would like to thank our internal project mates and department faculties for their full-fledged guidance and giving courage to carry out the project.We are very much thankful to one and all that helped me for the successful completion of our project.

SHAIK MANEESHA (**17UJ1A0521**)
GURUBANI ANJALI (**17UJ1A0534**)
K. SAI SATYAVATHI (**17UJ1A0508**)
PAVAN RASHITH (**17UJ1A0533**)
MANOJ KUMAR (**18UJ5A0503**)

ABSTRACT

The main intension of the proposed work is to detecting drowsiness by monitoring the eyes, and Driver fatigue in any vehicle accidents by observation of eye movements and blink patterns, using self-developed image-processing algorithm (python based algorithm) along with machine learning technique. In order to achieve promising results, python based algorithm monitors the drowsiness system activities, finds symptoms of driver fatigue, and detecting the localization of the eyes, which involves looking at the entire image of the eye, and determining the position of the eyes for avoiding road accidents and death. Driver fatigue is the main cause of traffic accidents. The development of technologies for detecting drowsiness at the wheel is a major challenge in the field of accident avoidance systems.

Initially, drowsiness and Driver fatigue detecting systems was utilized for capture the entire image of the eye as a input, and then extracting the features based on the facial image sequence. In this method, first, each facial image in the sequence is divided into nonoverlapping blocks of the same size, and Gabor wavelets are employed to extract multiscale and multiorientation features.

Considering the facial performance of human fatigue is a dynamic process that developed over time, each block's features are analyzed in the sequence. Finally, self-developed image-processing algorithm is applied to select the most discriminating fatigue features. The proposed method was tested on a self-built database which includes a wide range of human subjects of different genders, poses, and illuminations in real-life fatigue conditions. Experimental results show the effectiveness of the proposed method. The implementation is done by using Python open-source software programming language.

.**Keywords**—Driver drowsiness; eye detection; yawn detection; blink pattern; fatigue

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION

Traffic safety is a common problem faced by most of the countries in the world. Road traffic accidents each year come up to 10 million , accounting for about 90% of the global security incidents, a majority of which are caused by driver fatigue. So the ability to effectively identify driver fatigue and timely give reminder is the key to safe driving.

Current methods for the detection of driver fatigue can be generally summarized as the following three categories. The first method is identification based on physiological information, such as electroencephalogram (EEG), electrocardiogram (ECG), pulse, and blood pressure. The second method is identification through the behavior of the driver, including driver's grip force on the steering wheel, speed, acceleration, and braking . The first method obtains high recognition accuracy but is not easy to be accepted by drivers as it is intrusive to measure the heart and breathing rates and the brain activity, while the second method can be implemented nonintrusively, but it is subject to several limitations, including the driver experiences, vehicle type, and driving conditions. With the rapid development of computer vision technology, the third method, fatigue detection based on computer vision, is put forward, which uses optical sensors or cameras to get fatigue features and then analyzes whether a driver is in the state of fatigue. Due to its characteristic of nonintrusion and high recognition accuracy, this method becomes more and more practical and popular.

Through the review above, we find that the previous work focuses only on features of eyes or mouth. As a result, the extracted features could not well represent the state of driver fatigue. And the premise of the implementation of these methods is the position of the eyes and mouth that can be accurately located, which is difficult in

the real-life driving condition. Besides, in the previous work, the features in a single image were used to detect whether the driver is tired. In some cases these methods have

achieved good results, but the facial performance of human fatigue is a procedure that changes over time. Methods based on a single face image cannot reflect the dynamic process of fatigue, thus leading to a high false detecting rate. Therefore, in this paper we extract fatigue features from the whole human face by multiscale and multiorientation Gabor wavelets and then analyze the features in the image sequence. Finally, Adaboost algorithm is applied to select the most discriminating fatigue features.

1.2. FEATURES EXTRACTION BASED ON GABOR WAVELETS

Currently, most of the fatigue features extracted are geometry features of the eyes or mouth. However, accurate extraction of these features needs precise positioning, which is difficult in the real-life driving condition. This problem can be solved by analyzing the texture features of fatigue. In recent decades, many scholars have done a lot of researches to explore the methods of texture features extraction, including those based on cooccurrence matrix and wavelet transformation. Smith and Chang extracted texture features from wavelet subband and experiments show that it can achieve better classification results. Ma and Manjunath compared varieties of wavelet transformation, such as orthogonal wavelets, biorthogonal wavelets, tree structured wavelets, and Gabor wavelets. The results show that Gabor wavelets transform is optimal. Therefore, this paper uses Gabor wavelets to extract the texture features of fatigue.

1.3. GABOR WAVELETS TRANSFORMATION

They have been used in varieties of image-processing applications such as face recognition. The two-dimensional Gabor wavelets can be defined as where and , respectively, represent the orientation and scale of Gabor wavelets, is a complex operator, defines the bandwidth of wavelet filter, z=(x,y) is the pixel coordinates, $k_v=k_{max}/f^v$ is wavelet frequency, and represents wave vector.

There are many products out there that provide the measure of fatigue level in the drivers which are implemented in many vehicles. The driver drowsiness detection system provides the similar functionality but with better results and additional benefits. Also, it alerts the

user on reaching a certain saturation point of the drowsiness measure.

$$k_{u,v} = k_v(\cos\phi_u, \sin\phi_u)^T$$

The two-dimensional Gabor wavelets can be defined as

$$\psi_{u,v}(z) = \frac{\|k_{u,v}\|^2}{\sigma^2} \exp\left(-\frac{\|k_{u,v}\|^2\|z\|^2}{2\sigma^2}\right) \cdot \left(\exp\left(ik_{u,v} \cdot z\right) - \exp\left(-\frac{\sigma^2}{2}\right)\right),$$

When a subject is in fatigue, features of different facial behaviors have different scales. In order to extract all the important features of fatigue, the paper convolves the face image with 5-scale(v{0,1,2,3,4}) and 8-orientation (u{0,1,2.....7}) Gabor wavelets, defined as

$$G_{u,v}(z) = \psi_{u,v}(z) * I(z) = \iint \psi_{u,v}(z)\, I(z)\, dx\, dy,$$

where represents convolution operation, I(Z) is pixel value at the point z , and $G_{u,v}(Z)$ is the filtering result in orientation u and v scale .

Before Gabor wavelets are applied to each image in the sequence, original face images are preprocessed, gray-scaled, and normalized with a size of 64 × 64. Each image in the sequence is convolved with 40 Gabor wavelets, resulting in 40 multiscale and multiorientation feature images.

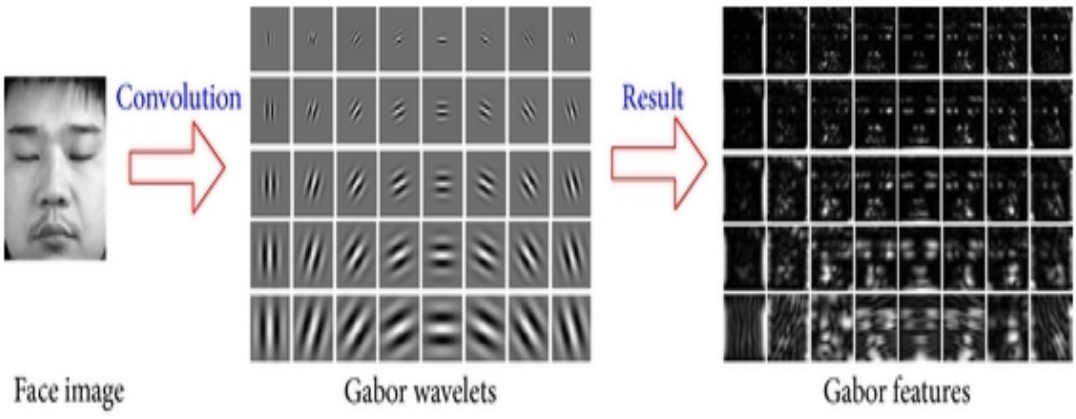


Figure 1.1: Multiscale and multiorientation features extraction of a face image.

1.3 HUMAN PSYCHOLOGY WITH CURRENT TECHNOLOGY

Humans have always invented machines and devised techniques to ease and protect their lives, for mundane activities like traveling to work, or for more interesting purposes like aircraft travel. With the advancement in technology, modes of transportation kept on advancing and our dependency on it started increasing exponentially. It has greatly affected our lives as we know it. Now, we can travel to places at a pace that even our grandparents wouldn't have thought possible. In modern times, almost everyone in this world uses some sort of transportation every day. Some people are rich enough to have their own vehicles while others use public transportation. However, there are some rules and codes of conduct for those who drive irrespective of their social status. One of them is staying alert and active while driving.

Neglecting our duties towards safer travel has enabled hundreds of thousands of tragedies to get associated with this wonderful invention every year. It may seem like a trivial thing to most folks but following rules and regulations on the road is of utmost importance. While on road, an automobile wields the most power and in irresponsible hands, it can be destructive and sometimes, that carelessness can harm lives even of the people on the road. One kind of carelessness is not admitting when we are too tired to drive. In order to monitor and prevent a destructive outcome from such negligence, many researchers have written research papers on driver drowsiness detection systems. But at times, some of the points and observations made by the system are not accurate enough. Hence, to provide data and another perspective on the problem at hand, in order to improve their implementations and to further optimize the solution, this project has been done.

1.4 FACTS & STATISTICS

Our current statistics reveal that just in 2015 in India alone, 148,707 people died due to car related accidents. Of these, at least 21 percent were caused due to fatigue causing drivers to make mistakes. This can be a relatively smaller number still, as among the multiple causes that can lead to an accident, the involvement of fatigue as a cause is generally grossly underestimated. Fatigue combined with bad infrastructure in developing countries like India is a recipe for disaster.

Fatigue, in general, is very difficult to measure or observe unlike alcohol and drugs, which have clear key indicators and tests that are available easily. Probably, the best solutions to this problem are awareness about fatigue-related accidents and promoting drivers to admit fatigue when needed. The former is hard and much more expensive to achieve, and the latter is not possible without the former as driving for long hours is very lucrative. When there is an increased need for a job, the wages associated with it increases leading to more and more people adopting it. Such is the case for driving transport vehicles at night. Money motivates drivers to make unwise decisions like driving all night even with fatigue. This is mainly because the drivers are not themselves aware of the huge risk associated with driving when fatigued. Some countries have imposed restrictions on the number of hours a driver can drive at a stretch, but it is still not enough to solve this problem as its implementation is very difficult and costly.

1.5 DROWSINESS DETECTION SYSTEM SCOPE

There are many products out there that provide the measure of fatigue level in the drivers which are implemented in many vehicles. The driver drowsiness detection system provides the similar functionality but with better results and additional benefits. Also, it alerts the user on reaching a certain saturation point of the drowsiness measure.

CHAPTER-2

LITERATURE SURVEY

2.1 LITERATURE SURVEY

M. H. Sigari (2009) the method based on computer vision has been working primarily through extracting the features of eyes and mouth. The most representative is Perclos (percentage of eyelid closure) method.

Wierwille et al. (1994) first used Perclos as metrics of the degree of fatigue in a research project funded by NHTSA to monitor driver fatigue. Then Dinges et al. (1998) proved that Perclos could accurately reflect the driver fatigue level. There are also other scholars trying to detect fatigue through other features of eyes.

Heitmann et al. (2001) tried to detect physical and mental condition of the driver through the driver's direction of the gaze and the change in diameter of the pupil, used eye blink frequency to detect fatigue , extracted features from eyes and mouth to monitor fatigue.

Liu et al. (2002) incorporated Kalman filter and mean shift to track eyes, extracting eye's motion information as driver features, used the distance of eyelids to decide whether the driver was in fatigue.

Daugman (1985) extended Gabor filter from one-dimensional to two-dimensional. The two-dimensional Gabor wavelets, a set of filter components of different orientations and different scales, can analyze gray change of the image in various scales and orientations.

Wang and Shi (2005) identified and located the mouth by a priori knowledge and then used the degree of mouth openness to determine driver yawning. Fan et al. (2007) extracted features through analysis of the degree of the mouth openness.

