MAJOR PROJECT REPORT
ON
"DRIVER DROWSINESS DETECTION"
SUBMITTED IN PARTIAL FULFILLMENT OF THE DEGREE
OF
BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

SUBMITTED BY:
SANDEEP BHAISORA
150180101045
IV YEAR
B.T.K.I.T. DWARAHAT

UNDER THE GUIDANCE OF:
DR. VISHAL KUMAR
ASSISTANT PROFESSOR
COMPUTER SCIENCE AND ENGG.
B.T.K.I.T. DWARAHAT

DECLARATION

I hereby declare that the project work entitled **“Driver Drowsiness Detection”** submitted to the B.T.K.I.T., DWARAHAT, is a record of an original work done by me under the guidance of Dr. Vishal Kumar , ASSISTANT PROFESSOR (CSE DEPT.), and this project work is submitted in the fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Sandeep Bhaisora

150180101045

CERTIFICATE

This is to certify that the project entitled **"DRIVER DROWSINESS DETECTION"** is submitted by Mr. Sandeep Bhaisora to the department of Computer Science and Engineering, B.T.K.I.T DWARAHAT, Uttarakhand in partial fulfillment of the requirements of Bachelor of Technology in Computer Science and Engineering during final year. It is authentic and unique record of work carried out under our supervision and guidance and the matter embodied in this project is original and has not been submitted for the award of any other degree.

We wish them success for their future endeavors.

Project Guide & Project Head
Dr. Vishal kumar

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to **Dr. Vishal Kumar, Project guide and Project Head** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents, friends & all the faculty members of B.T.K.I.T, Dwarahat , for their kind co-operation and encouragement which help me in completion of this project.

Sincerely,

Sandeep Bhaisora

150180101045

Computer Science & Engineering

Contents

- Introduction.....7
- Background.....10
- Why OpenCV.....11
- Overview of Computer Vision.....15
- About OpenCV.....18
- Dtata flow diagram for algorithm.....22
- Facial Recognition Applications.....23
- Drowsiness Detection.....24
- Implementation.....29
- Result.....34
- Face Recognition Outputs.....36
- Limitations of algorithm.....38
- Face eye Recognition.....40
- Future work.....41
- Conclusion.....42
- References.....43

ABSTRACT

Driver fatigue is one of the major causes of accidents in the world. Detecting the drowsiness of the driver is one of the surest ways of measuring driver fatigue. In this project we aim to develop a prototype drowsiness detection system. This system works by monitoring the eyes of the driver and sounding an alarm when he/she is drowsy.

The system so designed is a non-intrusive real-time monitoring system. The priority is on improving the safety of the driver without being obtrusive. In this project the eye blink of the driver is detected. If the drivers eyes remain closed for more than a certain period of time, the driver is said to be drowsy and an alarm is sounded. The programming for this is done in OpenCV using the Haarcascade library for the detection of facial features.

INTRODUCTION

Driver fatigue is a significant factor in a large number of vehicle accidents. Recent statistics estimate that annually 1,200 deaths and 76,000 injuries can be attributed to fatigue related crashes.

The development of technologies for detecting or preventing drowsiness at the wheel is a major challenge in the field of accident avoidance systems. Because of the hazard that drowsiness presents on the road, methods need to be developed for counteracting its affects.

The aim of this project is to develop a prototype drowsiness detection system. The focus will be placed on designing a system that will accurately monitor the open or closed state of the driver's eyes in real-time.

By monitoring the eyes, it is believed that the symptoms of driver fatigue can be detected early enough to avoid a car accident. Detection of fatigue involves the observation of eye movements and blink patterns in a sequence of images of a face.

Initially, we decided to go about detecting eye blink patterns using Matlab. The procedure used was the geometric manipulation of intensity levels. The algorithm used was as follows.

First we input the facial image using a webcam. Preprocessing was first performed by binarizing the image. The top and sides of the face were detected to narrow down the area where the eyes exist. Using the sides of the face, the center of the face was found which will be used as a reference when .computing the left and right eyes. Moving down from the top of the face, horizontal averages of the face area were calculated. Large changes in the averages were used to define the eye area. There was little change

in the horizontal average when the eyes were closed which was used to detect a blink.

However Matlab had some serious limitations. The processing capacities required by Matlab were very high. Also there were some problems with speed in real time processing. Matlab was capable of processing only 4-5 frames per second. On a system with a low RAM this was even lower. As we all know an eye blink is a matter of milliseconds. Also a drivers head movements can be pretty fast. Though the Matlab program designed by us detected an eye blink, the performance was found severely wanting.

This is where OpenCV came in. OpenCV is an open source computer vision library.Itis designed for computational efficiency and with a strong focus on real time applications.Ithelps to build sophisticated vision applications quickly and easily. OpenCV satisfied the low processing power and high speed requirements of ourapplication.

We have used the Haar training applications in OpenCV to detect the face and eyes. This creates a classifier given a set of positive and negative samples. The steps were asfollows:-

Gather a data set of face and eye. These shouldbestored in one or more directories indexed by a text file. A lot of high quality data is required for the classifier to workwell.

- The utility application create samples() is used to build a vector output file. Using this file we can repeat the training procedure. It extracts the positive samples from images before normalizing and resizing to specified width and height.

The Viola Jones cascade decides whether or not the object in an image is similar to the training set. Any image that doesn't contain the object of interest can be turned into negative sample. So in order to learn any object

it is required to take a sample of negative background image. All these negative images are put in one file and then it's indexed.
Training of the image is done using boosting. In training we learn the group of classifiers one at a time. Each classifier in the group is a weak classifier. These weak classifiers are typically composed of a single variable decision tree called stumps. In training the decision stump learns its classification decisions from its data and also learns a weight for its vote from its accuracy on the data. Between training each classifier one by one, the data points are re weighted so that more attention is paid to the data points where errors were made. This process continues until the total error over the data set arising from the combined weighted vote of the decision trees falls below a certain threshold.[

This algorithm is effective when a large number of training data are available.

For our project face and eye classifiers are required. So we used the learning objects method to create our own haarclassifier .xml files.
Around 2000 positive and 3000 negative samples are taken. Training them is a time intensive process. Finally face.xml and haarcascade-eye.xml files are created.

These xml files are directly used for object detection. It detects a sequence of objects (in our case face and eyes). Haarcascade-eye.xml is designed only for open eyes. So when eyes are closed the system doesn't detect anything. This is a blink. When a blink lasts for more than 5 frames, the driver is judged to be drowsy and an alarm is sounded.

Background

Several studies have shown various possible techniques that can detect the driver drowsiness. Such driver drowsiness detection can be measured using physiological measures, ocular measure and performance measure. Among these physiological measure and ocular measure can give more accurate results. Physiological measure includes brain waves, heart rate, pulse rate measurements and these requires some sort of physical connection with the driver such as connecting electrode to the driver body. But this leads to discomfartable driving conditions. But ocular measure can be done without physical connection. Ocular measure to detect driver eye condition and possibl

e vision based on eye closure is well suited for real world driving conditions, since it can detect the eyes open/closed state non-intrusively using a camera. In Real Time Driver Drowsiness System using Image Processing, capturing drivers eye state using computer vision based drowsiness detection systems have been done by analyzing the interval of eye closure and developing an algorithm to detect the driver‟s drowsiness in advance and to warn

the driver by in vehicles alarm

WHY OPENCV

2.1 Eye blink detection usingMatlab:-

In our four year B.Tech career, of all the programming languages we had obtained the most proficiency in Matlab. Hence it was no surprise that we initially decided to do the project using Matlab. The procedure used was the geometric manipulation of intensity levels. The algorithm used was as follows.

First we input the facial image using a webcam. Preprocessing was first performed by binarizing the image. The top and sides of the face were detected to narrow down the area where the eyes exist. Using the sides of the face, the center of the face was found which will be used as a reference when computing the left and right eyes. Moving down from the top of the face, horizontal averages of the face area were calculated. Large changes in the averages were used to define the eye area. There was little change in the horizontal average when the eyes were closed which was used to detect a blink.

However Matlab had some serious limitations. The processing capacities required by Matlab were very high. Also there were some problems with speed in real time processing. Matlab was capable of processing only 4-5 frames per second. On a system with a low RAM this was even lower. As we all know an eye blink is a matter of milliseconds. Also a drivers head movements can be pretty fast. Though the Matlab program designed by us detected an eye blink, the performance was found severely wanting.

2.2 What IsOpenCV?

OpenCV [OpenCV] is an open source (see*http://opensource.org*) computer vision library available from*http://SourceForge.net/projects/opencvlibrary*.
OpenCV was designed for computational efficiency and having a high focus on real-time image detection. OpenCV is coded with optimized C and can take work with multicore processors. If we desire more automatic optimization using Intelarchitectures [Intel], you can buy Intel's Integrated Performance Primitives (IPP) libraries [IPP]. These consist of low-level routines in various algorithmic areas which are optimized. OpenCV automatically uses the IPP library, at runtime if that library is installed.
One of OpenCVs goals is to provide a simple-to-use computer vision infrastructure which helps people to build highly sophisticated vision applications fast. The OpenCV library, containing over 500 functions, spans many areas in vision. Because computer vision and machine learning oft en goes hand-in-hand,
OpenCV also has a complete, general-purpose, Machine Learning Library (MLL).
This sub library is focused on statistical pattern recognition and clustering.

The MLL is very useful for the vision functions that are the basis of OpenCV's usefulness, but is general enough to be used for any machine learning problem.

