CHAPTER I
INTRODUCTION

1.1 Problem description:

The increasing number of traffic accidents due to a driver's diminished vigilance level is a serious problem for the society. Driver's abilities of vehicle control, natural reflex, recognition and perception decline due to drowsiness and fatigue, reducing the driver's vigilance level. These pose serious danger to their own lives as well as lives of other people. According to the U.S. National Highway Traffic Safety Administration (NHTSA), drowsiness and falling asleep while driving are responsible for at least 100,000 automobile crashes annually. An annual average of roughly 40,000 nonfatal injuries and 1,550 fatalities result from these crashes. These figures only present the casualties happening during midnight to early morning, and underestimate the true level of the involvement of drowsiness because they do not include crashes during daytime hours. Vehicles with driver intelligence system that can detect drowsiness of the driver and send alarm may avert fatal accidents.

1.2 Objective:

The objective is to calculate the eye aspect ratio and write program in openCV python to detect the drowsiness of the eye. In case of drowsiness the eye aspect ratio gets below the threshold level. At this level, the alarm gets activated and alerts the driver.

Major Objective of the project is

- To simplify image processing process
- To use openCV for eye gaze detection
- To avoid accidents due to drowsiness.

1.3 Block Diagram:

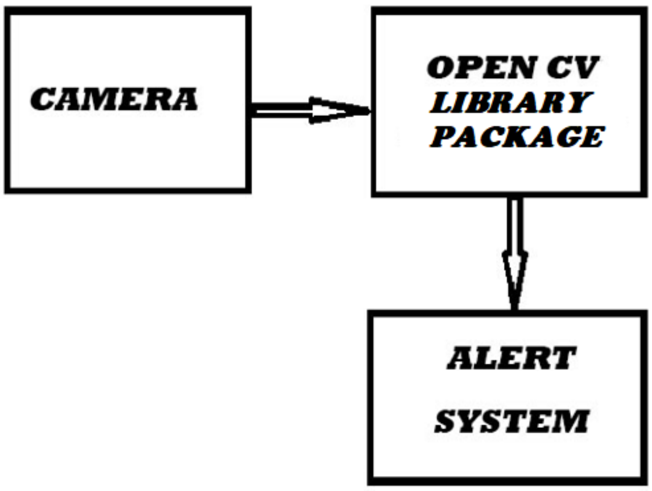


Figure 1.1: Shows the complete block diagram of the project

We use a camera to monitor the driver in the vehicle. Then we use those video signals from the camera to detect the eye of the driver. For eye detection and continuous tracking of the eye we use the openCV and various other packages in python. The drowsiness of the driver is detected by calculating EAR (Eye Aspect Ratio). From this project we can find whether the driver is in a state of drowsiness or not. Each person's eye is different from one and another, so we use EAR for finding the open or close state of the driver's eye. Depending on how many seconds the eye is closed we are detecting the drowsiness of the driver. After detecting the drowsiness we have to alert the driver to avoid accidents. For that an alert system is used which play an alarm sound after detecting the drowsiness of the driver.

1.4 Tools Used:

- OpenCV
- Dlib
- Numpy
- Scipy
- Pygame
- Imutils

1.5 Cost Estimation:

ITEM	**QUANTITY**	**COST(in Rs)**
Camera	1	2000
Buzzer	1	100
Total		**2100**

CHAPTER 2

TOOLS

2.1 OpenCV:

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general purpose programming language started by **Guido van Rossum** that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of **Numpy**, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

The OpenCV libraries, distributed by us, on the Microsoft Windows operating system are in a Dynamic Linked Libraries (DLL). These have the advantage that all the content of the library are loaded only at runtime, on demand, and that countless programs may use the same library file. This means that if you have ten applications using the OpenCV library, no need to have around a version for each one of them. Of course you need to have the dll of the OpenCV on all systems where you want to run your application.

To build an application with OpenCV you need to do two things:

• Tell to the compiler how the OpenCV library looks. You do this by showing it the header files.

• Tell to the linker from where to get the functions or data structures of OpenCV, when they are needed.

If you use the lib system you must set the path where the library files are and specify in which one of them to look. During the build the linker will look into these libraries and add the definitions and implementation of all used functions and data structures to the executable file. If you use the DLL system you must again specify all this, however now for a different reason. This is a Microsoft OS specific stuff. It seems that the linker needs to know that where in the DLL to search for the data structure or function at the runtime. This information is stored inside lib files. Nevertheless, they aren't static libraries. They are so called import libraries. This is why when you make some DLLs in Windows you will also end up with some lib extension libraries. The good part is that at runtime only the DLL is required.