Chu et al. (2004) used the Fisher classifier to extract the mouth position and shape, then used the mouth region's geometry character as the feature value, and put all of these features together to make up an eigenvector as the input of a three-level BP network; then he got the output among three different spirit states: normal, speaking, and dozing.

Sl. No.	Title of the paper	Authors	Year	Technology used	Advantage	Disadvantage
1	A survey on Drowsy Driver Detection System	Kusumakumari.B.M	2017	• Steering wheel movement • Head pause • Frequntly Yawning	Different methods are used to detect the drowsiness and alert the driver	Road geometry influence the result.
2	Driver Behaviour Analysis for safe Driving	Ali Gokhan Yavuz	2015	• Facial Expression • Head-Movement • Eyelid Movement	Percentage of Eye closure is a reliable and valid metric to determine the alertness level	Lighting Condition Background.
3	Driver Drowsiness detection	Tejaswini Jagdale	2017	• GrayScale Image Processing • Image Blurring	Detect drowsiness and gives an alert	Slow in process.

LITERATURE SURVEY

In the Comparison of various drowsiness and Driver fatigue detecting systems as discussed in the Literature Survey, the MATLAB based Neural Network model performance seems to be more effective and is of less disadvantages than other models.

So, in this project work MATLAB based Neural Network model is considered as a better existing system and due to this reason, it has been used as a reference during the comparison of performance of drowsiness and Driver fatigue detecting systems with high accuracy.

MATLAB based Neural Network model increasing numbers of internal intrusion in the industry and tougher regulatory and compliance requirements,

organisations are facing tough challenges to protect both their sensitive image capture against internal eye scanning and meet regulatory and compliance requirements.

In existing system (MATLAB based Neural Network model) the driver drowsiness detection system involves controlling accident due to unconsciousness through Eye blink.Here one eye blink sensor is fixed in vehicle where if driver loses consciousness, then it alerts the driver through buzzer to prevent vehicle from accident.

2.2 DISADVANTAGES OF EXISTING SYSTEM

- Not Reliable
- May damage retina
- Highly expensive
- Intrusive
- Not portable
- Matlab based algorithm is costly and not efficient.
- It takes long time to process and not user friendly
- Which prevents further research and development of the system at affordable price from many free sources.

The Proposed self-developed image-processing algorithm (python based algorithm) using machine learning technique is the efficient algorithm to address these issues in various drowsiness and Driver fatigue detecting systems

CHAPTER-3

SYSTEM ANALYSIS

Python: Python is the basis of the program that we wrote. It utilizes many of the python libraries.

3.1 Libraries:

- Numpy: Pre-requisite for Dlib
- Scipy: Used for calculating Euclidean distance between the eyelids.
- Playsound: Used for sounding the alarm
- Dlib: This program is used to find the frontal human face and estimate its pose using 68 face landmarks.
- Imutils: Convenient functions written for Opencv.
- Opencv: Used to get the video stream from the webcam, etc.

3.2 OS: Program is tested on Windows 10 build 1903 and PopOS 19.04

Laptop: Used to run our code.

Webcam: Used to get the video feed.

3.3MODULES

SCIPY

SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

The additional benefit of basing SciPy on Python is that this also makes a powerful

programming language available for use in developing sophisticated programs and specialized applications. Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world. Everything from parallel programming to web and data-base subroutines and classes have been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy.

This tutorial will acquaint the first-time user of SciPy with some of its most important features. It assumes that the user has already installed the SciPy package. Some general Python facility is also assumed, such as could be acquired by working through the Python distribution's Tutorial. For further introductory help the user is directed to the Numpy documentation.

For brevity and convenience, we will often assume that the main packages (numpy, scipy, and matplotlib) have been imported as:

```
>>>
>>> import numpy as np
>>> import matplotlib as mpl
>>> import matplotlib.pyplot as plt
```

These are the import conventions that our community has adopted after discussion on public mailing lists. You will see these conventions used throughout NumPy and SciPy source code and documentation. While we obviously don't require you to follow these conventions in your own code, it is highly recommended.

SciPy Organization

SciPy is organized into subpackages covering different scientific computing domains. These are summarized in the following table:

Subpackage	Description
Cluster	Clustering algorithms
Constants	Physical and mathematical constants
Fftpack	Fast Fourier Transform routines

Subpackage	Description
Integrate	Integration and ordinary differential equation solvers
Interpolate	Interpolation and smoothing splines
Io	Input and Output
Linalg	Linear algebra
Ndimage	N-dimensional image processing
Odr	Orthogonal distance regression
Optimize	Optimization and root-finding routines
Signal	Signal processing
Sparse	Sparse matrices and associated routines
Spatial	Spatial data structures and algorithms
Special	Special functions
Stats	Statistical distributions and functions

Scipy sub-packages need to be imported separately, for example:

```
>>>
>>> from scipy import linalg, optimize
```

Because of their ubiquitousness, some of the functions in these subpackages are also made available in the scipy namespace to ease their use in interactive sessions and programs. In addition, many basic array functions from numpy are also available at the top-level of the scipy package. Before looking at the sub-packages individually, we will first look at some of these common functions.

Special functions (scipy.special)

The main feature of the scipy.special package is the definition of numerous special functions of mathematical physics. Available functions include airy, elliptic, bessel, gamma, beta, hypergeometric, parabolic cylinder, mathieu, spheroidal wave, struve, and kelvin. There are also some low-level stats functions that are not intended for general use as an easier interface to these functions is provided by the stats module. Most of these functions can take array

arguments and return array results following the same broadcasting rules as other math functions in Numerical Python. Many of these functions also accept complex numbers as input. For a complete list of the available functions with a one-line description type >>> help(special). Each function also has its own documentation accessible using help. If you don't see a function you need, consider writing it and contributing it to the library. You can write the function in either C, Fortran, or Python. Look in the source code of the library for examples of each of these kinds of functions.

Bessel functions of real order(jn, jn_zeros)

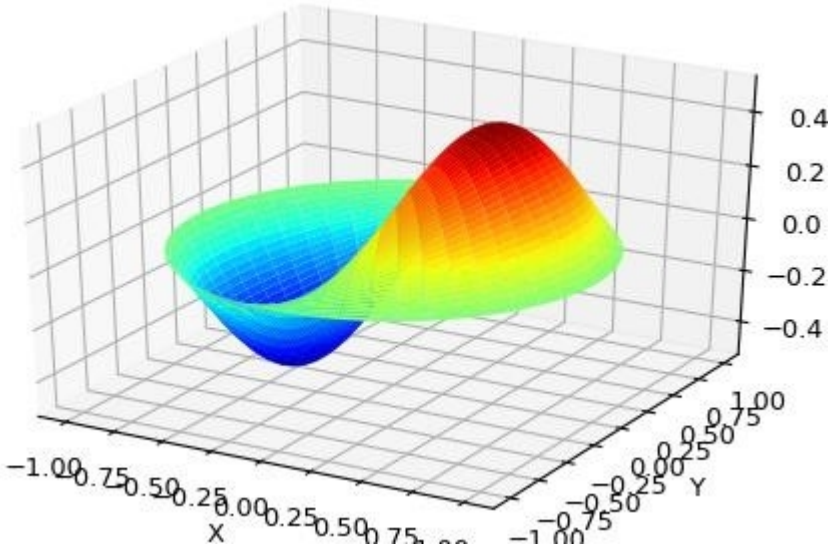
Bessel functions are a family of solutions to Bessel's differential equation with real or complex order alpha:

x2d2ydx2+xdydx+(x2−α2)y=0

Among other uses, these functions arise in wave propagation problems such as the vibrational modes of a thin drum head. Here is an example of a circular drum head anchored at the edge:

```
>>>
>>> from scipy import special
>>> def drumhead_height(n, k, distance, angle, t):
...     kth_zero = special.jn_zeros(n, k)[-1]
...     return np.cos(t) * np.cos(n*angle) * special.jn(n, distance*kth_zero)
>>> theta = np.r_[0:2*np.pi:50j]
>>> radius = np.r_[0:1:50j]
>>> x = np.array([r * np.cos(theta) for r in radius])
>>> y = np.array([r * np.sin(theta) for r in radius])
>>> z = np.array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])
>>>
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import Axes3D
>>> from matplotlib import cm
>>> fig = plt.figure()
```

```
>>> ax = Axes3D(fig)

>>> ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet)

>>> ax.set_xlabel('X')

>>> ax.set_ylabel('Y')

>>> ax.set_zlabel('Z')

>>> plt.show()
```



Cython Bindings for Special Functions (scipy.special.cython_special)

Scipy also offers Cython bindings for scalar, typed versions of many of the functions in special. The following Cython code gives a simple example of how to use these functions:

```
cimport scipy.special.cython_special as csc


cdef:
    double x = 1
    double complex z = 1 + 1j
    double si, ci, rgam
    double complex cgam
```

```
rgam = csc.gamma(x)

print(rgam)

cgam = csc.gamma(z)

print(cgam)

csc.sici(x, &si, &ci)

print(si, ci)
```

(See the Cython documentation for help with compiling Cython.) In the example the function csc.gamma works essentially like its ufunc counterpart gamma, though it takes C types as arguments instead of NumPy arrays. Note in particular that the function is overloaded to support real and complex arguments; the correct variant is selected at compile time. The function csc.sici works slightly differently from sici; for the ufunc we could write ai, bi = sici(x) whereas in the Cython version multiple return values are passed as pointers. It might help to think of this as analogous to calling a ufunc with an output array: sici(x, out=(si, ci)).

There are two potential advantages to using the Cython bindings:

- They avoid Python function overhead
- They do not require the Python Global Interpreter Lock (GIL)

The following sections discuss how to use these advantages to potentially speed up your code, though of course one should always profile the code first to make sure putting in the extra effort will be worth it.

Avoiding Python Function Overhead

For the ufuncs in special, Python function overhead is avoided by vectorizing, that is, by passing an array to the function. Typically this approach works quite well, but sometimes it is more convenient to call a special function on scalar inputs inside a loop, for example when implementing your own ufunc. In this case the Python function overhead can become significant. Consider the following example:

```
import scipy.special as sc

cimport scipy.special.cython_special as csc
```

```
def python_tight_loop():
cdef:
    int n
    double x = 1

  for n in range(100):
    sc.jv(n, x)

def cython_tight_loop():
  cdef:
    int n
    double x = 1

  for n in range(100):
    csc.jv(n, x)
```

and cython_tight_loop took about 18.2 microseconds to run. Obviously this example is contrived: one could just call special.jv(np.arange(100), 1) and get results just as fast as incython_tight_loop. The point is that if Python function overhead becomes significant in your code then the Cython bindings might be useful.

Releasing the GIL

One often needs to evaluate a special function at many points, and typically the evaluations are trivially parallelizable. Since the Cython bindings do not require the GIL, it is easy to run them in parallel using Cython's prange function. For example, suppose that we wanted to compute the fundamental solution to the Helmholtz equation:

$\Delta_x G(x,y) + k^2 G(x,y) = \delta(x-y)$,

where k is the wavenumber and δ is the Dirac delta function. It is known that in two dimensions the unique (radiating) solution is

G(x,y)=i4H0(1)(k|x−y|),

where H0(1) is the Hankel function of the first kind, i.e. the function hankel1. The following example shows how we could compute this function in parallel:

```
from libc.math cimport fabs

cimport cython

from cython.parallel cimport prange


import numpy as np

import scipy.special as sc

cimport scipy.special.cython_special as csc


def serial_G(k, x, y):

    return 0.25j*sc.hankel1(0, k*np.abs(x - y))


@cython.boundscheck(False)

@cython.wraparound(False)

cdef void _parallel_G(double k, double[:,:] x, double[:,:] y,

                      double complex[:,:] out) nogil:

    cdef int i, j


    for i in prange(x.shape[0]):

        for j in range(y.shape[0]):

            out[i,j] = 0.25j*csc.hankel1(0, k*fabs(x[i,j] - y[i,j]))
```

```
def parallel_G(k, x, y):
    out = np.empty_like(x, dtype='complex128')
    _parallel_G(k, x, y, out)
    return out
```

(For help with compiling parallel code in Cython see here.) If the above Cython code is in a file test.pyx, then we can write an informal benchmark which compares the parallel and serial versions of the function:

```
import timeit


import numpy as np


from test import serial_G, parallel_G


def main():
    k = 1
    x, y = np.linspace(-100, 100, 1000), np.linspace(-100, 100, 1000)
    x, y = np.meshgrid(x, y)


    def serial():
        serial_G(k, x, y)


    def parallel():
        parallel_G(k, x, y)


    time_serial = timeit.timeit(serial, number=3)
    time_parallel = timeit.timeit(parallel, number=3)
```

```
    print("Serial method took {:.3} seconds".format(time_serial))
    print("Parallel method took {:.3} seconds".format(time_parallel))


if__name__== "_main_":
    main()
```

On one quad-core computer the serial method took 1.29 seconds and the parallel method took 0.29 seconds.