2.3 What Is ComputerVision?

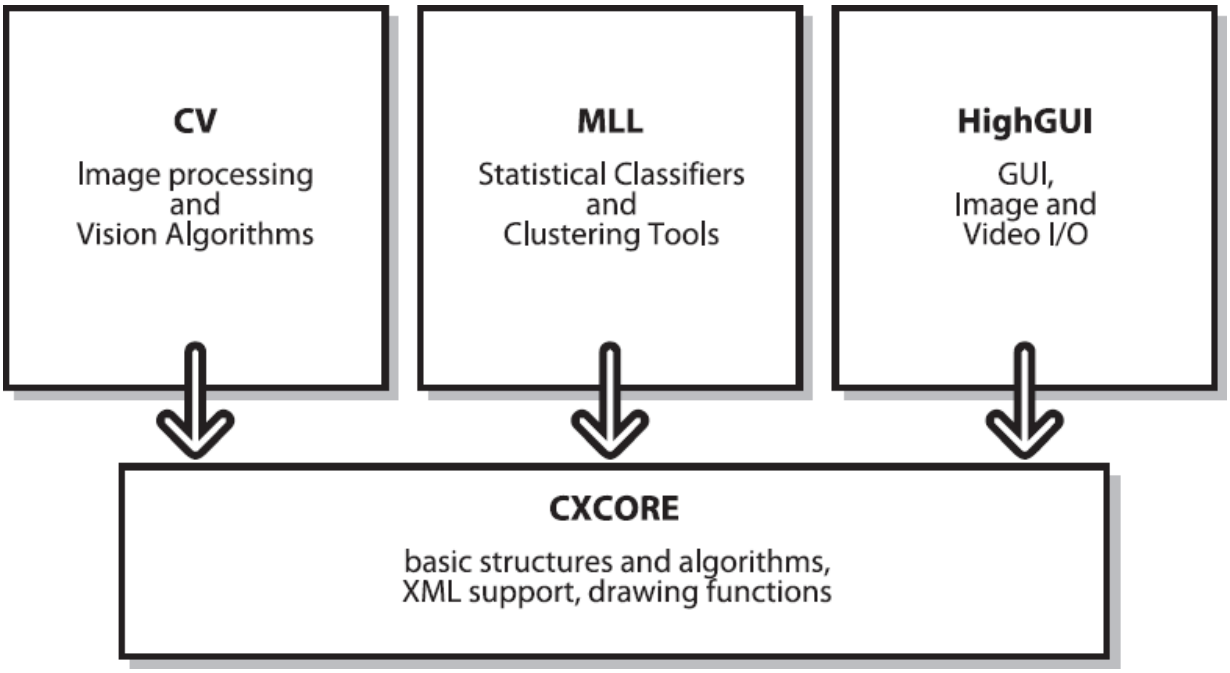
Computer vision is the transforming of data from a still, or video camera into either a representation or a new decision. All such transformations are performed to achieve a particular goal. A computer obtains a grid of numbers from a camera or from the disk, and that's that. Usually, there is no built in pattern recognition or automatic control of focus and aperture, no cross-associations with years of experience. For the most part, vision systems are still fairly naïve.

2.4 The Origin ofOpenCV

OpenCV came out of an Intel Research initiative meant to advance CPU-intensive applications. Toward this end, Intel launched various projects that included real-time ray tracing and also 3D display walls. One of the programmers working for Intel at the time was visiting universities. He noticed that a few top university groups, like the MIT Media Lab, used to have well-developed as well as internally open computer vision infrastructures—code that was passed from one student to another and which gave each subsequent student a valuable foundation while developing his own vision application. Instead of having to reinvent the basic functions from beginning, new student may start by adding to that which came before.

2.5 OpenCV Structure andContent

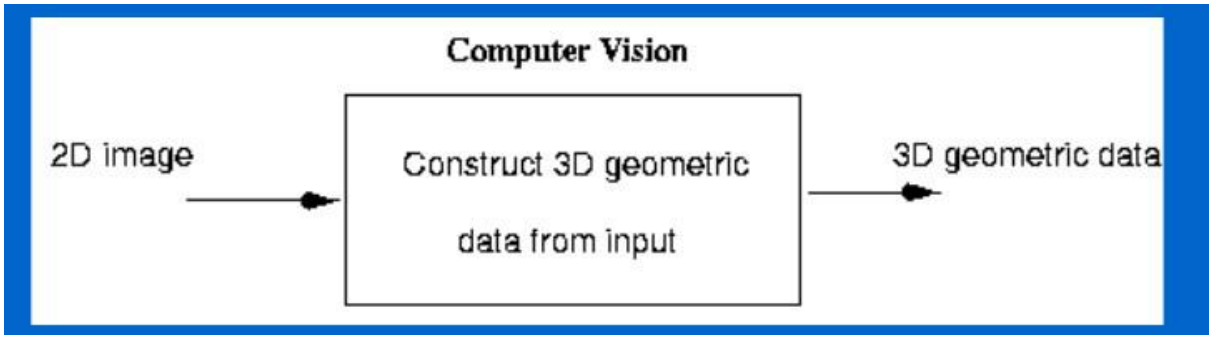
OpenCV can be broadly structured into five primary components, four of which are shown in the figure. The CV component contains mainly the basic image processing and higher-level computer vision algorithms; MLL the machine learning library includes many statistical classifiers as well as clustering tools. HighGUI component contains I/O routines with functions for storing, loading video & images, while CXCore contains all the basic data structures and content.


CV
Image processing
and
Vision Algorithms
MLL
Statistical Classifiers
and
Clustering Tools
HighGUI
GUI,
Image and
Video I/O
CXCORE
basic structures and algorithms,
XML support, drawing functions

Overview of Computer Vision

What is Computer Vision?

• Deals with the development of the theoretical and algorithmic basis by which useful information about the 3D world can be automatically extracted and analyzed from a single or multiple 2D images of the world.



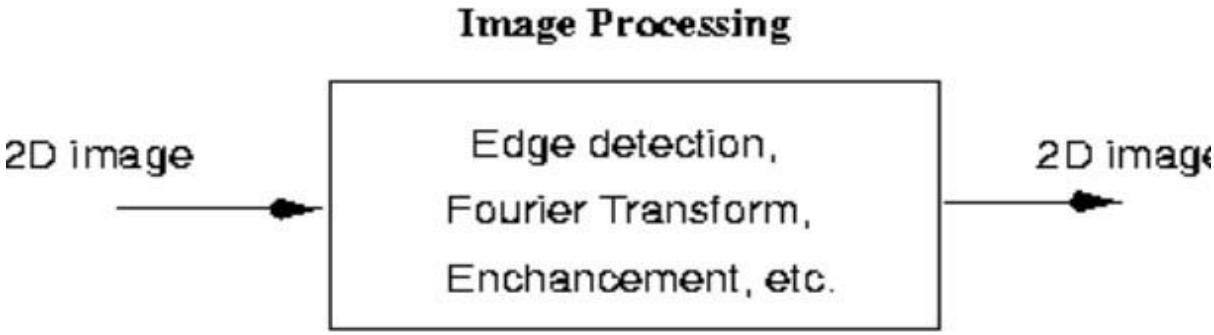
Computer Vision, Also Known As ..

- Image Analysis
- Scene Analysis
- Image Understanding

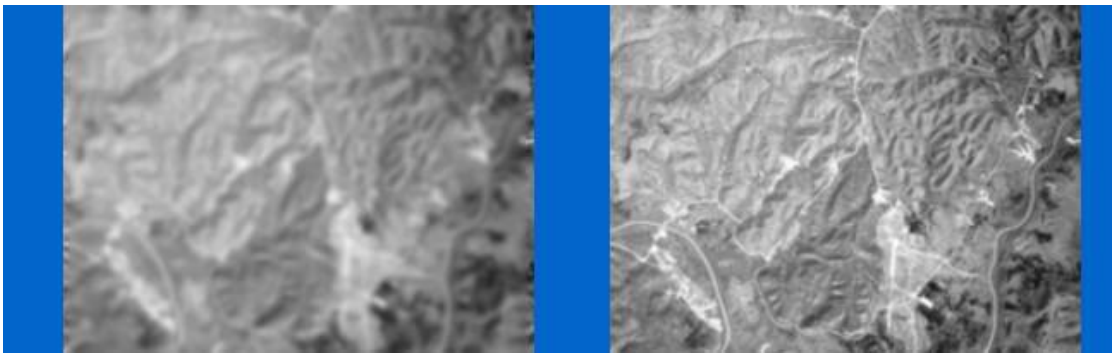
Some Related Disciplines

- Image Processing
- Computer Graphics
- Pattern Recognition
- Robotics
- Artificial Intelligence