2.2 Dlib:

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

Compiling Python modules such as NumPy, SciPy etc. is a tedious task. Anaconda is a Compiling Python modules such as NumPy, SciPy etc. is a tedious task. Anaconda is a great Python distribution which comes with a lot of pre-compiled Python packages. So we will use Anaconda as our Python distribution.

An alternative to Anaconda is that you install official Python library and use Christoph Gohlke's awesome repository to install pre-compiled Python modules.

This tutorial is based on using Anaconda so we may not be able to help if you choose to use Gohlke's precompiled binaries or if you compile Python libraries from source.

If you intend to use Dlib only in C++ projects, you can skip Python installation part.

Now let's go through the steps to install Dlib. Follow our previous post Install OpenCV3 on Windows to complete Step 1, 2 and 3.

Step 1: Install Visual Studio 2015

Step 2: Install CMake v3.8.2

Step 3: Install Anaconda 3

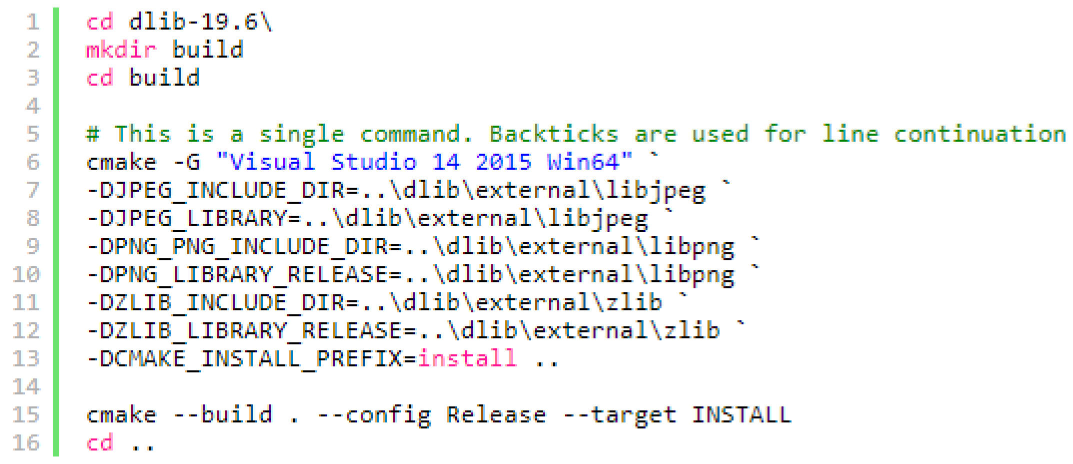
Step 4: Download Dlib

Download Dlib v19.6 from http://dlib.net/files/dlib-19.6.zip

Step 5: Build Dlib library

Extract this compressed file. Open Windows PowerShell or Command Prompt and move to the directory where you have extracted this file.

If you are running these commands on Command Prompt replace ` (backtick) with ^ (caret).



```
cd dlib-19.6\
mkdir build
cd build

# This is a single command. Backticks are used for line continuation
cmake -G "Visual Studio 14 2015 Win64" `
-DJPEG_INCLUDE_DIR=..\dlib\external\libjpeg `
-DJPEG_LIBRARY=..\dlib\external\libjpeg `
-DPNG_PNG_INCLUDE_DIR=..\dlib\external\libpng `
-DPNG_LIBRARY_RELEASE=..\dlib\external\libpng `
-DZLIB_INCLUDE_DIR=..\dlib\external\zlib `
-DZLIB_LIBRARY_RELEASE=..\dlib\external\zlib `
-DCMAKE_INSTALL_PREFIX=install ..

cmake --build . --config Release --target INSTALL
cd ..
```