Functions not in scipy.special

Some functions are not included in special because they are straightforward to implement with existing functions in NumPy and SciPy. To prevent reinventing the wheel, this section provides implementations of several such functions which hopefully illustrate how to handle similar functions. In all examples NumPy is imported as `np` and special is imported as `sc`.

The binary entropy function:

```
def binary_entropy(x):
    return -(sc.xlogy(x, x) + sc.xlog1py(1 - x, -x))/np.log(2)
```

A rectangular step function on [0, 1]:

```
def step(x):
    return 0.5*(np.sign(x) + np.sign(1 - x))
```

Translating and scaling can be used to get an arbitrary step function.

The ramp function:

```
def ramp(x):
    return np.maximum(0, x)
```

Opencv

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for

computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform.

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

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies.

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

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full- featured CUDAand OpenCL interfaces are being actively developed right now. There are

over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available:

- Core functionality (core) - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- Image Processing (imgproc) - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- Video Analysis (video) - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- Camera Calibration and 3D Reconstruction (calib3d) - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- 2D Features Framework (features2d) - salient feature detectors, descriptors, and descriptor matchers.
- Object Detection (objdetect) - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- High-level GUI (highgui) - an easy-to-use interface to simple UI capabilities.
- Video I/O (videoio) - an easy-to-use interface to video capturing and video codecs.
- ... some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

OpenCV was started at Intel in 1999 by Gary Bradsky and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle who won 2005 DARPA Grand Challenge. Later its active development continued under the support of Willow Garage, with

Gary Bradsky and Vadim Pisarevsky leading the project. Right now, OpenCV supports a lot of algorithms related to Computer Vision and Machine Learning and it is expanding day-by-day.

Currently OpenCV supports a wide variety of programming languages like C++, Python, Java etc and is available on different platforms including Windows, Linux, OS X, Android, iOS etc. Also, interfaces based on CUDA and OpenCL are also under active development for high-speed GPU operations.

OpenCV-Python is the Python API of OpenCV. It combines the best qualities of OpenCV C++ API and Python language.

OpenCV-Python

Python is a general purpose programming language started by Guido van Rossum, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python. This is how OpenCV-Python works, it is a Python wrapper around original C++ implementation.

And the support of Numpy makes the task more easier. Numpy is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the OpenCV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can combine it with OpenCV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So OpenCV-Python is an appropriate tool for fast prototyping of computer vision problems

Automatic Memory Management

OpenCV handles all the memory automatically.

First of all, std::vector, cv::Mat, and other data structures used by the functions and methods have destructors that deallocate the underlying memory buffers when needed. This means that the destructors do not always deallocate the buffers as in case of Mat. They take into account possible data sharing. A destructor decrements the reference counter associated with the matrix data buffer. The buffer is deallocated if and only if the reference counter reaches zero, that is, when no other structures refer to the same buffer. Similarly, when a Mat instance is copied, no actual data is really copied. Instead, the reference counter is incremented to memorize that there is another owner of the same data. There is also the Mat::clone method that creates a full copy of the matrix data.

Automatic Allocation of the Output Data

OpenCV deallocates the memory automatically, as well as automatically allocates the memory for output function parameters most of the time. So, if a function has one or more input arrays (cv::Mat instances) and some output arrays, the output arrays are automatically allocated or reallocated. The size and type of the output arrays are determined from the size and type of input arrays. If needed, the functions take extra parameters that help to figure out the output array properties.

The array frame is automatically allocated by the >> operator since the video frame resolution and the bit-depth is known to the video capturing module. The array edges is automatically allocated by the cvtColor function. It has the same size and the bit-depth as the input array. The number of channels is 1 because the color conversion code cv::COLOR_BGR2GRAY is passed, which means a color to grayscale conversion. Note that frame and edges are allocated only once during the first execution of the loop body since all the next video frames have the same resolution. If you somehow change the video resolution, the arrays are automatically reallocated.

The key component of this technology is the Mat::create method. It takes the desired array size and type. If the array already has the specified size and type, the method does nothing. Otherwise, it releases the previously allocated data, if any (this part involves decrementing the reference counter and comparing it with zero), and then allocates a new buffer of the required size. Most functions call the Mat::create method for each output array, and so the automatic output data allocation is implemented.

Some notable exceptions from this scheme are cv::mixChannels, cv::RNG::fill, and a few other functions and methods. They are not able to allocate the output array, so you have to do this in advance.

Saturation Arithmetics

As a computer vision library, OpenCV deals a lot with image pixels that are often encoded in a compact, 8- or 16-bit per channel, form and thus have a limited value range. Furthermore, certain operations on images, like color space conversions, brightness/contrast adjustments, sharpening, complex interpolation (bi-cubic, Lanczos) can produce values out of the available range. If you just store the lowest 8 (16) bits of the result, this results in visual artifacts and may affect a further image analysis. To solve this problem, the so-called saturation arithmetics is used. For example, to store r, the result of an operation, to an 8-bit image, you find the nearest value within the 0..255 range:

I(x,y)=min(max(round(r),0),255)

Similar rules are applied to 8-bit signed, 16-bit signed and unsigned types. This semantics is used everywhere in the library. In C++ code, it is done using the cv::saturate_cast<> functions that resemble standard C++ cast operations.

Fixed Pixel Types. Limited Use of Templates

Templates is a great feature of C++ that enables implementation of very powerful, efficient and yet safe data structures and algorithms. However, the extensive use of templates may dramatically increase compilation time and code size. Besides, it is difficult to separate an interface and implementation when templates are used exclusively. This could be fine for basic algorithms but not good for computer vision libraries where a single algorithm may span thousands lines of code. Because of this and also to simplify development of bindings

for other languages, like Python, Java, Matlab that do not have templates at all or have limited template capabilities, the current OpenCV implementation is based on polymorphism and runtime dispatching over templates. In those places where runtime dispatching would be too slow (like pixel access operators), impossible (generic cv::Ptr<> implementation), or just very inconvenient (cv::saturate_cast<>()) the current implementation introduces small template classes, methods, and functions. Anywhere else in the current OpenCV version the use of templates is limited.

Consequently, there is a limited fixed set of primitive data types the library can operate on. That is, array elements should have one of the following types:

- 8-bit unsigned integer (uchar)
- 8-bit signed integer (schar)
- 16-bit unsigned integer (ushort)
- 16-bit signed integer (short)
- 32-bit signed integer (int)
- 32-bit floating-point number (float)
- 64-bit floating-point number (double)
- a tuple of several elements where all elements have the same type (one of the above). An array whose elements are such tuples, are called multi-channel arrays, as opposite to the single-channel arrays, whose elements are scalar values. The maximum possible number of channels is defined by the CV_CN_MAX constant, which is currently set to 512.

For these basic types, the following enumeration is applied:

```
enum { CV_8U=0, CV_8S=1, CV_16U=2, CV_16S=3, CV_32S=4, CV_32F=5, CV_64F=6 };
```

Multi-channel (n-channel) types can be specified using the following options:

- CV_8UC1 ... CV_64FC4 constants (for a number of channels from 1 to 4)
- CV_8UC(n) ... CV_64FC(n) or CV_MAKETYPE(CV_8U, n) ... CV_MAKETYPE(CV_64F, n) macros when the number of channels is more than 4 or unknown at the compilation time.

Examples:

```
Mat mtx(3, 3, CV_32F); // make a 3x3 floating-point matrix
Mat cmtx(10, 1, CV_64FC2); // make a 10x1 2-channel floating-point
// matrix (10-element complex vector)
Mat img(Size(1920, 1080), CV_8UC3); // make a 3-channel (color) image
// of 1920 columns and 1080 rows.
Mat grayscale(image.size(), CV_MAKETYPE(image.depth(), 1)); // make a 1-channel image of
```

Arrays with more complex elements cannot be constructed or processed using OpenCV. Furthermore, each function or method can handle only a subset of all possible array types. Usually, the more complex the algorithm is, the smaller the supported subset of formats is. See below typical examples of such limitations:

- The face detection algorithm only works with 8-bit grayscale or color images.
- Linear algebra functions and most of the machine learning algorithms work with floating-point arrays only.
- Basic functions, such as cv::add, support all types.
- Color space conversion functions support 8-bit unsigned, 16-bit unsigned, and 32-bit floating-point types.

The subset of supported types for each function has been defined from practical needs and could be extended in future based on user requests.

InputArray and OutputArray

Many OpenCV functions process dense 2-dimensional or multi-dimensional numerical arrays. Usually, such functions take cppMat as parameters, but in some cases it's more convenient to use std::vector<> (for a point set, for example) or cv::Matx<> (for 3x3 homography matrix and such). To avoid many duplicates in the API, special "proxy" classes have been introduced. The base "proxy" class is cv::InputArray. It is used for passing read- only arrays on a function input. The derived from InputArray class cv::OutputArray is used to specify an output array for a function. Normally, you should not care of those intermediate types (and you should not declare variables of those types explicitly) - it will all just work automatically. You can assume that instead of InputArray/OutputArray you can always

use Mat, std::vector<>, cv::Matx<>, cv::Vec<> or cv::Scalar. When a function has an optional input or output array, and you do not have or do not want one, pass cv::noArray().

Error Handling

OpenCV uses exceptions to signal critical errors. When the input data has a correct format and belongs to the specified value range, but the algorithm cannot succeed for some reason (for example, the optimization algorithm did not converge), it returns a special error code (typically, just a boolean variable).

The exceptions can be instances of the cv::Exception class or its derivatives. In its turn, cv::Exception is a derivative of std::exception. So it can be gracefully handled in the code using other standard C++ library components.

Multi-threading and Re-enterability

The current OpenCV implementation is fully re-enterable. That is, the same function or the same methods of different class instances can be called from different threads. Also, the same Mat can be used in different threads because the reference-counting operations use the architecture-specific atomic instructions.

imutils

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

Installation

Provided you already have NumPy, SciPy, Matplotlib, and OpenCV already installed, the imutils package is completely pip-installable:

```
$ pip install imutils
```

Finding function OpenCV functions by name

OpenCV can be a big, hard to navigate library, especially if you are just getting started learning computer vision and image processing. The find_function method allows you to quickly search function names across modules (and optionally sub-modules) to find the function you are looking for.

Example:

Let's find all function names that contain the text contour:

```
import imutils

imutils.find_function("contour")
```

Output:

```
1. contourArea

2. drawContours

3. findContours

4. isContourConvex
```

The contourArea function could therefore be accessed via: cv2.contourArea

Translation

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

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

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

Example:

```
# translate the image x=25 pixels to the right and y=75 pixels up

translated = imutils.translate(workspace, 25, -75)
```

Dlib

Dlib is principally a C++ library, however, you can use a number of its tools from python applications. This page documents the python API for working with these dlib tools. If you haven't done so already, you should probably look at the python example programs first before consulting this reference. These example programs are little mini-tutorials for using dlib from python.

Detailed API Listing

dlib.**angle_between_lines**(*a: dlib.line*, *b: dlib.line*) → float

ensures

- returns the angle, in degrees, between the given lines. This is a number in the range [0 90].

dlib.**apply_cca_transform**(*m: dlib.matrix*, *v: dlib.sparse_vector*) → dlib.vector

requires

- max_index_plus_one(v) <= m.nr()

ensures

- returns trans(m)*v (i.e. multiply m by the vector v and return the result)

class dlib.**array**

This object represents a 1D array of floating point numbers. Moreover, it binds directly to the C++ type std::vector<double>.