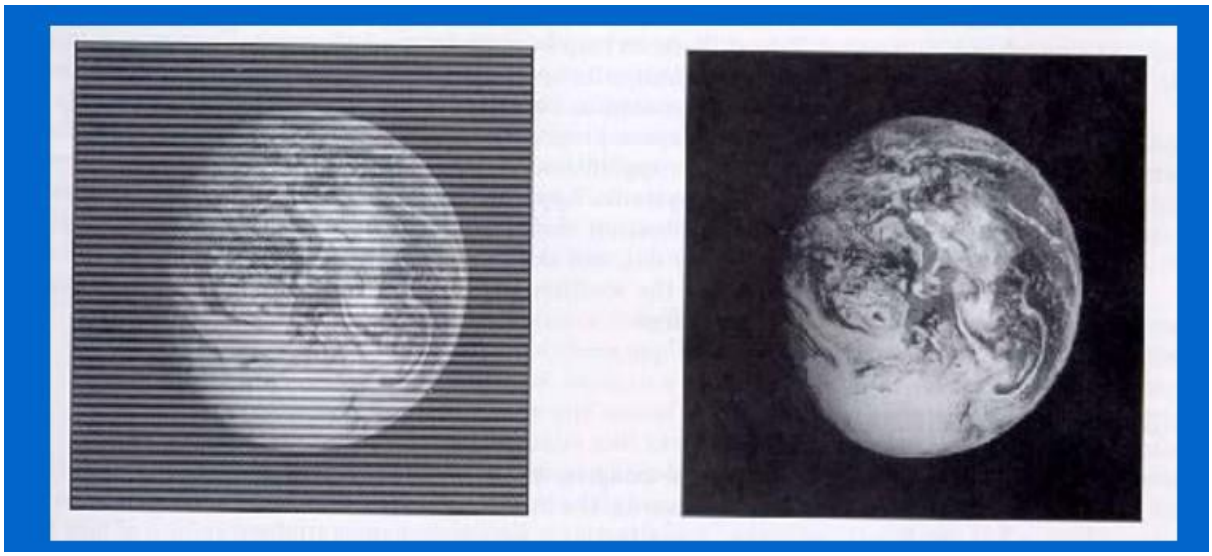
1) ImageProcessing



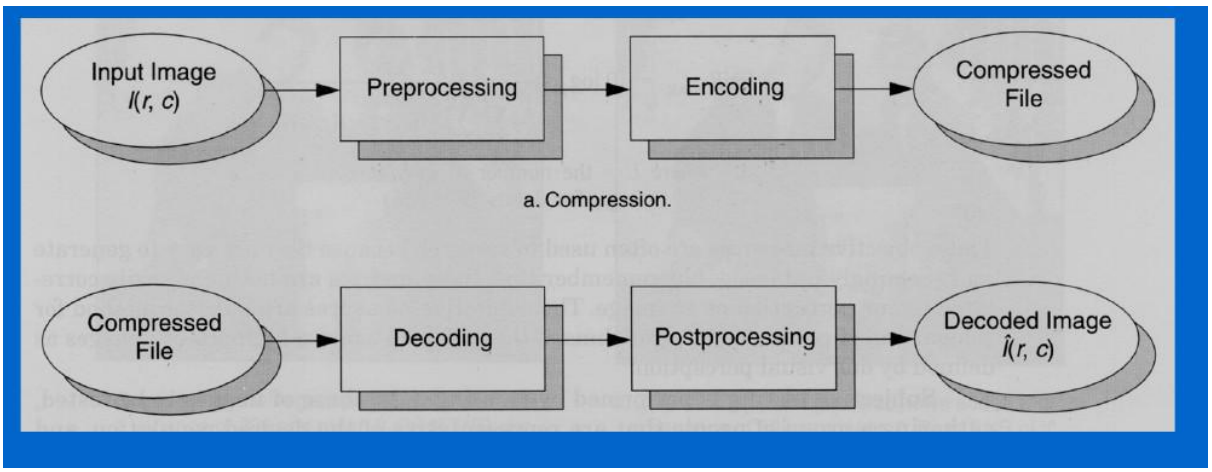
- **ImageEnhancement**



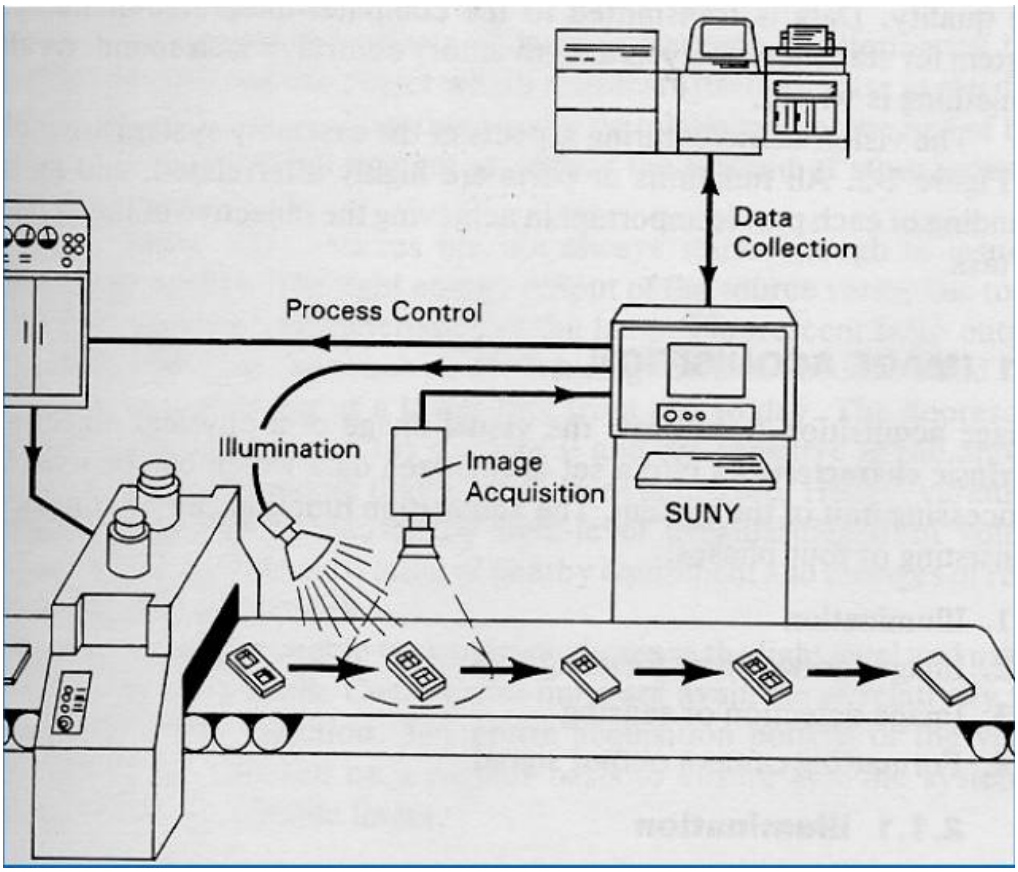
- **Image Restoration(e.g., correcting out-focus images)**



- **Image Compression**



An Industrial Computer Vision System



About OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real-time computer vision.Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license.
OpenCV supports the deep learning frameworks TensorFlow, Torch/PyTorch and Caffe.

In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

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.

Programming language

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience.

Since version 3.4, **OpenCV.js** is a JavaScript binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

What’s TkInter?

The Tkinter module (“Tk interface”) is the standard Python interface to the Tk GUI toolkit from Scriptics (formerly developed by Sun Labs).
Both Tk and Tkinter are available on most Unix platforms, as well as on Windows and Macintosh systems. Starting with the 8.0 release, Tk offers native look and feel on all platforms.

Tkinter consists of a number of modules. The Tk interface is provided by a binary extension module named _**tkinter**. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

The public interface is provided through a number of Python modules. The most important interface module is the **Tkinter** module itself. To use Tkinter, all you need to do is to import the **Tkinter** module:

import Tkinter

Or, more often:

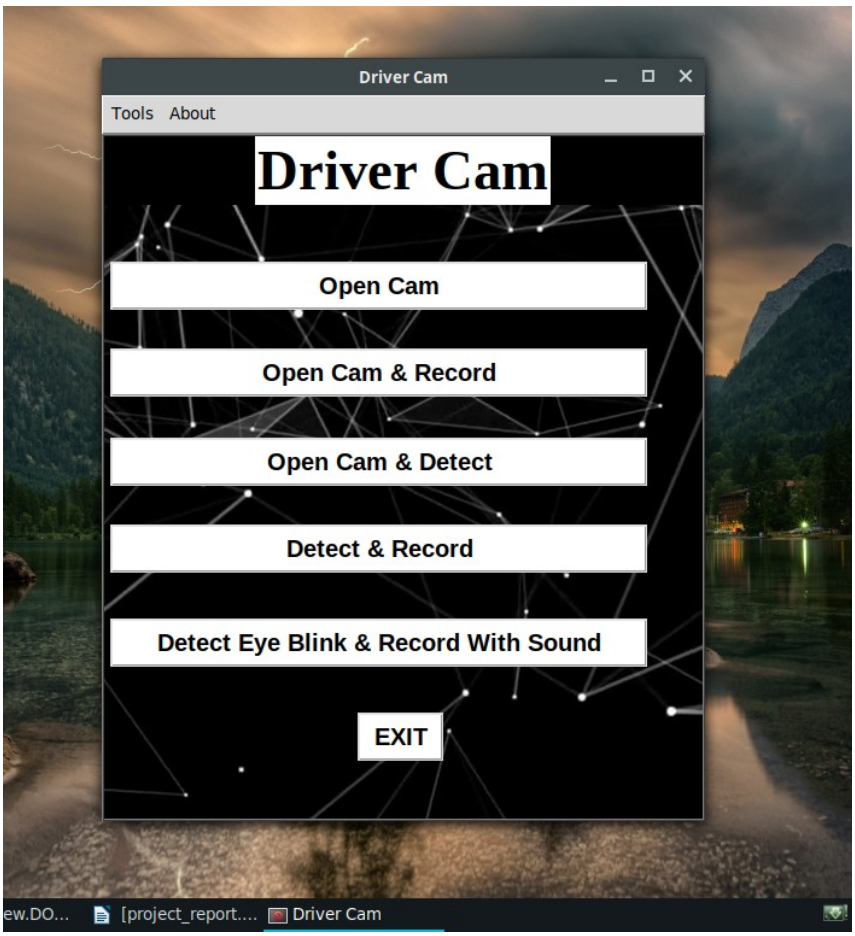
from Tkinter import *

The Tkinter module only exports widget classes and associated constants, so you can safely use the from-in form in most cases. If you prefer not to, but still want to save some typing, you can use import-as:

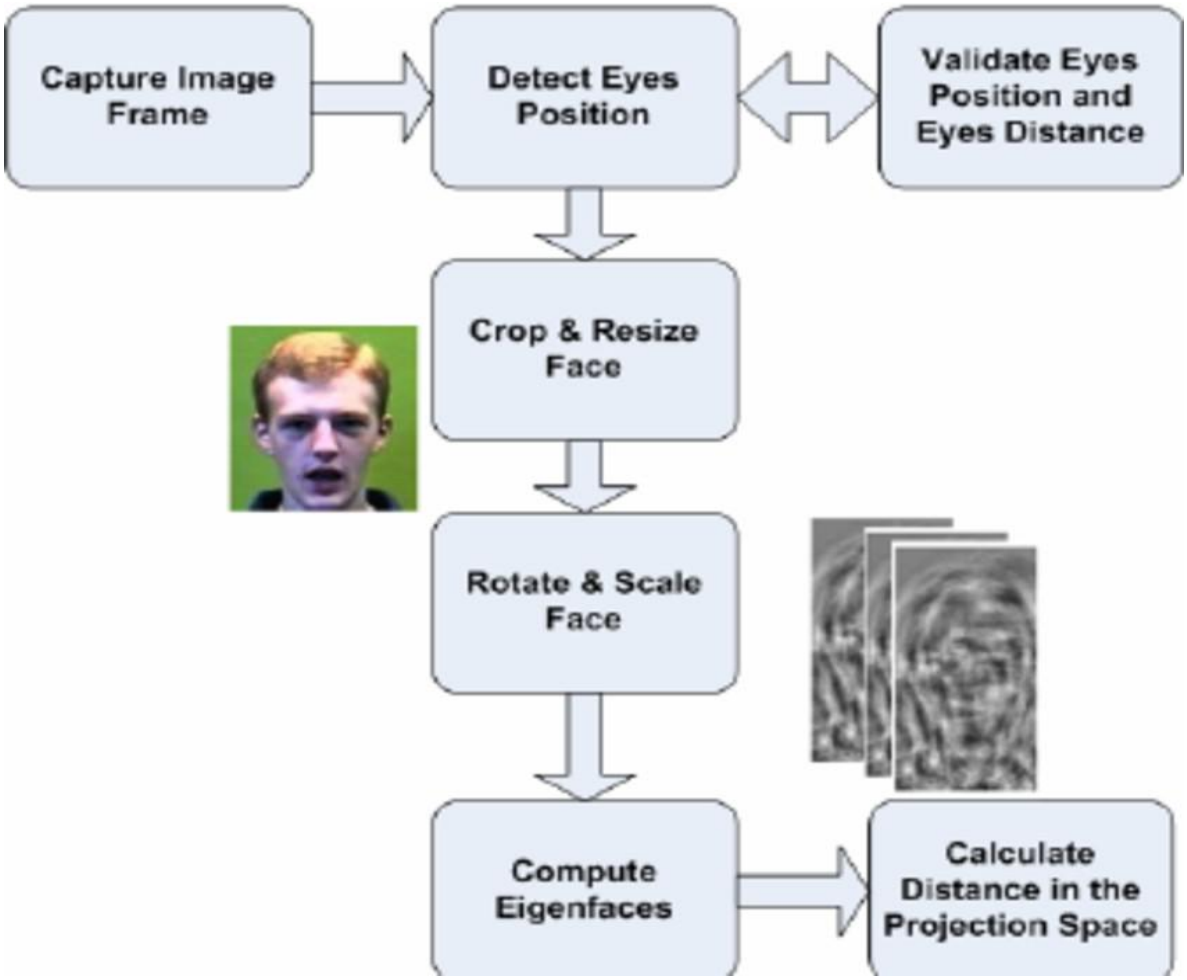
import Tkinter as Tk

Tkinter Gui:

This project uses tkinter for following gui implementation:



Data flow diagram illustrating the basic steps for face detection and recognition



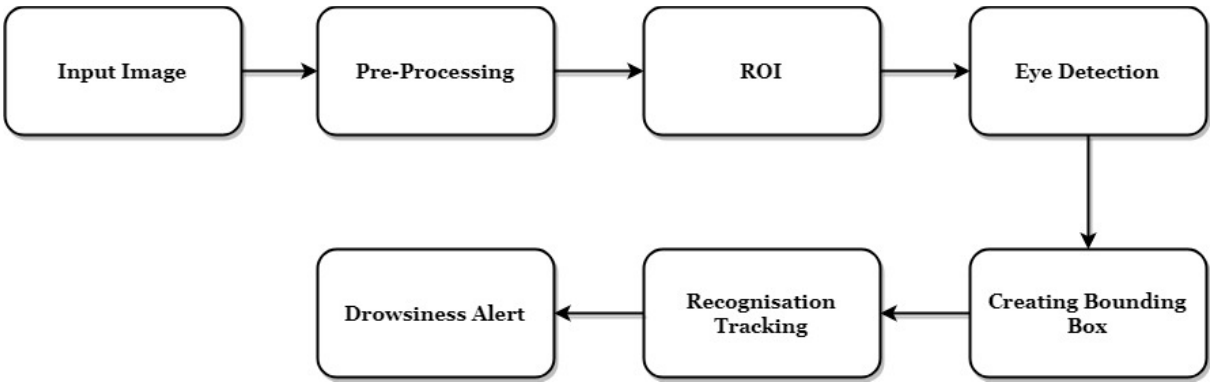
Facial Recognition Applications

- **Security:** Companies are training deep learning algorithms to recognize fraud detection, reduce the need for traditional passwords, and to improve the ability to distinguish between a human face and a photograph.
- **Healthcare:** Machine learning is being combined with computer vision to more accurately track patient medication consumption and support pain management procedures.
- **Marketing:** Fraught with ethical considerations, marketing is a burgeoning domain of facial recognition innovation, and it's one we can expect to see more of as facial recognition becomes ubiquitous. Criminal identification
- **Criminal Identification**
- **Payments**
- Image Search:Google recently introduced the ability to search for images by comparing them with others. By uploading an image or giving Google an image URL, it will show you where that image is used on the Web, and display similar images too.

Driver Drowsiness Detetction

There are several different algorithms and methods for eye tracking, and monitoring. Most of them in some way relate to features of the eye (typically reflections from the eye) within a video image of the driver. The original aim of this project was to use the retinal reflection as a means to finding the eyes on the face, and then using the absence of this reflection as a way of detecting when the eyes are closed. Applying this algorithm on consecutive video frames may aid in the calculation of eye closure period. Eye closure period for drowsy drivers are longer than normal blinking. It is also very little longer time could result in severe crash. So we will warn the driver as soon as closed eye is detected.

Block Diagram



Building the drowsiness detector with OpenCV

We'll need the SciPy package so we can compute the Euclidean distance between facial landmarks points in the eye aspect ratio calculation (not strictly a requirement, but you should have SciPy installed if you intend on doing any work in the computer vision, image processing, or machine learning space).
We'll also need the imutils_package, my series of computer vision and image processing functions to make working with OpenCV easier.

If you don't already have imutils installed on your system.

$ pip install --upgrade imutils

We'll also import the Thread class so we can play our alarm in a separate thread from the main thread to ensure our script doesn't pause execution while the alarm sounds.

In order to actually play our WAV/MP3 alarm, we need the playsound library, a pure Python, cross-platform implementation for playing simple sounds.

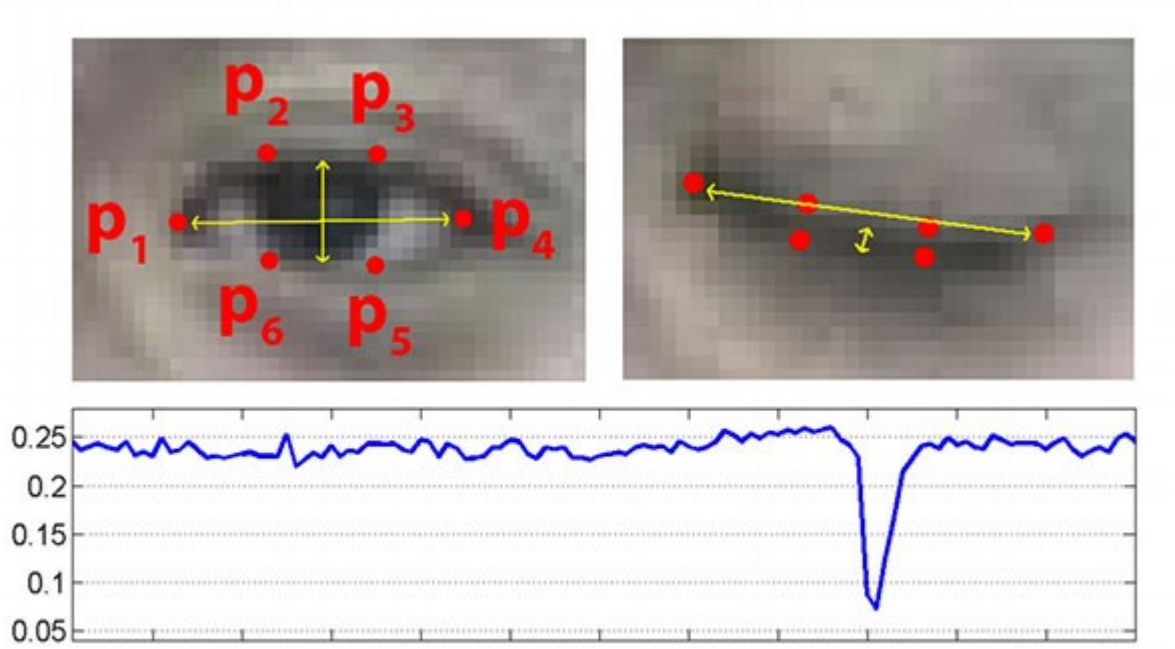
The playsound library is conveniently installable via pip

$ pip install playsound

We also need to define the eye_aspect_ratio function which is used to compute the ratio of distances between the vertical eye landmarks and the distances between the horizontal eye landmarks:

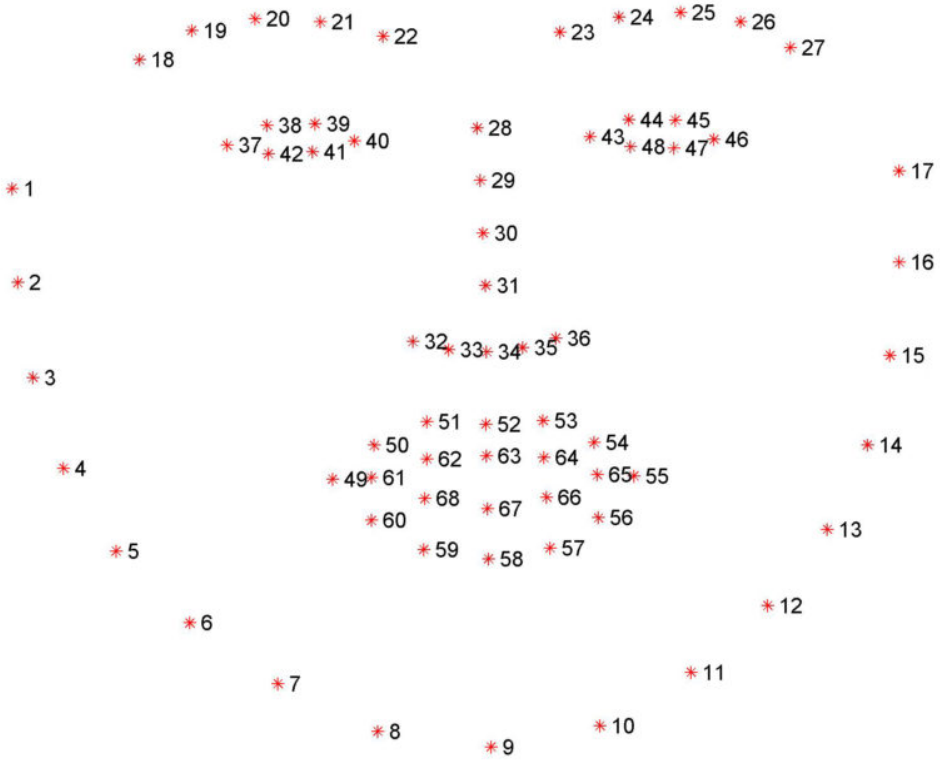
The return value of the eye aspect ratio will be approximately constant when the eye is open. The value will then rapid decrease towards zero during a blink.