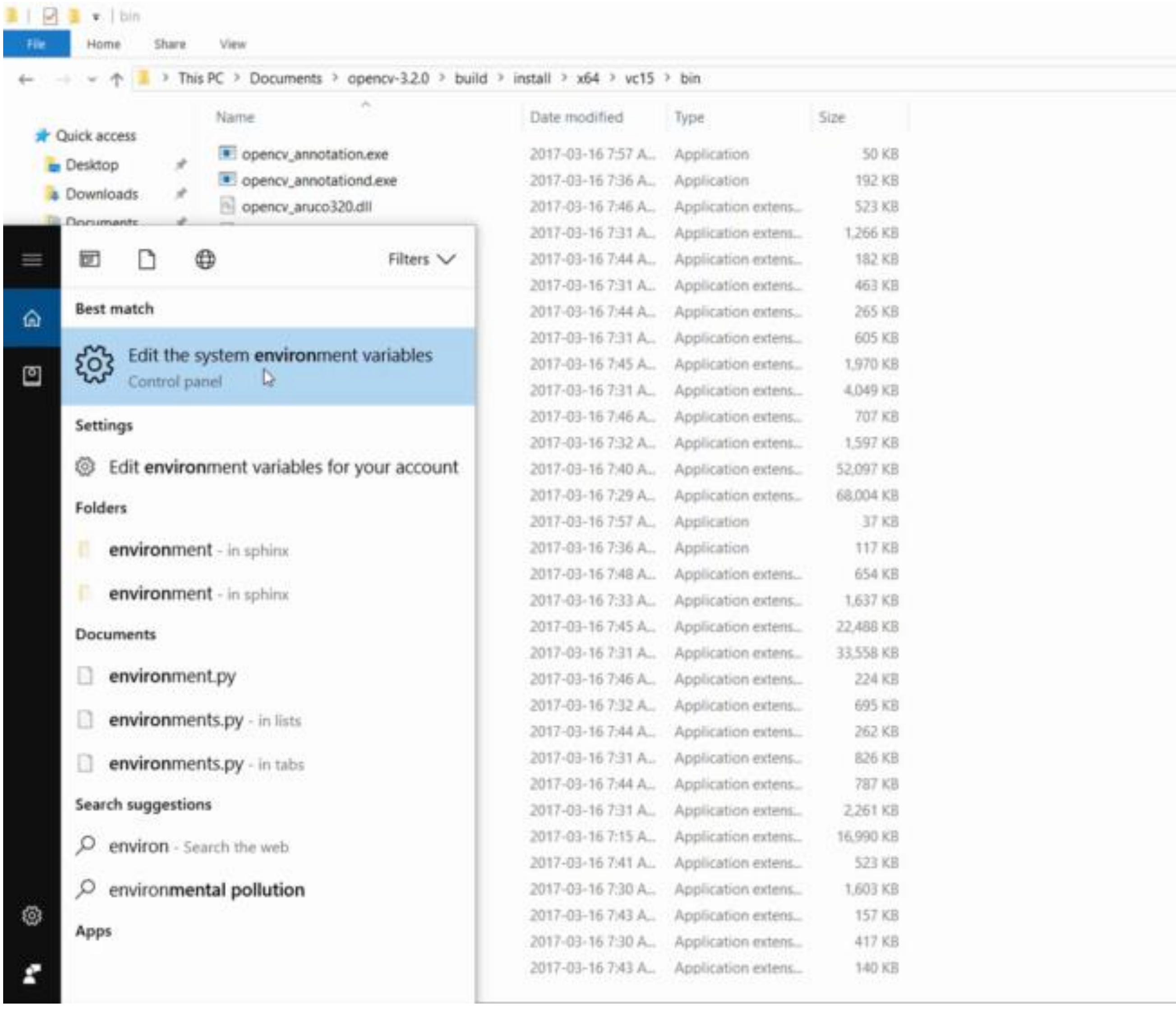
Fig.2.1. Dlib installation

Dlib will be installed within dlib-19.6\build\install directory. We will use CMake to build Dlib examples but you can use Visual Studio too. This directory (dlib-19.6\build\install) contains include and library folders which you can specify in Visual Studio to build projects using Dlib.

Step 6: Update user environment variable – dlib_DIR

This environment variable is needed for CMake to find out where Dlib is installed. CMake looks for a file named dlibConfig.cmake within directory dlib_DIR to find Dlib’s include and library directories.