__init__(**args*, ***kwargs*)

Overloaded function.

1. __init (self: dlib.array) -> None

2. __init (self: dlib.array, arg0: dlib.array) -> None

Copy constructor

3. __init (self: dlib.array, arg0: iterable) -> None

4. __init (self: dlib.array, arg0: object) -> None

append(*self: dlib.array*, *x: float*) → None

Add an item to the end of the list

clear(*self: dlib.array*) → None

count(*self: dlib.array*, *x: float*) → int

Return the number of times x appears in the list
extend(**args*, ***kwargs*)

Overloaded function.

1. extend(self: dlib.array, L: dlib.array) -> None

Extend the list by appending all the items in the given list
2. extend(self: dlib.array, arg0: list) -> None

insert(*self: dlib.array*, *i: int*, *x: float*) → None

Insert an item at a given position.

pop(**args*, ***kwargs*)

Overloaded function.

1. pop(self: dlib.array) -> float

Remove and return the last item
2. pop(self: dlib.array, i: int) -> float

Remove and return the item at index i
remove(*self: dlib.array*, *x: float*) → None
Remove the first item from the list whose value is x. It is an error if there is no such item.
resize(*self: dlib.array*, *arg0: int*) → None

dlib.**as_grayscale**(*img: array*) → array

Convert an image to 8bit grayscale. If it's already a grayscale image do nothing and just

return img.

dlib.**assignment_cost**(*cost: dlib.matrix*, *assignment: list*) → float

requires

- cost.nr() == cost.nc() (i.e. the input must be a square matrix)
- for all valid i:
    - 0 <= assignment[i] < cost.nr()

ensures

- Interprets cost as a cost assignment matrix. That is, cost[i][j] represents the cost of assigning i to j.

- Interprets assignment as a particular set of assignments. That is, i is assigned to assignment[i].
- returns the cost of the given assignment. That is, returns a number which is:

sum over i: cost[i][assignment[i]]

dlib.**auto_train_rbf_classifier**(**args*, ***kwargs*)

Overloaded function.

1. auto_train_rbf_classifier(x: dlib.vectors, y: dlib.array, max_runtime_seconds: float, be_verbose: bool=True) -> dlib._normalized_decision_function_radial_basis

requires

- y contains at least 6 examples of each class. Moreover, every element in y is either +1 or -1.
- max_runtime_seconds >= 0
- len(x) == len(y)
- all the vectors in x have the same dimension.

ensures

- This routine trains a radial basis function SVM on the given binary classification training data. It uses the svm_c_trainer to do this. It also uses find_max_global() and 6-fold cross-validation to automatically determine the best settings of the SVM's hyper parameters.
- Note that we interpret y[i] as the label for the vector x[i]. Therefore, the returned function, df, should generally satisfy sign(df(x[i])) == y[i] as often as possible.
- The hyperparameter search will run for about max_runtime and will print messages to the screen as it runs if be_verbose==true.

2. auto_train_rbf_classifier(x: numpy.ndarray[(rows,cols),float64], y: numpy.ndarray[float64], max_runtime_seconds: float, be_verbose: bool=True) -> dlib._normalized_decision_function_radial_basis

requires

- y contains at least 6 examples of each class. Moreover, every element in y is either +1 or -1.

- max_runtime_seconds >= 0
- len(x.shape(0)) == len(y)
- x.shape(1) > 0

ensures

- This routine trains a radial basis function SVM on the given binary classification training data. It uses the svm_c_trainer to do this. It also uses find_max_global() and 6-fold cross-validation to automatically determine the best settings of the SVM's hyper parameters.
- Note that we interpret y[i] as the label for the vector x[i]. Therefore, the returned function, df, should generally satisfy sign(df(x[i])) == y[i] as often as possible.
- The hyperparameter search will run for about max_runtime and will print messages to the screen as it runs if be_verbose==true.

dlib.**cca**(*L: dlib.sparse_vectors*, *R: dlib.sparse_vectors*, *num_correlations: int*, *extra_rank: int=5L*, *q: int=2L*, *regularization: float=0L*) → dlib.cca_outputs

requires

- num_correlations > 0
- len(L) > 0
- len(R) > 0
- len(L) == len(R)
- regularization >= 0
- L and R must be properly sorted sparse vectors. This means they must list their elements in ascending index order and not contain duplicate index values. You can use make_sparse_vector() to ensure this is true.

ensures

- This function performs a canonical correlation analysis between the vectors in L and R. That is, it finds two transformation matrices, Ltrans and Rtrans, such that row vectors in the transformed matrices L*Ltrans and R*Rtrans are as correlated as possible (note that in this notation we interpret L as a matrix with the input vectors in

its rows). Note also that this function tries to find transformations which produce num_correlations dimensional output vectors.

- Note that you can easily apply the transformation to a vector using apply_cca_transform(). So for example, like this:
    - apply_cca_transform(Ltrans, some_sparse_vector)
- returns a structure containing the Ltrans and Rtrans transformation matrices as well as the estimated correlations between elements of the transformed vectors.
- This function assumes the data vectors in L and R have already been centered (i.e. we assume the vectors have zero means). However, in many cases it is fine to use uncentered data with cca(). But if it is important for your problem then you should center your data before passing it to cca().
- This function works with reduced rank approximations of the L and R matrices. This makes it fast when working with large matrices. In particular, we use the dlib::svd_fast() routine to find reduced rank representations of the input matrices by calling it as follows: svd_fast(L, U,D,V, num_correlations+extra_rank, q) and similarly for R. This means that you can use the extra_rank and q arguments to cca() to influence the accuracy of the reduced rank approximation. However, the default values should work fine for most problems.
- The dimensions of the output vectors produced by L*#Ltrans or R*#Rtrans are ordered such that the dimensions with the highest correlations come first. That is, after applying the transforms produced by cca() to a set of vectors you will find that dimension 0 has the highest correlation, then dimension 1 has the next highest, and so on. This also means that the list of estimated correlations returned from cca() will always be listed in decreasing order.
- This function performs the ridge regression version of Canonical Correlation Analysis when regularization is set to a value > 0. In particular, larger values indicate the solution should be more heavily regularized. This can be useful when the dimensionality of the data is larger than the number of samples.
- A good discussion of CCA can be found in the paper "Canonical Correlation Analysis" by David Weenink. In particular, this function is implemented using

equations 29 and 30 from his paper. We also use the idea of doing CCA on a reduced rank approximation of L and R as suggested by Paramveer S. Dhillon in his paper "Two Step CCA: A new spectral method for estimating vector models of words".

class dlib.**cca_outputs**

Ltrans

Rtrans

__init__

x.__init (...) initializes x; see help(type(x)) for signature

correlations

dlib.**center**(**args*, ***kwargs*)

Overloaded function.

1. center(rect: dlib.rectangle) -> dlib.point

returns the center of the given rectangle

2. center(rect: dlib.drectangle) -> dlib.dpoint

returns the center of the given rectangle

dlib.**centered_rect**(**args*, ***kwargs*)

Overloaded function.

1. centered_rect(p: dlib.point, width: int, height: int) -> dlib.rectangle
2. centered_rect(p: dlib.dpoint, width: int, height: int) -> dlib.rectangle
3. centered_rect(rect: dlib.rectangle, width: int, height: int) -> dlib.rectangle
4. centered_rect(rect: dlib.drectangle, width: int, height: int) -> dlib.rectangle

dlib.**centered_rects**(*pts: dlib.points*, *width: int*, *height: int*) → dlib.rectangles

dlib.**chinese_whispers**(*edges: list*) → list

Given a graph with vertices represented as numbers indexed from 0, this algorithm takes a list of edges and returns back a list that contains a labels (found clusters) for each vertex. Edges are tuples with either 2 elements (integers presenting indexes of connected vertices) or 3 elements, where additional one element is float which presents distance weight of the edge). Offers direct access to dlib::chinese_whispers.

dlib.**chinese_whispers_clustering**(*descriptors: list*, *threshold: float*) → list

Takes a list of descriptors and returns a list that contains a label for each descriptor. Clustering is done using dlib::chinese_whispers.

class dlib.**chip_details**
WHAT THIS OBJECT REPRESENTS This object describes where an image chip is to be extracted from within another image. In particular, it specifies that the image chip is contained within the rectangle self.rect and that prior to extraction the image should be rotated counter-clockwise by self.angle radians. Finally, the extracted chip should have self.rows rows and self.cols columns in it regardless of the shape of self.rect. This means that the extracted chip will be stretched to fit via bilinear interpolation when necessary.

__init_(**args*, ***kwargs*)
Overloaded function.

1. __init (self: dlib.chip_details, rect: dlib.drectangle) -> None
2. __init (self: dlib.chip_details, rect: dlib.rectangle) -> None

ensures

- self.rect == rect_
- self.angle == 0
- self.rows == rect.height()
- self.cols == rect.width()

3. __init (self: dlib.chip_details, rect: dlib.drectangle, size: int) -> None
4. __init (self: dlib.chip_details, rect: dlib.rectangle, size: int) -> None

ensures

- self.rect == rect
- self.angle == 0
- self.rows and self.cols is set such that the total size of the chip is as close to size as possible but still matches the aspect ratio of rect.
- As long as size and the aspect ratio of of rect stays constant then self.rows and self.cols will always have the same values. This means that, for example, if you want all your chips to have the same dimensions then ensure that size is always the same and also that rect always has the same aspect ratio. Otherwise the calculated values of self.rows and self.cols may be different for different chips. Alternatively, you can use

the chip_details constructor below that lets you specify the exact values for rows and cols.

5. __init__ (self: dlib.chip_details, rect: dlib.drectangle, size: int, angle: float) -> None
6. __init__ (self: dlib.chip_details, rect: dlib.rectangle, size: int, angle: float) -> None

ensures

- self.rect == rect
- self.angle == angle
- self.rows and self.cols is set such that the total size of the chip is as close to size as possible but still matches the aspect ratio of rect.
- As long as size and the aspect ratio of of rect stays constant then self.rows and self.cols will always have the same values. This means that, for example, if you want all your chips to have the same dimensions then ensure that size is always the same and also that rect always has the same aspect ratio. Otherwise the calculated values of self.rows and self.cols may be different for different chips. Alternatively, you can use the chip_details constructor below that lets you specify the exact values for rows and cols.

7. __init__ (self: dlib.chip_details, rect: dlib.drectangle, dims: dlib.chip_dims) -> None
8. __init__ (self: dlib.chip_details, rect: dlib.rectangle, dims: dlib.chip_dims) -> None

ensures

- self.rect == rect
- self.angle == 0
- self.rows == dims.rows
- self.cols == dims.cols

9. __init (self: dlib.chip_details, rect: dlib.drectangle, dims: dlib.chip_dims, angle: float) -> None
10. __init (self: dlib.chip_details, rect: dlib.rectangle, dims: dlib.chip_dims, angle: float) -> None

ensures

- self.rect == rect
- self.angle == angle
- self.rows == dims.rows
- self.cols == dims.cols

11. __init (self: dlib.chip_details, chip_points: dlib.dpoints, img_points: dlib.dpoints, dims: dlib.chip_dims) -> None
12. __init (self: dlib.chip_details, chip_points: dlib.points, img_points: dlib.points, dims: dlib.chip_dims) -> None

requires

- len(chip_points) == len(img_points)
- len(chip_points) >= 2

ensures

- The chip will be extracted such that the pixel locations chip_points[i] in the chip are mapped to img_points[i] in the original image by a similarity transform. That is, if you know the pixelwize mapping you want between the chip and the original image then you use this function of chip_details constructor to define the mapping.
- self.rows == dims.rows
- self.cols == dims.cols
- self.rect and self.angle are computed based on the given size of the output chip (specified by dims) and the similarity transform between the chip and image (specified by chip_points and img_points).

angle

cols

rect

rows

class dlib.**chip_dims**

WHAT THIS OBJECT REPRESENTS This is a simple tool for passing in a pair of row and column values to the chip_details constructor.