If the eye is closed, the eye aspect ratio will again remain approximately constant, but will be *much smaller* than the ratio when the eye is open.

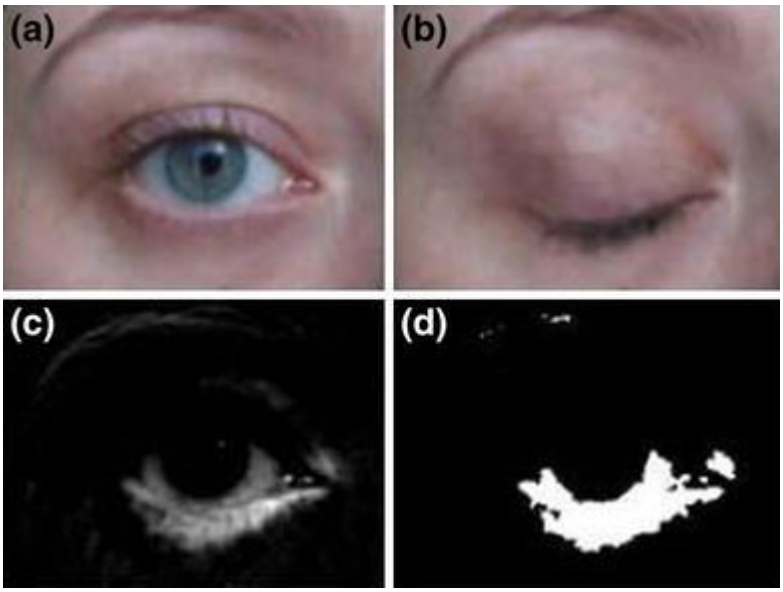


The dlib library ships with a Histogram of Oriented Gradients-based face detector along with a facial landmark predictor- we instantiate both of these in the following code block:

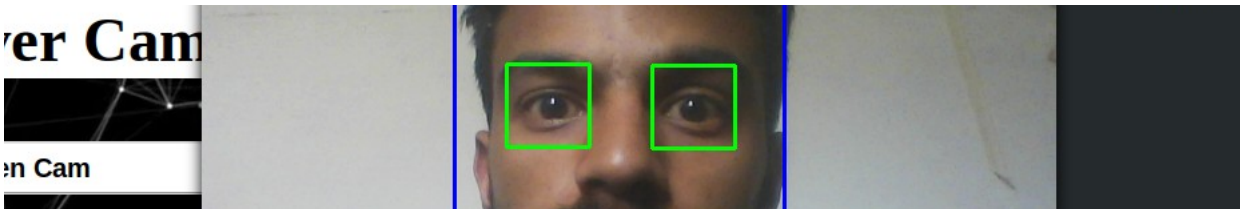
The facial landmarks produced by dlib are an indexable list:



OpenCV eye detection



Detecting Eyes using OpenCV python output:



IMPLEMENTATION

The following code is used to implement the algorithm..

```
import numpy
from pygame import mixer
import time
import cv2
from tkinter import *
import tkinter.messagebox
root=Tk()
root.geometry('500x570')
frame = Frame(root, relief=RIDGE, borderwidth=2)
frame.pack(fill=BOTH,expand=1)
root.title('Driver Cam')
frame.config(background='light blue')
label = Label(frame, text="Driver Cam",bg='light blue',font=('Times 35 bold'))
label.pack(side=TOP)
filename = PhotoImage(file="E:\python\Driver Cam\demo.png")
background_label = Label(frame,image=filename)
background_label.pack(side=TOP)

def hel():
help(cv2)
def Contri():
tkinter.messagebox.showinfo("Contributors","\n1.Mayur Kadam\n2. Abhishek Ezhava \n3. Rajendra Patil \n")

def anotherWin():
tkinter.messagebox.showinfo("About",'Driver Cam version v1.0\n Made Using\n-OpenCV\n-Numpy\n-Tkinter\n In Python 3')

menu = Menu(root)
root.config(menu=menu)
subm1 = Menu(menu)
menu.add_cascade(label="Tools",menu=subm1)
subm1.add_command(label="Open CV Docs",command=hel)
subm2 = Menu(menu)
menu.add_cascade(label="About",menu=subm2)
```

```
subm2.add_command(label="Driver Cam",command=anotherWin)
subm2.add_command(label="Contributors",command=Contri)

def exitt():
exit()

def web():
capture =cv2.VideoCapture(0)
while True:
ret,frame=capture.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
cv2.imshow('frame',frame)
if cv2.waitKey(1) & 0xFF ==ord('q'):
break
capture.release()
cv2.destroyAllWindows()

def webrec():
capture =cv2.VideoCapture(0)
fourcc=cv2.VideoWriter_fourcc(*'XVID')
op=cv2.VideoWriter('Sample1.avi',fourcc,11.0,(640,480))
while True:
ret,frame=capture.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
cv2.imshow('frame',frame)
op.write(frame)
if cv2.waitKey(1) & 0xFF ==ord('q'):
break
op.release()
capture.release()
cv2.destroyAllWindows()

def webdet():
capture =cv2.VideoCapture(0)
face_cascade = cv2.CascadeClassifier('lbpcascade_frontalface.xml')
eye_glass = cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')

while True:
ret, frame = capture.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray)

for (x,y,w,h) in faces:
font = cv2.FONT_HERSHEY_COMPLEX
cv2.putText(frame,'Face',(x+w,y+h),font,1,(250,250,250),2,cv2.LINE_AA)
cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
roi_gray = gray[y:y+h, x:x+w]
```

```
roi_color = frame[y:y+h, x:x+w]

eye_g = eye_glass.detectMultiScale(roi_gray)
for (ex,ey,ew,eh) in eye_g:
cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

cv2.imshow('frame',frame)
if cv2.waitKey(1) & 0xff == ord('q'):
break
capture.release()
cv2.destroyAllWindows()
def webdetRec():
capture =cv2.VideoCapture(0)
face_cascade = cv2.CascadeClassifier('lbpcascade_frontalface.xml')
eye_glass = cv2.CascadeClassifier('haarcascade_eye_tree_eyeglasses.xml')
fourcc=cv2.VideoWriter_fourcc(*'XVID')
op=cv2.VideoWriter('Sample2.avi',fourcc,9.0,(640,480))
while True:
ret, frame = capture.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray)

for (x,y,w,h) in faces:
font = cv2.FONT_HERSHEY_COMPLEX
cv2.putText(frame,'Face',(x+w,y+h),font,1,(250,250,250),2,cv2.LINE_AA)
cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
roi_gray = gray[y:y+h, x:x+w]
roi_color = frame[y:y+h, x:x+w]

eye_g = eye_glass.detectMultiScale(roi_gray)
for (ex,ey,ew,eh) in eye_g:
cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
op.write(frame)
cv2.imshow('frame',frame)
if cv2.waitKey(1) & 0xff == ord('q'):
break
op.release()
capture.release()
cv2.destroyAllWindows()

def alert():
mixer.init()
alert=mixer.Sound('beep-07.wav')
alert.play()
time.sleep(0.1)
alert.play()
```

```
def blink():
capture =cv2.VideoCapture(0)
face_cascade = cv2.CascadeClassifier('lbpcascade_frontalface.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
blink_cascade = cv2.CascadeClassifier('CustomBlinkCascade.xml')

while True:
ret, frame = capture.read()
gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray)

for (x,y,w,h) in faces:
font = cv2.FONT_HERSHEY_COMPLEX
cv2.putText(frame,'Face',(x+w,y+h),font,1,(250,250,250),2,cv2.LINE_AA)
cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
roi_gray = gray[y:y+h, x:x+w]
roi_color = frame[y:y+h, x:x+w]

eyes = eye_cascade.detectMultiScale(roi_gray)
for(ex,ey,ew,eh) in eyes:
cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

blink = blink_cascade.detectMultiScale(roi_gray)
for(eyx,eyy,eyw,eyh) in blink:
cv2.rectangle(roi_color,(eyx,eyy),(eyx+eyw,eyy+eyh),(255,255,0),2)
alert()
cv2.imshow('frame',frame)
if cv2.waitKey(1) & 0xFF ==ord('q'):
break

capture.release()
cv2.destroyAllWindows()

but1=Button(frame,padx=5,pady=5,width=39,bg='white',fg='black',relief=GROOVE,comma
nd=web,text='Open Cam',font=('helvetica 15 bold'))
but1.place(x=5,y=104)

but2=Button(frame,padx=5,pady=5,width=39,bg='white',fg='black',relief=GROOVE,comma
nd=webrec,text='Open Cam & Record',font=('helvetica 15 bold'))
but2.place(x=5,y=176)

but3=Button(frame,padx=5,pady=5,width=39,bg='white',fg='black',relief=GROOVE,comma
nd=webdet,text='Open Cam & Detect',font=('helvetica 15 bold'))
but3.place(x=5,y=250)

but4=Button(frame,padx=5,pady=5,width=39,bg='white',fg='black',relief=GROOVE,comma
nd=webdetRec,text='Detect & Record',font=('helvetica 15 bold'))
but4.place(x=5,y=322)

but5=Button(frame,padx=5,pady=5,width=39,bg='white',fg='black',relief=GROOVE,comma
nd=blink,text='Detect Eye Blink & Record With Sound',font=('helvetica 15 bold'))
but5.place(x=5,y=400)
```

```
but5=Button(frame,padx=5,pady=5,width=5,bg='white',fg='black',relief=GROOVE,text='EX
IT',command=exitt,font=('helvetica 15 bold'))
but5.place(x=210,y=478)

root.mainloop()
```