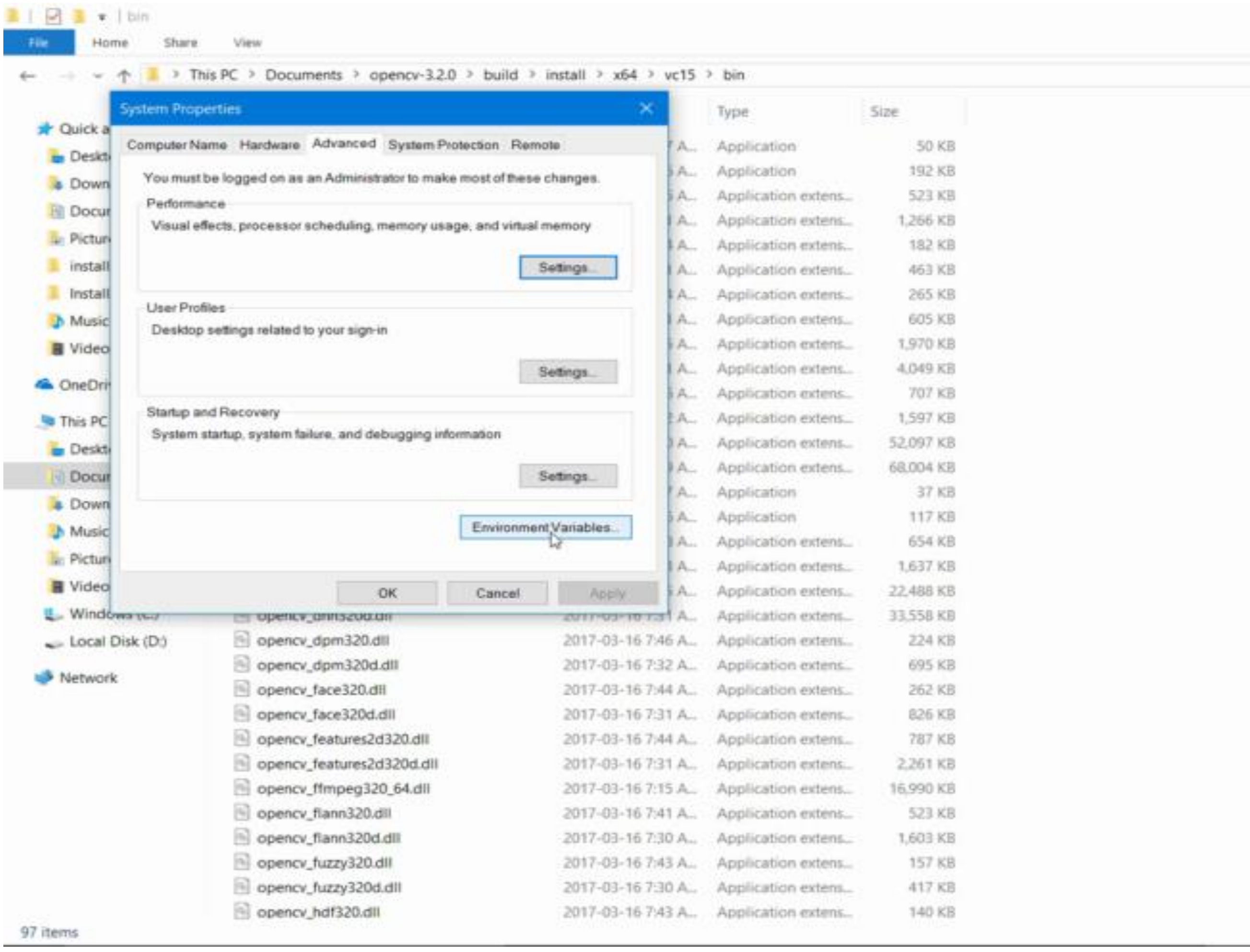
1. Press Windows Super key, search for “environment variables”.