__init__(*self: dlib.chip_dims*, *rows: int*, *cols: int*) → None

cols

rows

class dlib.**cnn_face_detection_model_v1**
This object detects human faces in an image. The constructor loads the face detection model from a file. You can download a pre-trained model

__call__(**args*, ***kwargs*)

Overloaded function.

1. __call (self: dlib.cnn_face_detection_model_v1, imgs: list, upsample_num_times: int=0L, batch_size: int=128L) -> std::vector<std::vector<dlib::mmod_rect, std::allocator<dlib::mmod_rect> >, std::allocator<std::vector<dlib::mmod_rect, std::allocator<dlib::mmod_rect> > > >

takes a list of images as input returning a 2d list of mmod rectangles

2. __call (self: dlib.cnn_face_detection_model_v1, img: array, upsample_num_times: int=0L) -> std::vector<dlib::mmod_rect, std::allocator<dlib::mmod_rect> >

Find faces in an image using a deep learning model.

- Upsamples the image upsample_num_times before running the face detector.

__init__(*self: dlib.cnn_face_detection_model_v1*, *filename: unicode*) → None

dlib.**convert_image**(**args*, ***kwargs*)

Overloaded function.

1. convert_image(img: numpy.ndarray[(rows,cols),uint8], dtype: unicode) -> array
2. convert_image(img: numpy.ndarray[(rows,cols),uint16], dtype: unicode) -> array
3. convert_image(img: numpy.ndarray[(rows,cols),uint32], dtype: unicode) -> array
4. convert_image(img: numpy.ndarray[(rows,cols),uint64], dtype: unicode) -> array
5. convert_image(img: numpy.ndarray[(rows,cols),int8], dtype: unicode) -> array
6. convert_image(img: numpy.ndarray[(rows,cols),int16], dtype: unicode) -> array
7. convert_image(img: numpy.ndarray[(rows,cols),int32], dtype: unicode) -> array
8. convert_image(img: numpy.ndarray[(rows,cols),int64], dtype: unicode) -> array
9. convert_image(img: numpy.ndarray[(rows,cols),float32], dtype: unicode) -> array
10. convert_image(img: numpy.ndarray[(rows,cols),float64], dtype: unicode) -> array

11. convert_image(img: numpy.ndarray[(rows,cols,3),uint8], dtype: unicode) -> array

Converts an image to a target pixel type. dtype must be a string containing one of the following:
uint8, int8, uint16, int16, uint32, int32, uint64, int64, float32, float, float64, double, or rgb_pixel
When converting from a color space with more than 255 values the pixel intensity is saturated at the minimum and maximum pixel values of the target pixel type. For example, if you convert a float valued image to uint8 then float values will be truncated to integers and values larger than 255 are converted to 255 while values less than 0 are converted to 0.

dlib.**convert_image_scaled**(**args*, ***kwargs*)

Overloaded function.

1. convert_image_scaled(img: numpy.ndarray[(rows,cols),uint8], dtype: unicode, thresh: float=4L) -> array
2. convert_image_scaled(img: numpy.ndarray[(rows,cols),uint16], dtype: unicode, thresh: float=4L) -> array
3. convert_image_scaled(img: numpy.ndarray[(rows,cols),uint32], dtype: unicode, thresh: float=4L) -> array
4. convert_image_scaled(img: numpy.ndarray[(rows,cols),uint64], dtype: unicode, thresh: float=4L) -> array
5. convert_image_scaled(img: numpy.ndarray[(rows,cols),int8], dtype: unicode, thresh: float=4L) -> array
6. convert_image_scaled(img: numpy.ndarray[(rows,cols),int16], dtype: unicode, thresh: float=4L) -> array
7. convert_image_scaled(img: numpy.ndarray[(rows,cols),int32], dtype: unicode, thresh: float=4L) -> array
8. convert_image_scaled(img: numpy.ndarray[(rows,cols),int64], dtype: unicode, thresh: float=4L) -> array
9. convert_image_scaled(img: numpy.ndarray[(rows,cols),float32], dtype: unicode, thresh: float=4L) -> array

10. convert_image_scaled(img: numpy.ndarray[(rows,cols),float64], dtype: unicode, thresh: float=4L) -> array
11. convert_image_scaled(img: numpy.ndarray[(rows,cols,3),uint8], dtype: unicode, thresh: float=4L) -> array

requires

- thresh > 0

ensures

- Converts an image to a target pixel type. dtype must be a string containing one of the following: uint8, int8, uint16, int16, uint32, int32, uint64, int64, float32, float, float64, double, or rgb_pixel

The contents of img will be scaled to fit the dynamic range of the target pixel type. The thresh parameter is used to filter source pixel values which are outliers. These outliers will saturate at the edge of the destination image's dynamic range.

- Specifically, for all valid r and c:
    - We scale img[r][c] into the dynamic range of the target pixel type. This is done using the mean and standard deviation of img. Call the mean M and the standard deviation D. Then the scaling from source to destination is performed using the following mapping:

let SRC_UPPER = min(M + thresh*D, max(img)) let SRC_LOWER = max(M - thresh*D, min(img)) let DEST_UPPER = max value possible for the selected dtype. let DEST_LOWER = min value possible for the selected dtype.

MAPPING: [SRC_LOWER, SRC_UPPER] -> [DEST_LOWER, DEST_UPPER]

Where this mapping is a linear mapping of values from the left range into the right range of values. Source pixel values outside the left range are modified to be at the appropriate end of the range.

class dlib.**correlation_tracker**

This is a tool for tracking moving objects in a video stream. You give it the bounding box of an object in the first frame and it attempts to track the object in the box from frame to frame.

This tool is an implementation of the method described in the following paper:

Danelljan, Martin, et al. 'Accurate scale estimation for robust visual tracking.' Proceedings of the British Machine Vision Conference BMVC. 2014.

__init__(*self: dlib.correlation_tracker*) → None

get_position(*self: dlib.correlation_tracker*) → dlib.drectangle

returns the predicted position of the object under track.

start_track(**args*, ***kwargs*)

Overloaded function.

1. start_track(self: dlib.correlation_tracker, image: array, bounding_box: dlib.drectangle) -> None

requires

- image is a numpy ndarray containing either an 8bit grayscale or RGB image.
- bounding_box.is_empty() == false

ensures

- This object will start tracking the thing inside the bounding box in the given image. That is, if you call update() with subsequent video frames then it will try to keep track of the position of the object inside bounding_box.
- #get_position() == bounding_box

2. start_track(self: dlib.correlation_tracker, image: array, bounding_box: dlib.rectangle) -> None

requires

- image is a numpy ndarray containing either an 8bit grayscale or RGB image.
- bounding_box.is_empty() == false

ensures

- This object will start tracking the thing inside the bounding box in the given image. That is, if you call update() with subsequent video frames then it will try to keep track of the position of the object inside bounding_box.
- #get_position() == bounding_box

update(**args*, ***kwargs*)

Overloaded function.

1. update(self: dlib.correlation_tracker, image: array) -> float

requires

- image is a numpy ndarray containing either an 8bit grayscale or RGB image.
- get_position().is_empty() == false (i.e. you must have started tracking by calling start_track())

ensures

- performs: return update(img, get_position())

2. update(self: dlib.correlation_tracker, image: array, guess: dlib.drectangle) -> float

requires

- image is a numpy ndarray containing either an 8bit grayscale or RGB image.
- get_position().is_empty() == false (i.e. you must have started tracking by calling start_track())

ensures

- When searching for the object in img, we search in the area around the provided guess.
- #get_position() == the new predicted location of the object in img. This location will be a copy of guess that has been translated and scaled appropriately based on the content of img so that it, hopefully, bounds the object in img.
- Returns the peak to side-lobe ratio. This is a number that measures how confident the tracker is that the object is inside #get_position(). Larger values indicate higher confidence.

3. update(self: dlib.correlation_tracker, image: array, guess: dlib.rectangle) -> float

requires

- image is a numpy ndarray containing either an 8bit grayscale or RGB image.
- get_position().is_empty() == false (i.e. you must have started tracking by calling start_track())

ensures

- When searching for the object in img, we search in the area around the provided guess.

- #get_position() == the new predicted location of the object in img. This location will be a copy of guess that has been translated and scaled appropriately based on the content of img so that it, hopefully, bounds the object in img.
- Returns the peak to side-lobe ratio. This is a number that measures how confident the tracker is that the object is inside #get_position(). Larger values indicate higher confidence.

dlib.**count_points_between_lines**(**args*, ***kwargs*)

Overloaded function.

1. count_points_between_lines(l1: dlib.line, l2: dlib.line, reference_point: dlib.dpoint, pts: dlib.points) -> float
2. count_points_between_lines(l1: dlib.line, l2: dlib.line, reference_point: dlib.dpoint, pts: dlib.dpoints) -> float

ensures

- Counts and returns the number of points in pts that are between lines l1 and l2. Since a pair of lines will, in the general case, divide the plane into 4 regions, we identify the region of interest as the one that contains the reference_point. Therefore, this function counts the number of points in pts that appear in the same region as reference_point.

dlib.**count_points_on_side_of_line**(**args*, ***kwargs*)

Overloaded function.

1. count_points_on_side_of_line(l: dlib.line, reference_point: dlib.dpoint, pts: dlib.points, dist_thresh_min: float=0L, dist_thresh_max: float=inf) -> int
2. count_points_on_side_of_line(l: dlib.line, reference_point: dlib.dpoint, pts: dlib.dpoints, dist_thresh_min: float=0L, dist_thresh_max: float=inf) -> int

ensures

- Returns a count of how many points in pts have a distance from the line l that is in the range [dist_thresh_min, dist_thresh_max]. This distance is a signed value that indicates how far a point is from the line. Moreover, if the point is on the same side as reference_point then the distance is positive, otherwise it is negative. So for example, If this range is [0, infinity] then this function counts how many points are on the same side of l as reference_point.

dlib.**count_steps_without_decrease**(*time_series: object*, *probability_of_decrease: float=0.51*) → int

requires

- time_series must be a one dimensional array of real numbers.
- 0.5 < probability_of_decrease < 1

ensures

- If you think of the contents of time_series as a potentially noisy time series, then this function returns a count of how long the time series has gone without noticeably decreasing in value. It does this by scanning along the elements, starting from the end (i.e. time_series[-1]) to the beginning, and checking how many elements you need to examine before you are confident that the series has been decreasing in value. Here, "confident of decrease" means the probability of decrease is >= probability_of_decrease.
- Setting probability_of_decrease to 0.51 means we count until we see even a small hint of decrease, whereas a larger value of 0.99 would return a larger count since it keeps going until it is nearly certain the time series is decreasing.
- The max possible output from this function is len(time_series).
- The implementation of this function is done using the dlib::running_gradient object, which is a tool that finds the least squares fit of a line to the time series and the confidence interval around the slope of that line. That can then be used in a simple statistical test to determine if the slope is positive or negative.

dlib.**count_steps_without_decrease_robust**(*time_series: object*, *probability_of_decrease: float=0.51*, *quantile_discard: float=0.1*) → int

requires

- time_series must be a one dimensional array of real numbers.
- 0.5 < probability_of_decrease < 1
- 0 <= quantile_discard <= 1

ensures

- This function behaves just like count_steps_without_decrease(time_series,probability_of_decrease) except that it

ignores values in the time series that are in the upper quantile_discard quantile. So for example, if the quantile discard is 0.1 then the 10% largest values in the time series are ignored.