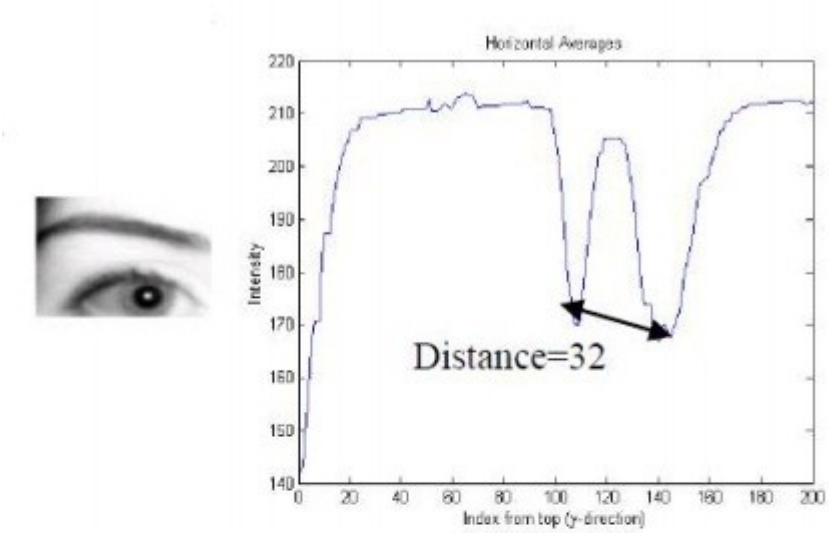
Result

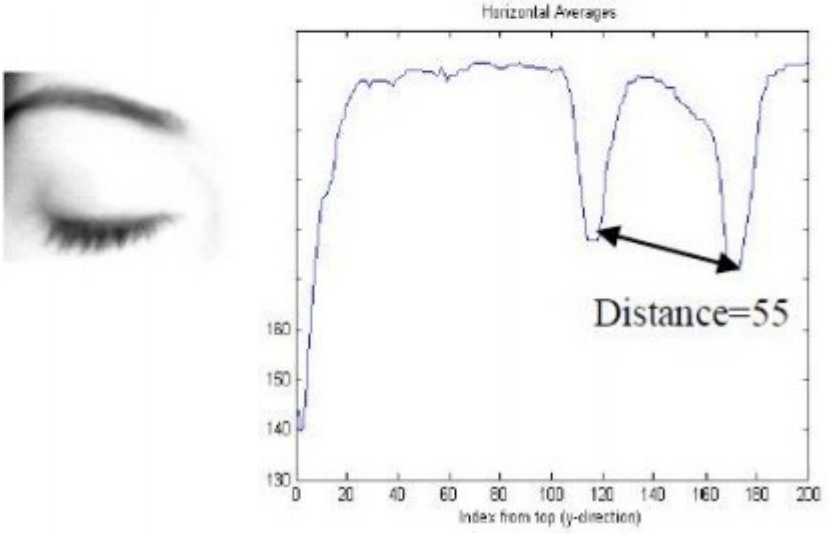
A. Drowsiness Detection Function

The state of the eyes (whether it is open or closed) is determined by distance between the first two intensity changes found in the above step. When the eyes are closed, the distance between the y – coordinates of the intensity changes is larger if compared to when the eyes are open. Fig. 3. Comparison between opened and closed eyes The limitation to this is if the driver moves their face closer to or further from the camera. If this occurs, the distances will vary, since the number of pixels the face takes up varies, as seen below. Because of this limitation, the system developed assumes that the driver's face stays almost the same distance from the camera at all times.

B. Judging Drowsiness

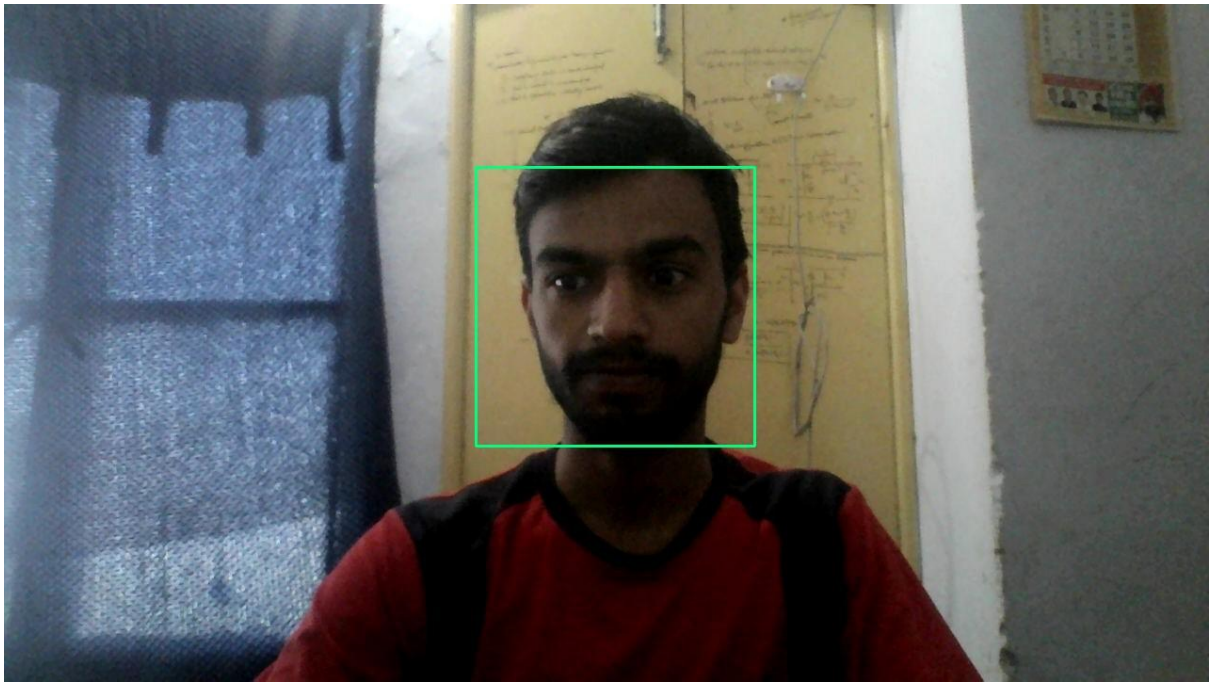
When there are 5 consecutive frames find the eye closed, then the alarm is activated, and a driver is alerted to wake up. Consecutive number of closed frames is needed to avoid including instances of eye closure due to blinking.


Horizontal Averages
Intensity
Distance=32
Index from top (y-direction)

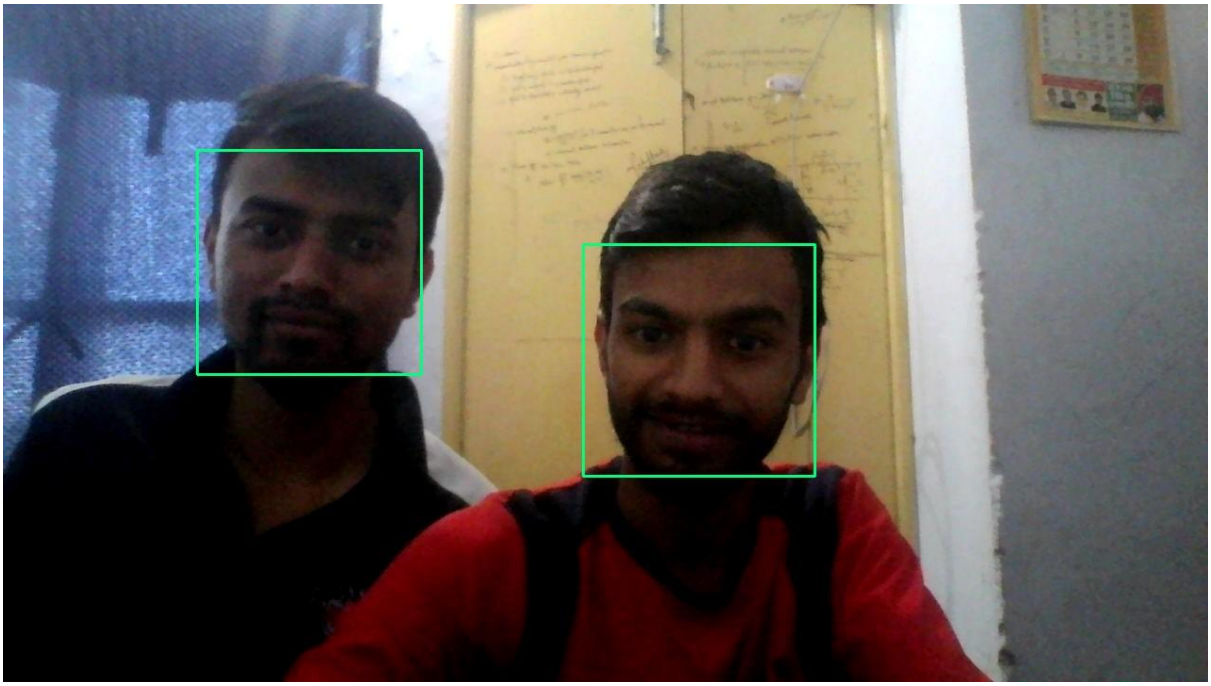

Horizontal Averages
Distance=55
Index from top (y-direction)