Name	Date modified	Type	Size
opencv_annotation.exe	2017-03-16 7:57 A...	Application	50 KB
opencv_annotationd.exe	2017-03-16 7:36 A...	Application	192 KB
opencv_aruco320.dll	2017-03-16 7:46 A...	Application extens...	523 KB
	2017-03-16 7:31 A...	Application extens...	1,266 KB
	2017-03-16 7:44 A...	Application extens...	182 KB
	2017-03-16 7:31 A...	Application extens...	463 KB
	2017-03-16 7:44 A...	Application extens...	265 KB
	2017-03-16 7:31 A...	Application extens...	605 KB
	2017-03-16 7:45 A...	Application extens...	1,970 KB
	2017-03-16 7:31 A...	Application extens...	4,049 KB
	2017-03-16 7:46 A...	Application extens...	707 KB
	2017-03-16 7:32 A...	Application extens...	1,597 KB
	2017-03-16 7:40 A...	Application extens...	52,097 KB
	2017-03-16 7:29 A...	Application extens...	68,004 KB
	2017-03-16 7:57 A...	Application	37 KB
	2017-03-16 7:36 A...	Application	117 KB
	2017-03-16 7:48 A...	Application extens...	654 KB
	2017-03-16 7:33 A...	Application extens...	1,637 KB
	2017-03-16 7:45 A...	Application extens...	22,488 KB
	2017-03-16 7:31 A...	Application extens...	33,558 KB
	2017-03-16 7:46 A...	Application extens...	224 KB
	2017-03-16 7:32 A...	Application extens...	695 KB
	2017-03-16 7:44 A...	Application extens...	262 KB
	2017-03-16 7:31 A...	Application extens...	826 KB
	2017-03-16 7:44 A...	Application extens...	787 KB
	2017-03-16 7:31 A...	Application extens...	2,261 KB
	2017-03-16 7:15 A...	Application extens...	16,990 KB
	2017-03-16 7:41 A...	Application extens...	523 KB
	2017-03-16 7:30 A...	Application extens...	1,603 KB
	2017-03-16 7:43 A...	Application extens...	157 KB
	2017-03-16 7:30 A...	Application extens...	417 KB
	2017-03-16 7:43 A...	Application extens...	140 KB

Fig.2.2.Setting up environment variables

2. Click Environment Variables in System Properties window.



System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

User Profiles
Desktop settings related to your sign-in
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK | Cancel | Apply

Name	Date modified	Type	Size
opencv_dpm320.dll	2017-03-16 7:46 A...	Application extens...	224 KB
opencv_dpm320d.dll	2017-03-16 7:32 A...	Application extens...	695 KB
opencv_face320.dll	2017-03-16 7:44 A...	Application extens...	262 KB
opencv_face320d.dll	2017-03-16 7:31 A...	Application extens...	826 KB
opencv_features2d320.dll	2017-03-16 7:44 A...	Application extens...	787 KB
opencv_features2d320d.dll	2017-03-16 7:31 A...	Application extens...	2,261 KB
opencv_ffmpeg320_64.dll	2017-03-16 7:15 A...	Application extens...	16,990 KB
opencv_flann320.dll	2017-03-16 7:41 A...	Application extens...	523 KB
opencv_flann320d.dll	2017-03-16 7:30 A...	Application extens...	1,603 KB
opencv_fuzzy320.dll	2017-03-16 7:43 A...	Application extens...	157 KB
opencv_fuzzy320d.dll	2017-03-16 7:30 A...	Application extens...	417 KB
opencv_hdf320.dll	2017-03-16 7:43 A...	Application extens...	140 KB

Fig 2.3.Environment variables

3. Click New in "User Variables" (in upper half of window).

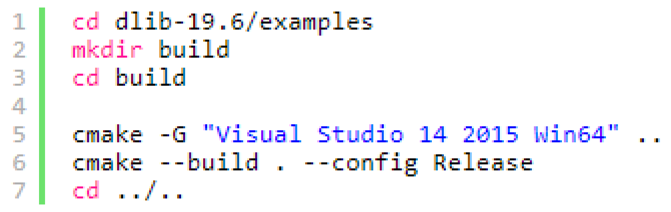
4. Under variable name write dlib_DIR and under variable value write full path to directory dlib-19.6\build\install\lib\cmake\dlib.On our machine, the path is: D:\programming\dlib-19.6\build\install\lib\cmake\dlib
   This directory contains file "dlibConfig.cmake". This is used by CMake to configure dlib_LIBS and dlib_INCLUDE_DIRS variables to generate project files. Before assigning the value to variable dlib_DIR make sure that this path has file dlibConfig.cmake.

5. Now click ok to save and close environment variables window.

Note: If you have an open Command Prompt/Power Shell window before these values were updated, you have to close and open a new Command Prompt/Power Shell window again.