CHAPTER-4

SYSTEM DESIGN

4.1 USE CASE DIAGRAM

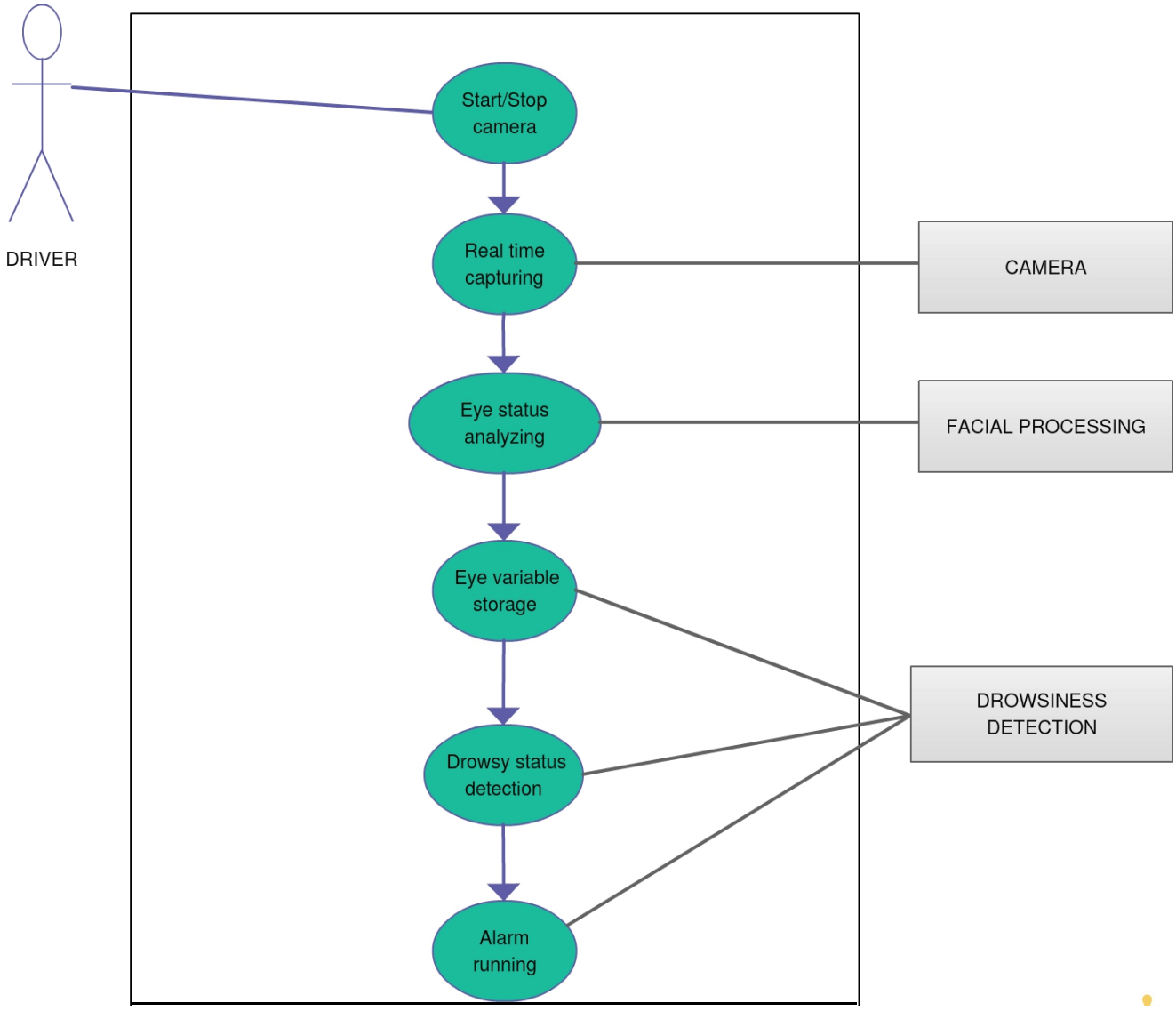


FIGURE 1

4.2 ACTIVITY DIAGRAM

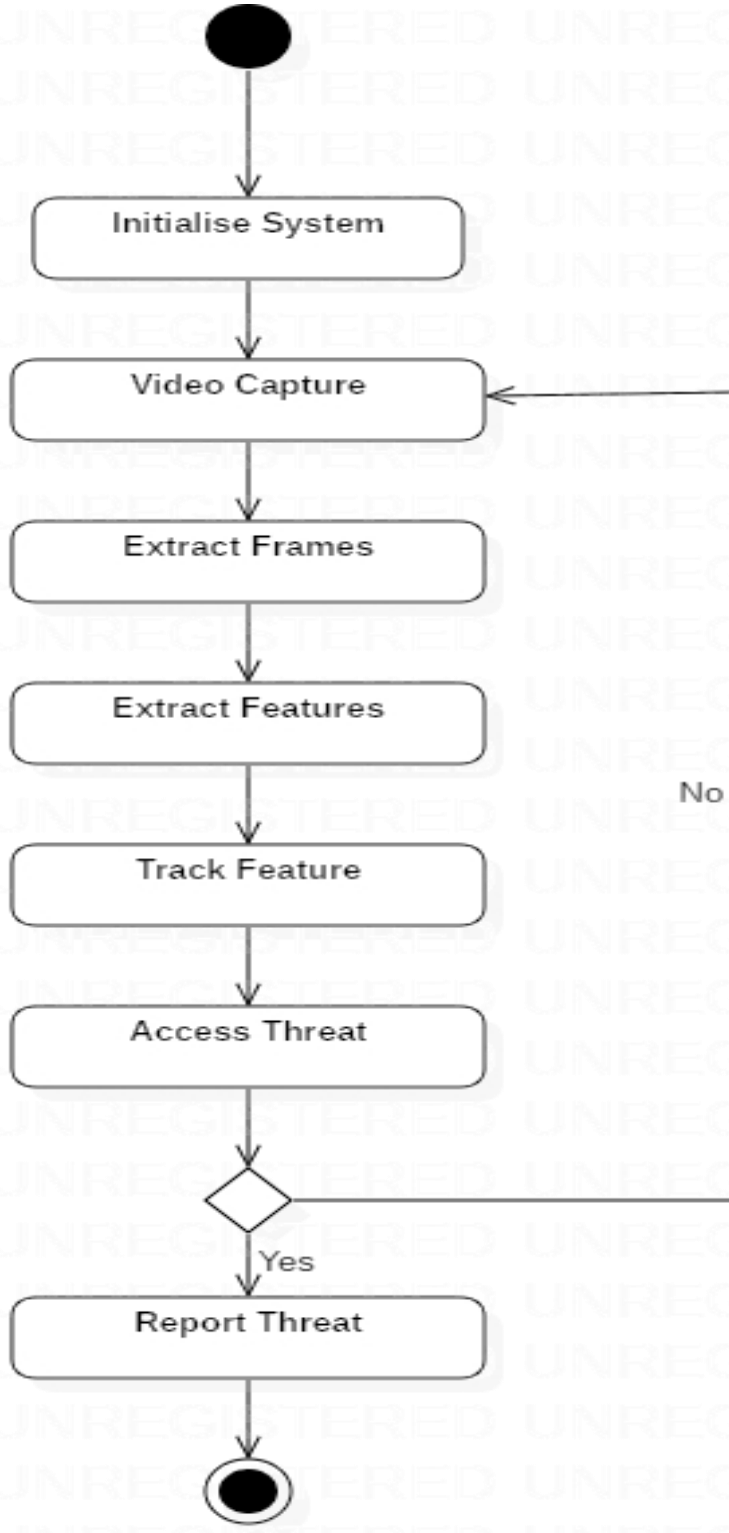


FIGURE 2

4.3 CLASS DIAGRAM

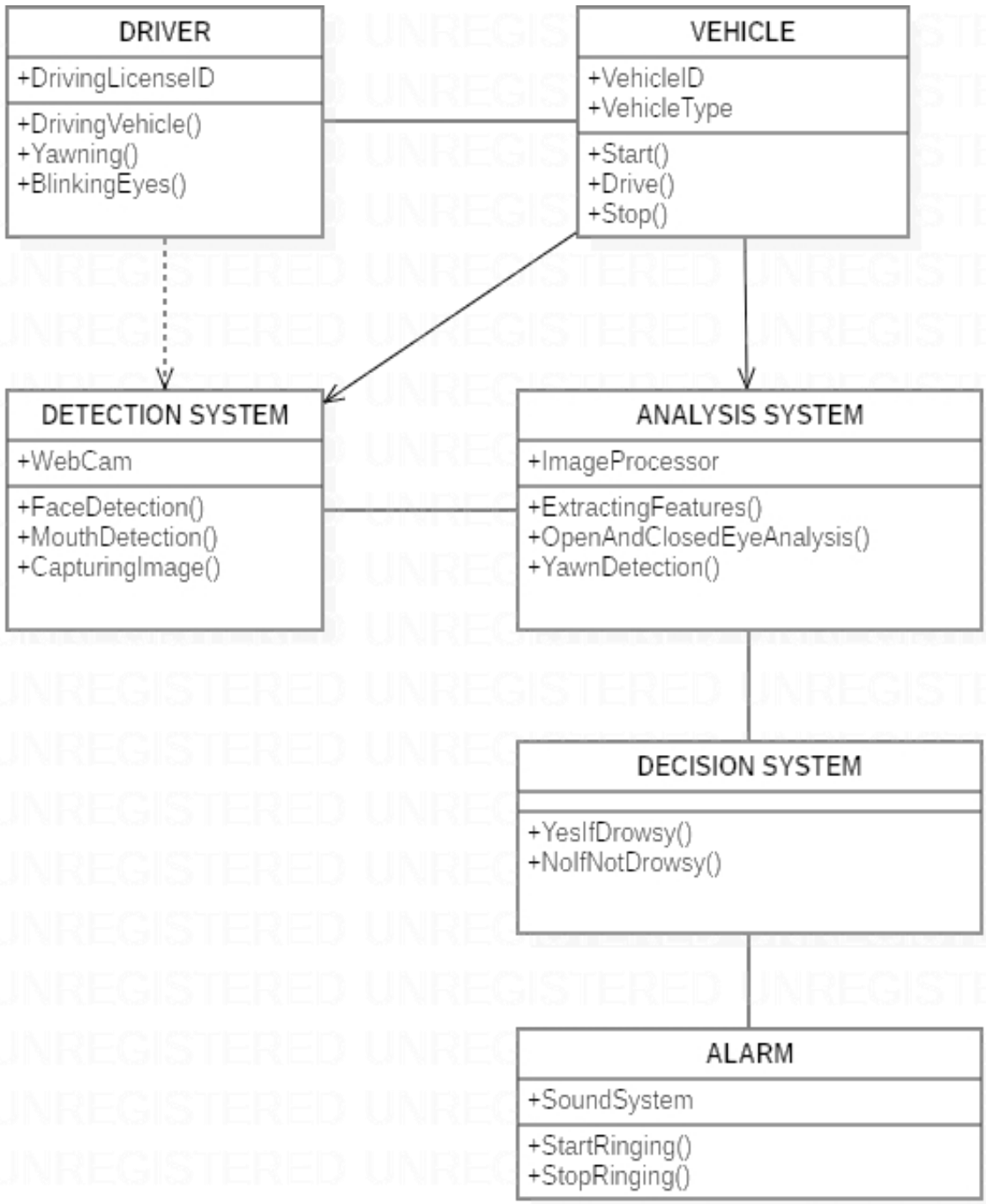


FIGURE 3

4.4 COLLABORATION DIAGRAM

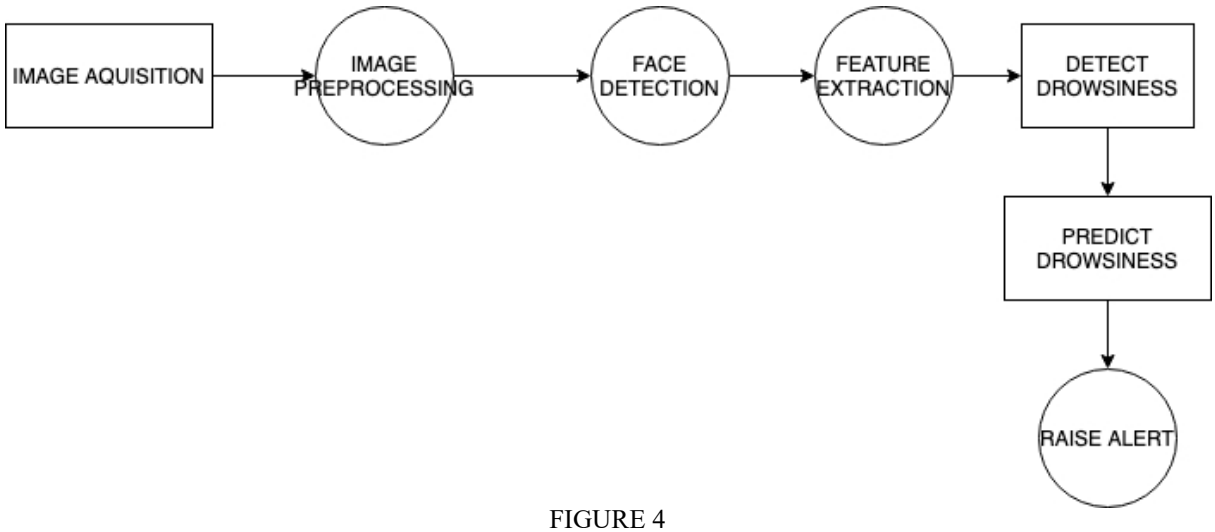


FIGURE 4

4.5 SEQUENCE DIAGRAM

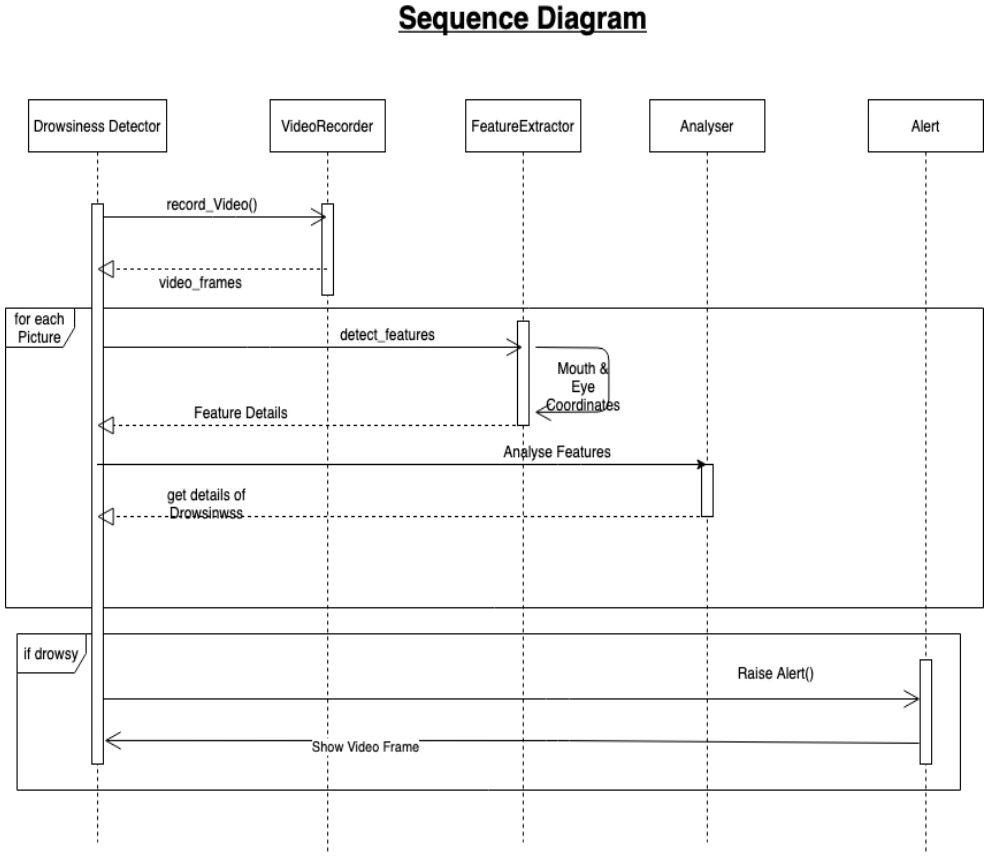


FIGURE 5

CHAPTER-5

SOFTWARE AND HARDWARE REQUIREMENTS

5.1 SOFTWARE REQUIREMENTS

5.1.1 Python:

- Python 3

Libraries

- Numpy
- Scipy
- Playsound
- Dlib
- Imutils
- opencv, etc.

5.1.2 Operating System

- Windows or Ubuntu

5.2 Hardware Requirements Specification

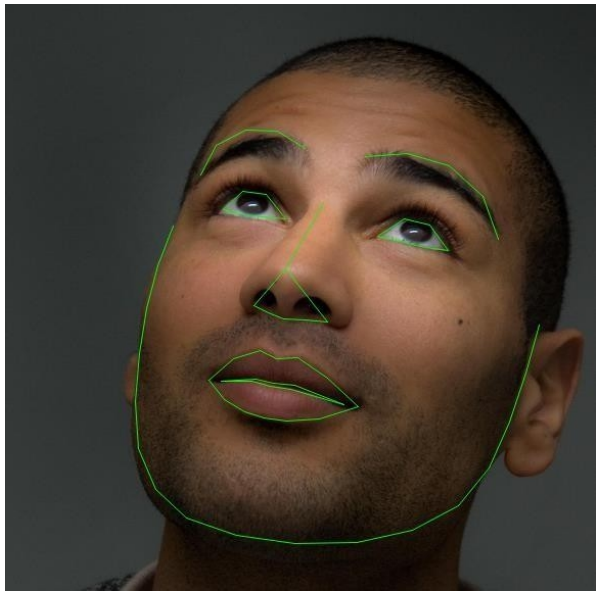
I. Laptop with basic hardware.
II. Webcam

CHAPTER-6

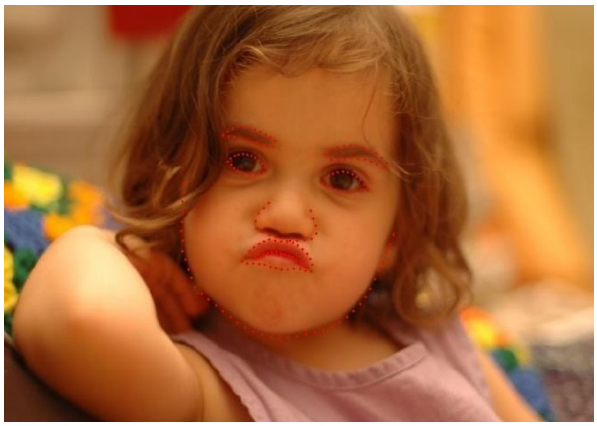
IMPLEMENTATION

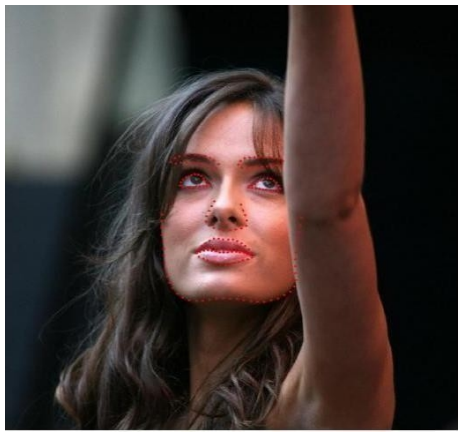
6.1 WORKING