Face Recognition outputs

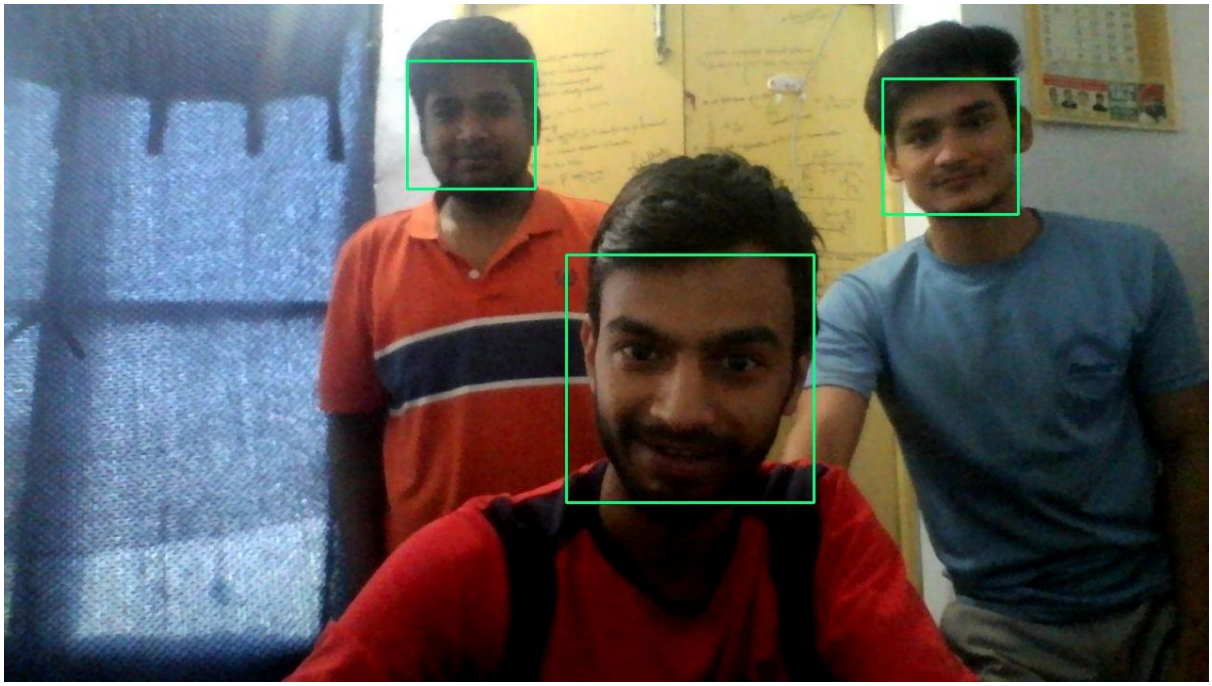
Single person face detection



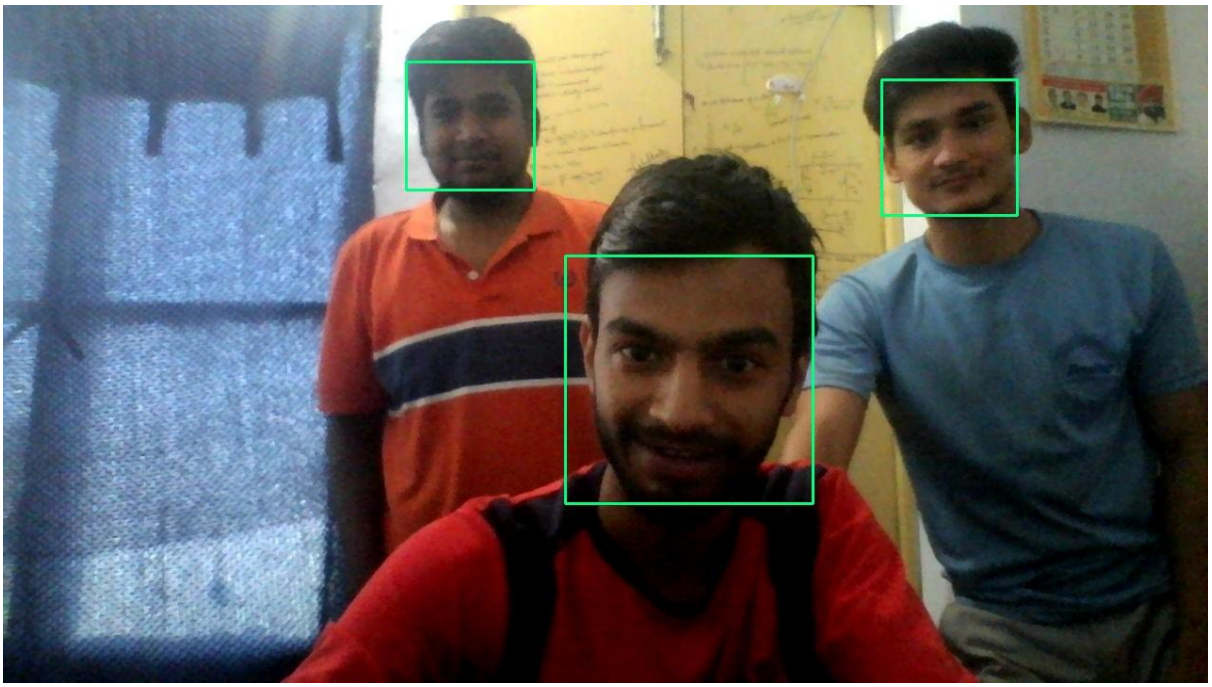
Two person face detection



Outputs with multiple faces

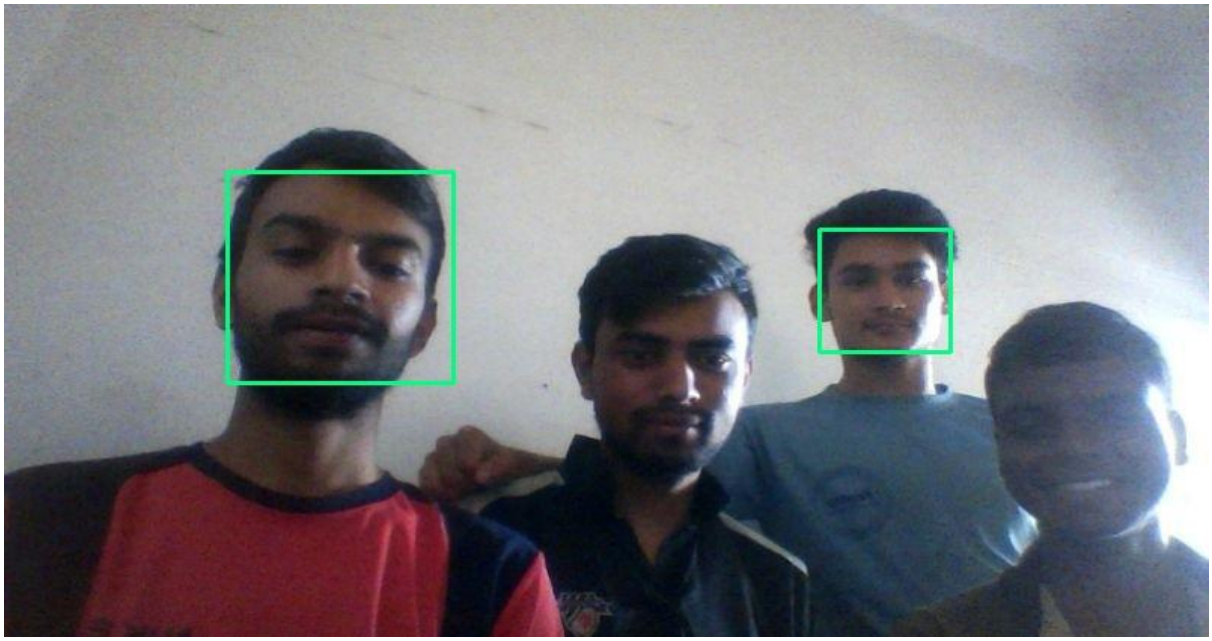


Outputs with multiple faces



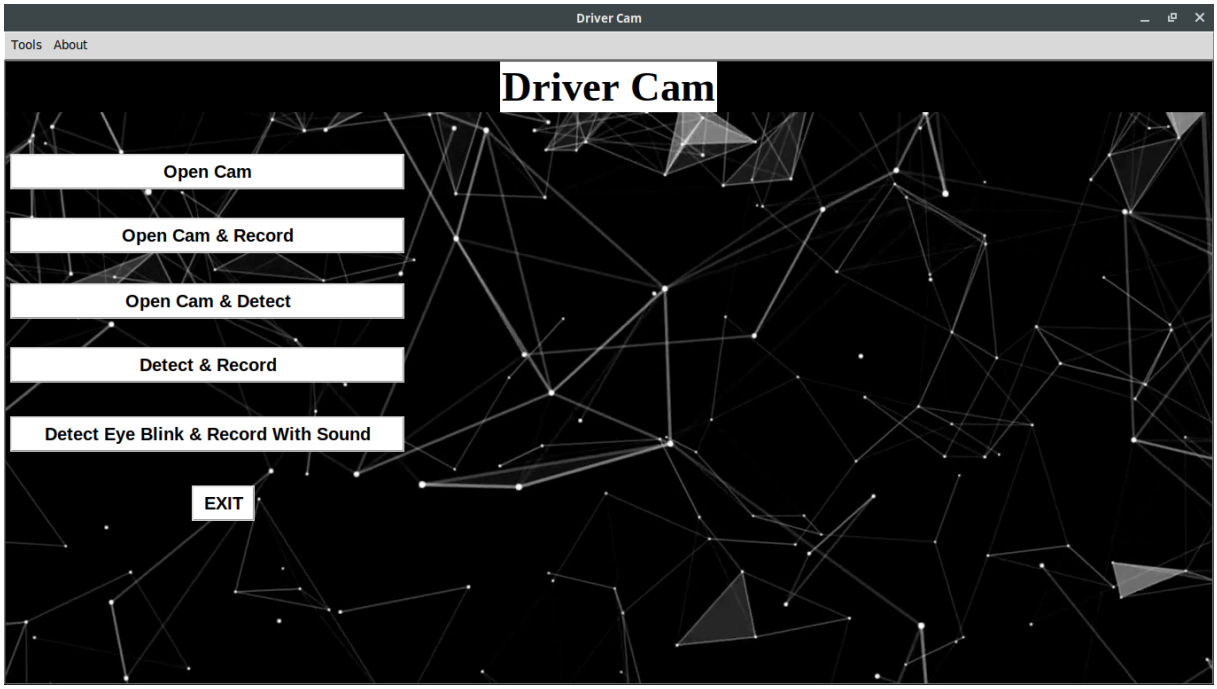
Limitations

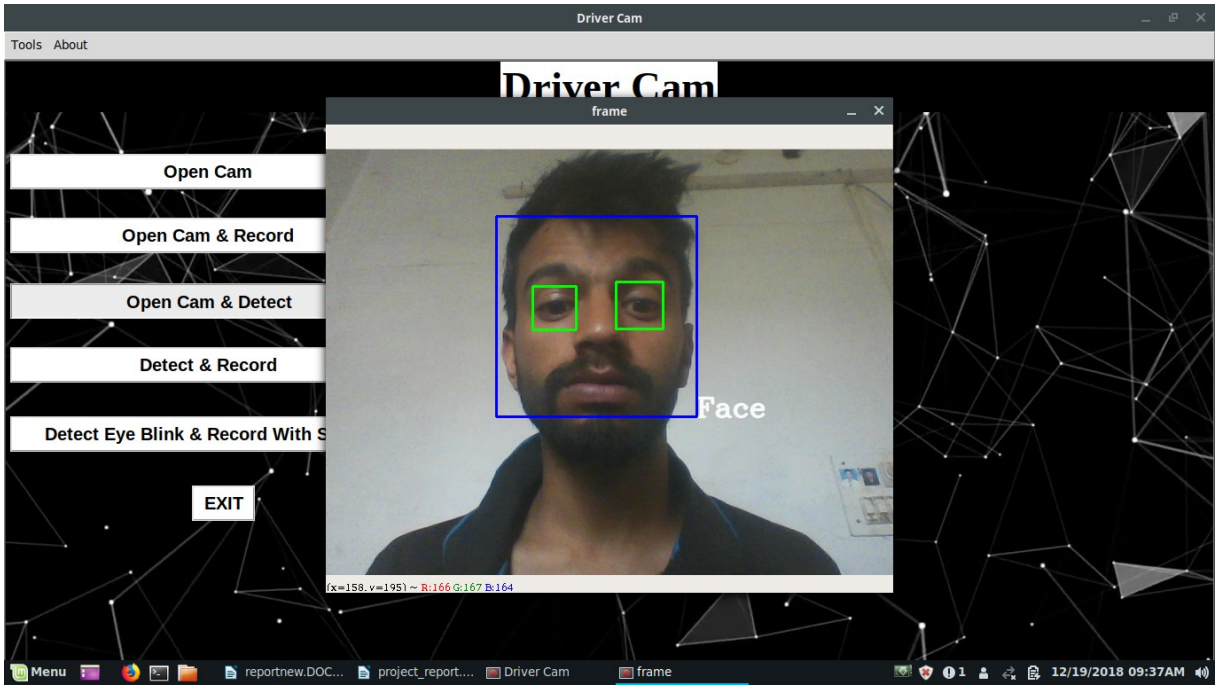
- **Limitation due to bad lighting**



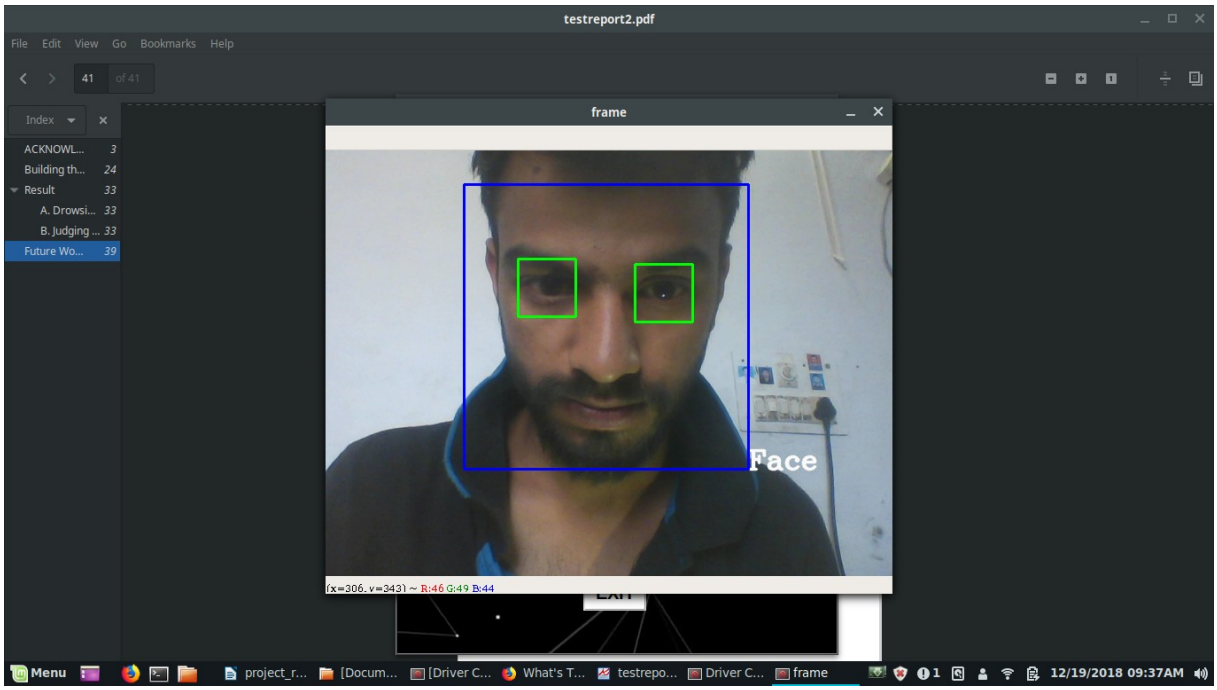
- **Due to bad orientation**

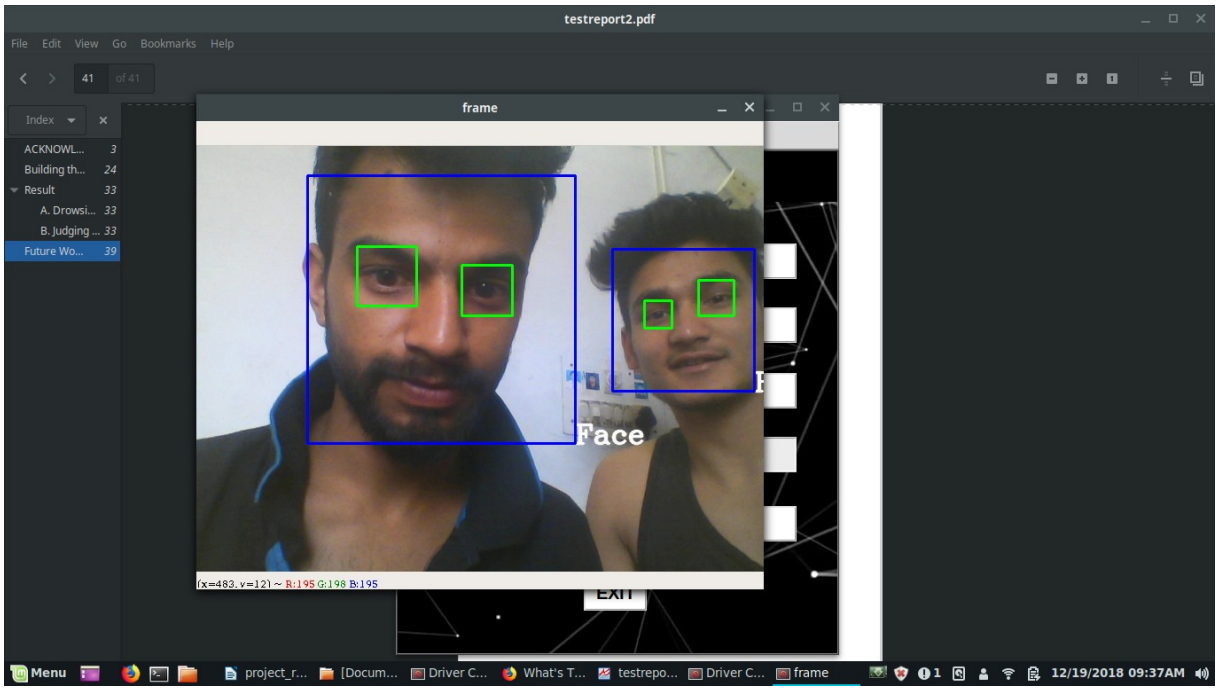
Drowsiness Detection System
Face and eye Recognition





Some more outputs:





Future Work:

Fatigue is often ranked as a major factor in causing road crashes although its contribution to individual cases is hard to measure and is often not reported as a cause of crash. Driver fatigue is particularly dangerous because one of the symptoms is decreased ability to judge our own level of tiredness. Estimates suggest that fatigue is a factor in up to 30% of fatal crashes and 15%of serious injury crashes. Fatigue also contributes to approximately 25% of insurance losses in the heavy vehicle industry. The yawing and nodding of head can be implemented for detecting drowsiness for future work.

Conclusion

An accurate and eficient object detection system has been developed which achieves comparable metrics with the existing state-of-the-art system.This project uses recent techniques in the field of computer vision and deep learning. Custom dataset was created using labeling and the evaluation was consistent. This can be used in real-time applications which require object detection for pre-processing in their pipeline.

An important scope would be to train the system on a video sequence for usage in tracking applications. Addition of a temporally consistent network would enable smooth detection and more optimal than per-frame detection.

References

1) Ross Girshick, Je Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

2) Ross Girshick. Fast R-CNN. In International Conference on Computer Vision (ICCV), 2015.

3) Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in Neural Information Processing Systems (NIPS), 2015.

4) Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Uni ed, real-time object detection. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

5) Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In ECCV, 2016.

6) Paul Viola and Michael j. Jones,"Rapid Object Detection using a Boosted Cascade of Simple Features,"International Journal of Computer Vision 57(2), pp 137–154, 2001.