Step 7: Build Dlib examples

We will use our CMakeLists.txt file instead of one which is shipped with Dlib. Download modified CMakeLists.txt file and put it in dlib-19.6\examples directory and replace the default one with this one. Then follow the steps given below:



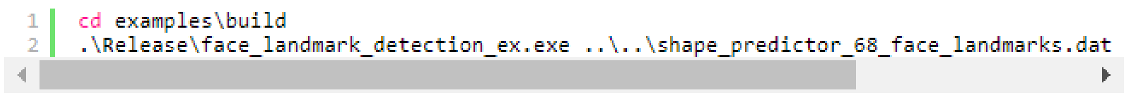
```
cd dlib-19.6/examples
mkdir build
cd build

cmake -G "Visual Studio 14 2015 Win64" ..
cmake --build . --config Release
cd ../..
```

Fig 2.4.Building Dlib

Once build is complete, it will generate executables for all examples in examples\build\Release folder.

Step 8: Test Dlib's C++ example

We will test Face Landmark Detection demo to check whether we have installed Dlib correctly. Download trained model of facial landmarks from Dlib's website. Extract this file (shape_predictor_68_face_landmarks.dat.bz2) to Dlib's root directory (dlib-19.6).



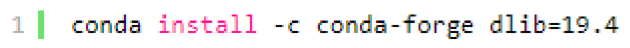
```
cd examples\build
.\Release\face_landmark_detection_ex.exe ..\..\shape_predictor_68_face_landmarks.dat
```

Fig 2.5.Testing Dlib's - C++

Step 9: Install Dlib's Python module (Only Anaconda 3)

Compiling Python bindings for Dlib from source is non-trivial. You have to compile Boost.Python from scratch and configure some environment variables (such as BOOST_ROOT and BOOST_LIBRARYDIR) before you can compile Python module of Dlib.

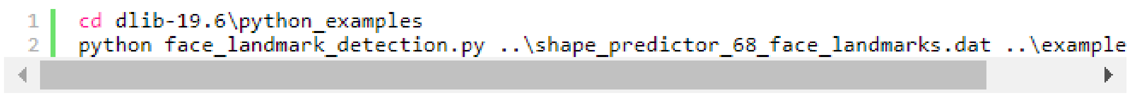
To save time and efforts it is suggested to use Anaconda 3. You can install a compiled binary of dlib v19.4 from Anaconda. At the time this article was updated Dlib's latest available version on Anaconda's conda-forge repository is 19.4. So we will install v19.4 instead of 19.6



```
conda install -c conda-forge dlib=19.4
```

Fig 2.6. Installation command

Step 10: Test Dlib's Python example



```
cd dlib-19.6\python_examples
python face_landmark_detection.py ..\shape_predictor_68_face_landmarks.dat ..\example
```

Fig 2.7. Testing Dlib's - Python

2.3 Numpy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

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

2.4 Scipy:

SciPy is an open source Python library used for scientific computing and technical computing.

SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.

SciPy is also a family of conferences for users and developers of these tools: SciPy (in the United States), EuroSciPy (in Europe) and SciPy.in (in India).

The SciPy library is currently distributed under the BSD license, and its development is sponsored and supported by an open community of developers. It is also supported by Numfocus which is a community foundation for supporting reproducible and accessible science.

The SciPy package of key algorithms and functions core to Python's scientific computing capabilities. Available sub-packages includes:

constants: physical constants and conversion factors (since version 0.7.0[5])

cluster: hierarchical clustering, vector quantization, K-means

FFT pack: Discrete Fourier Transform algorithms

integrate: numerical integration routines

interpolate: interpolation tools

io: data input and output

lib: Python wrappers to external libraries

linalg: linear algebra routines

misc: miscellaneous utilities (e.g. image reading/writing)

ndimage: various functions for multi-dimensional image processing

optimize: optimization algorithms including linear programming

signal: signal processing tools

sparse: sparse matrix and related algorithms

spatial: KD-trees, nearest neighbors, distance functions

special: special functions

stats: statistical functions

weave: tool for writing C/C++ code as Python multiline strings

2.5 Pygame:

Pygame (the library) is a Free and Open Source python programming language library for making multimedia applications like games built on top of the excellent SDL library. Like SDL, pygame is highly portable and runs on nearly every platform and operating system.

It does not require OpenGL. With many people having broken OpenGL setups, requiring OpenGL exclusively will cut into your user base significantly. Pygame uses either opengl, directx, windib, X11, linux frame buffer, and many other different backends including an ASCII art backend. OpenGL is often broken on linux systems, and also on windows systems - which is why professional games use multiple backends.