- In our program we used Dlib, a pre-trained program trained on the HELEN dataset to detect human faces using the pre-defined 68 landmarks.



Landmarked Image of a person by Dlib

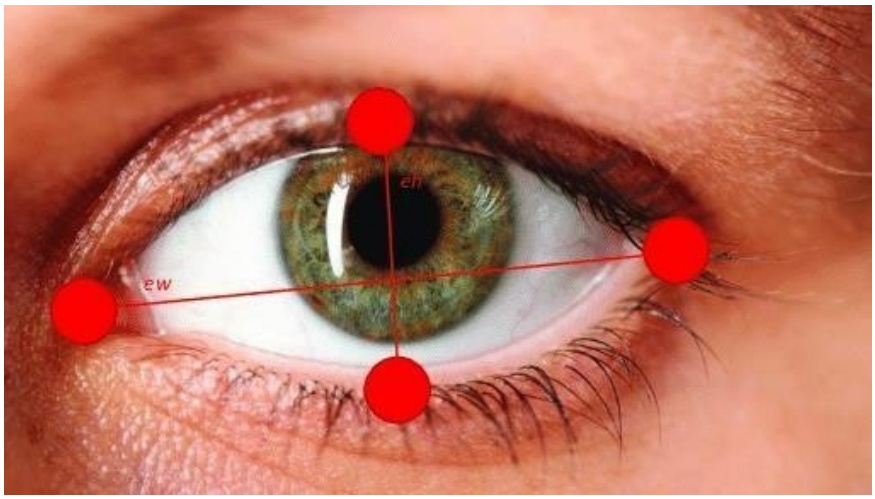




HELEN Dataset Samples

- After passing our video feed to the dlib frame by frame, we are able to detect left eye and right eye features of the face.

- Now, we drew contours around it using OpenCV.
- Using Scipy's Euclidean function, we calculated sum of both eyes' aspect ratio which is the sum of 2 distinct vertical distances between the eyelids divided by its horizontal distance.



Eyes with horizontal and vertical distance marked for Eye Aspect Ratio calculation.

- Now we check if the aspect ratio value is less than 0.25 (0.25 was chosen as a base case after some tests). If it is less an alarm is sounded and user is warned.

6.2 CODE

```
from imutils import face_utils
import imutils
import dlib
import cv2
import time
import threading
import math
from sklearn import tree
import pandas as pd
import numpy as np
import systemcheck
from playsound import playsound
```

```
chances = 0
drowsy = 0
siren = 0


endfps= 0

startfps=0


def training():
    a = pd.read_csv("blinkFatigue.csv")


    features = np.array(a['BPM']).reshape((len(a['BPM']),-1))

    labels = a['FATIGUE']

    clf = tree.DecisionTreeClassifier()

    clf = clf.fit(features, labels)

    return clf



def Euclidean_Distance(x,y):
        dis = math.sqrt(sum([(a - b) ** 2 for a, b in zip(x, y)]))

        return dis


def eye_aspect_ratio(eye):
        A = Euclidean_Distance(eye[1], eye[5])

        B = Euclidean_Distance(eye[2], eye[4])

        C = Euclidean_Distance(eye[0], eye[3])

        D = (A+B)**2
        ear = D / (2.0 * C)


        return ear


def mouth_aspect_ratio(mouth):
        A = Euclidean_Distance(mouth[0], mouth[6])

        A = A**2

        B = Euclidean_Distance(mouth[3], mouth[9])

        B = B**2
```

```
    ear = A/B

    return ear

blink = 0
yawn = 0
lastBlink = 0
blinkDur = 0
op = 0
timer1 = 0
thresh = 3.5
frame_check = 5
detect = dlib.get_frontal_face_detector()
# detect = dlib.cnn_face_detection_model_v1("human_face_detector.dat")
predict = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")# Dat
file is the crux of the code

(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["left_eye"] (rStart,
rEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["right_eye"] (mStart,
mEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["mouth"] (minStart,
                                        minEnd)                    =
face_utils.FACIAL_LANDMARKS_68_IDXS["inner_mouth"]
(leStart,                               leEnd)                     =
face_utils.FACIAL_LANDMARKS_68_IDXS["left_eyebrow"]
(reStart,                               reEnd)                     =
face_utils.FACIAL_LANDMARKS_68_IDXS["right_eyebrow"]
(nStart, nEnd) = face_utils.FACIAL_LANDMARKS_68_IDXS["nose"]

#print(face_utils.FACIAL_LANDMARKS_68_IDXS)

cap=cv2.VideoCapture(0)
flag=0
flag1 = 0
count = 0
start = time.time()
```

```
clf = training()

while True:

    siren = 0
    ret, frame= cap.read()
    count += 1
    #frame = imutils.resize(frame, width=450)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    subjects = detect(gray, 0)

    try:
        subject = list(subjects)[0]
        shape = predict(gray, subject)
        shape = face_utils.shape_to_np(shape)#converting to NumPy Array

        #print(len(shape))

        leftEye = shape[lStart:lEnd]
        leftEAR = eye_aspect_ratio(leftEye)

        rightEye = shape[rStart:rEnd]
        rightEAR = eye_aspect_ratio(rightEye)

        ear = (leftEAR + rightEAR) / 2.0

        #print('EAR :',ear)


        leftEyeHull = cv2.convexHull(leftEye)
        rightEyeHull = cv2.convexHull(rightEye)

        mouth = shape[mStart:mEnd]
        mouthHull = cv2.convexHull(mouth)
```

```
            mouthEAR = mouth_aspect_ratio(mouth)

            #print("Mouth Ratio" , mouthEAR)



nose = shape[nStart:nEnd]


noseHull = cv2.convexHull(nose)


            # jaw = shape[jStart:jEnd]
            # jawHull = cv2.convexHull(jaw)


            re = shape[reStart:reEnd]
            reHull = cv2.convexHull(re)


            le = shape[leStart:leEnd]
            leHull = cv2.convexHull(le)


            cv2.drawContours(frame, [leftEyeHull], -1, (255, 255, 0), 1)
            cv2.drawContours(frame, [rightEyeHull], -1, (255, 255, 0), 1)
            cv2.drawContours(frame, [mouthHull], -1, (255, 255, 0), 1)
            cv2.drawContours(frame, [noseHull], -1, (255, 255, 0), 1)
            #cv2.drawContours(frame, [jawHull], -1, (255, 255, 0), 1)
            cv2.drawContours(frame, [reHull], -1, (255, 255, 0), 1)
            cv2.drawContours(frame, [leHull], -1, (255, 255, 0), 1)
            # print('EAR: ', ear)
            if ear < thresh:


                if flag == 0 and time.time()-lastBlink > 1:
                    blink += 1
                    lastBlink = time.time()
                    print("Blink Detected", blink)


                # print (flag,end=' ')
                flag += 1
                if(flag > 10):
```

```
                cv2.putText(frame, " STAY ALERT ", (200, 250),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1)
                cv2.putText(frame, " DON'T SLEEP ", (200,400),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1)
                siren = 1


else:
            flag = 0


        if(time.time()- start > 30):
            print("Blink Per minute :",blink)
            p = np.array([blink]).reshape((1,-1))
            op = clf.predict(p)
            print('Chances of Drowsy :', op[0])
            start = time.time()
            blink = 0
            timer1 = 0


        if op[0] > 0:
            #print("Drowsy")
            cv2.putText(frame, " STAY ALERT ", (200, 250),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1)
            cv2.putText(frame, " YOU MAYBE SLEEPY ", (200,400),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 1)
            drowsy = 1
            siren = 1
            if(timer1 == 0):
                start= time.time()
                timer1 = 1


            elif(timer1 == 1 and ((time.time()- start) > 10)):
                op=0


        if(mouthEAR < 5):
            flag1 += 1
```

```
                if flag1 > frame_check:
                    yawn += 1
                    print("Yawn detected")
                    flag1 = 0

            else:

            flag1 = 0
        except:
            pass



        endfps = time.time()
        fps = int(1/(endfps-startfps))
        cv2.putText(frame, "FPS:- "+str(fps), (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 1)


        if(time.time() - start > 60):
            print('Yawns ', yawn)
            print('BPM : ', blink)


            if(yawn > 1 or (3 < blink < 6)):
                cv2.putText(frame, "Chances of Drowsiness Soon", (1, 100),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 1)
                chances = 1
                siren = 1


            yawn = 0
            blink = 0
            start = time.time()


        startfps = time.time()
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF
        if key == 27:
            break
```

```
        if siren == 1:
            playsound("siren.wav")
cv2.destroyAllWindows()
cap.release()
```

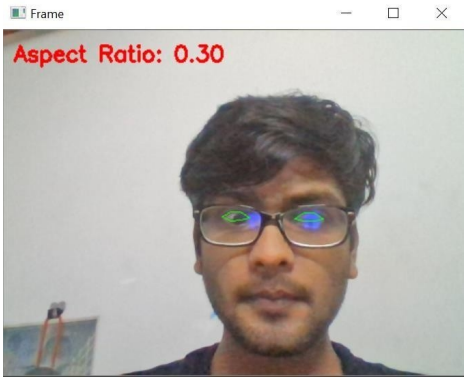
CHAPTER-7

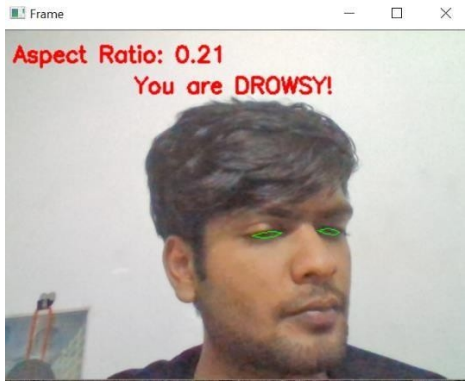
SYSTEM TESTING

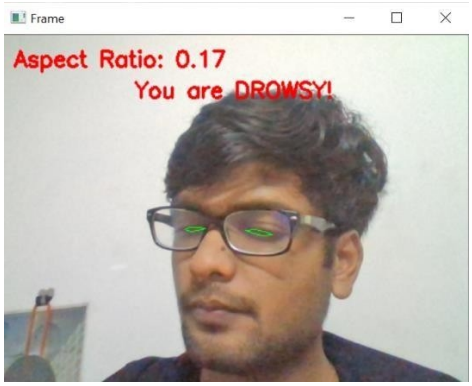
7.1 TEST CASES AND TEST RESULTS

Test ID	Test Case Title	Test Condition	System Behavior	Expected Result
T01	NSGY	Straight Face, Good Light, With Glasses	Non Drowsy	Non Drowsy
T02	YTGN	Tilted Face, Good Light, No Glasses	Drowsy	Drowsy
T03	YTGY	Tilted Face, Good Light, With Glasses	Drowsy	Drowsy

Note: Testing is performed manually



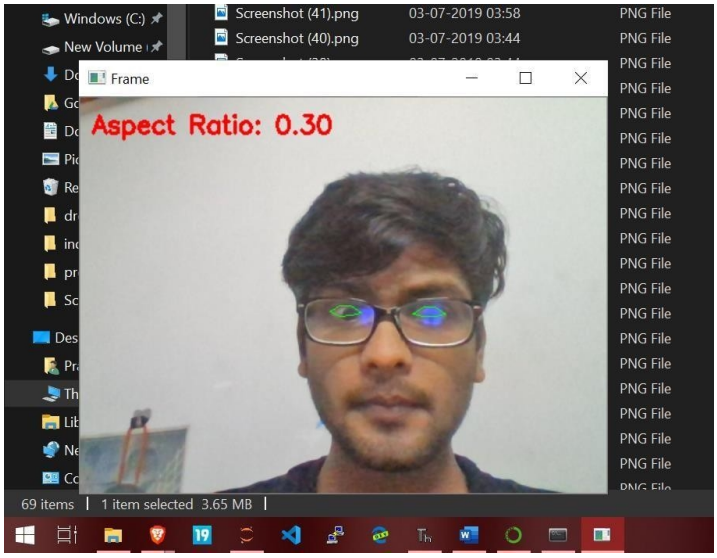




CHAPTER-8

OUTPUT SCREENS

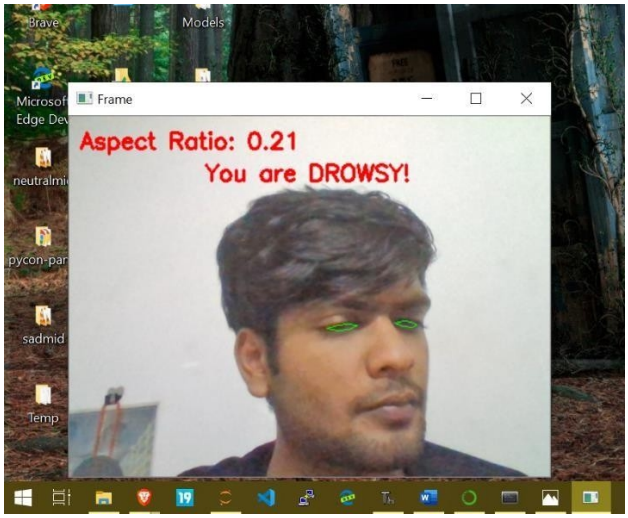
8.1 NON-DROWSY PERSON

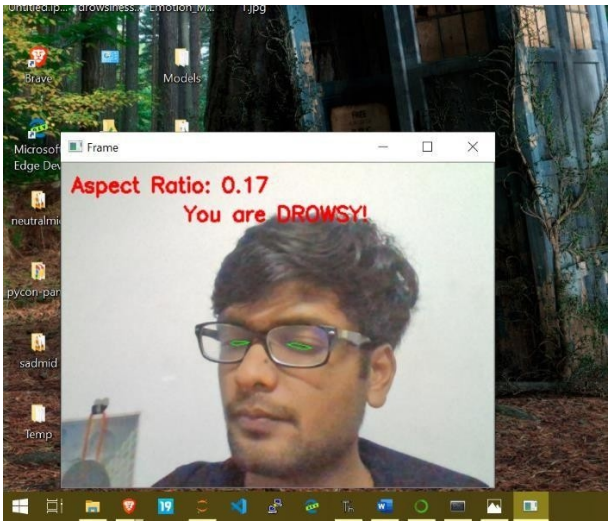


Non-drowsy state

8.2 DROWSY PERSON

Different aspect ratio at drowsy state





CHAPTER-9

CONCLUSION

It completely meets the objectives and requirements of the system. The framework has achieved an unfaltering state where all the bugs have been disposed of. The framework cognizant clients who are familiar with the framework and comprehend it's focal points and the fact that it takes care of the issue of stressing out for individuals having fatigue-related issues to inform them about the drowsiness level while driving.

A low cost, real time driver drowsiness monitoring system has been proposed based on visual behavior and machine learning Approach. Here, visual behavior features like eye aspect ratio, mouth opening ratio and nose length ratio are computed from the streaming video, captured by a webcam.

An self-developed image-processing algorithm (python based algorithm) using machine learning technique has been developed to detect driver drowsiness in real time.

CHAPTER-10

FUTURE SCOPE

The model can be improved incrementally by using other parameters like blink rate, yawning, state of the car, etc. If all these parameters are used it can improve the accuracy by a lot.

We plan to further work on the project by adding a sensor to track the heart rate in order to prevent accidents caused due to sudden heart attacks to drivers.

Same model and techniques can be used for various other uses like Netflix and other streaming services can detect when the user is asleep and stop the video accordingly. It can also be used in application that prevents user from sleeping.

The future works may focus on the utilization of outer factors such as vehicle states, sleeping hours, weather conditions, mechanical data etc, for fatigue measurement. Driver drowsiness pose a major threat to high to highway Safety, and the problem is particularly severe for commercial motor vehicle operations.Currently there is not adjustment in zoom or direction of the camera during operation. Future work may be to automatically zoom in on the eyes once they are localized.

CHAPTER-11

REFERENCES

- Computationally efficient facedetection; b. Schlkopf-a. Blake, s. Romdhani, and p. Torr Use of the hough transformation to detect lines and curves in picture; r. Duda and p. E. Hart..

- Jain, "face detection in color images; r. L. Hsu, m. Abdel- mottaleb, and a. K. Jain. Open/closed eye analysis for drowsiness detection;P.r. tabrizi and r. A. Zoroofi.

- W. L. Ou, M. H. Shih, C. W. Chang, X. H. Yu, C. P. Fan, "Intelligent Video-Based Drowsy Driver Detection System under Various Illuminations and Embedded Software Implementation", 2015 international Conf. on Consumer Electronics - Taiwan, 2015.

- W. B. Horng, C. Y. Chen, Y. Chang, C. H. Fan, "Driver Fatigue Detection based on Eye Tracking and Dynamic Template Matching", IEEE International Conference on Networking,, Sensing and Control, Taipei, Taiwan, March 21-23, 2004.

- S. Singh, N. P. papanikolopoulos, "Monitoring Driver Fatigue using Facial Analysis Techniques", IEEE Conference on Intelligent Transportation System, pp 314-318.