Multi core CPUs can be used easily. With dual core CPUs common, and 8 core CPUs cheaply available on desktop systems, making use of multi core CPUs allows you to do more in your game. Selected pygame functions release the dreaded python GIL, which is something you can do from C code.

Uses optimized C, and Assembly code for core functions. C code is often 10-20 times faster than python code, and assembly code can easily be 100x or more times faster than python code.

It is truly portable and supports Linux (pygame comes with most mainstream linux distributions), Windows (95,98,me,2000,XP,vista, 64bit windows etc), Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, and QNX. The code contains support for AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS, and OS/2, but these are not officially supported. It's Simple and easy to use. Kids and adults make games with pygame. Before the Raspberry Pi, the microbit, or the OLPC, Pygame has been taught in courses to young kids, and college students. It's also used by people who first programmed in z80 assembler, or c64 basic, and for Indie game productions.It also does not require a GUI to use all functions. You can use pygame without a monitor - like if you want to use it just to process images, get joystick input, or play sounds.It does not have hundreds of thousands of lines of code for things you won't use anyway. The core is kept simple, and extra things like GUI libraries, and effects are developed separately outside of pygame. It's not the best game library.

2.6 Imutils:

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

2.7 Resizing:

Resizing an image in OpenCV is accomplished by calling the cv2.resize function. However, special care needs to be taken to ensure that the aspect ratio is maintained. This resize function of imutils maintains the aspect ratio and provides the keyword arguments width and height so the image can be resized to the intended width/height while (1) maintaining aspect ratio and (2) ensuring the dimensions of the image do not have to be explicitly computed by the developer.

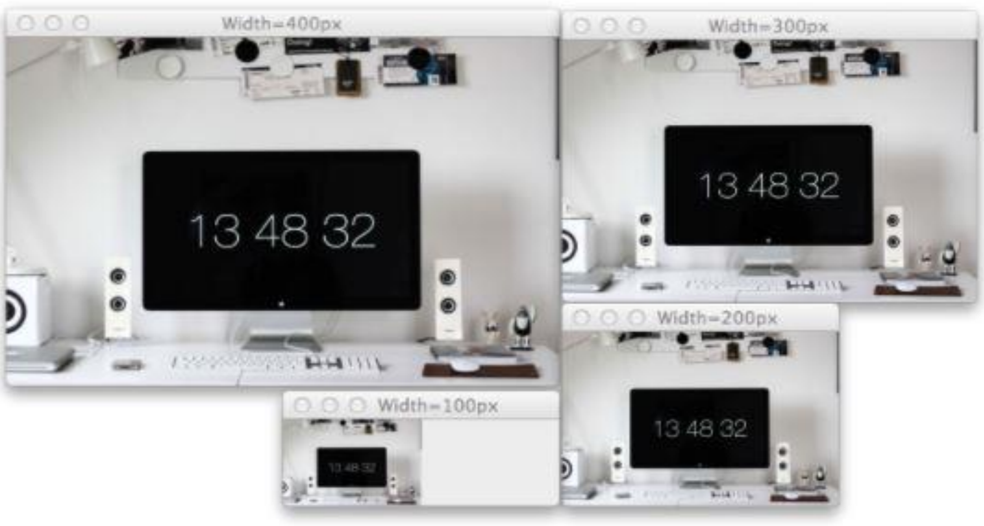


Fig 2.8.Resizing

CHAPTER 3
IMAGE PROCESSING TECHNIQUES AND EAR

3.1 Types of Image Processing Techniques:

- Binarization
- Pixelation
- Tree based SVM
- Active Appearance Model(AAM)

3.1.1 Binarization:

Binarization is the process of converting a pixel image to a binary image. A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color. In the document-scanning industry, this is often referred to as "bi-tonal".

Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit—i.e., a 0 or 1. The names black-and-white, B&W, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images. In Photoshop parlance, a binary image is the same as an image in "Bitmap" mode.

Binary images often arise in digital image processing as masks or as the result of certain operations such as segmentation, thresholding, and dithering. Some input/output devices, such as laser printers, fax machines, and bi-level computer displays, can only handle bi-level images.

A binary image can be stored in memory as a bitmap, a packed array of bits. A 640×480 image requires 37.5 KiB of storage. Because of the small size of the image files, fax machine and document management solutions usually use this format. Most binary images also compress well with simple run-length compression schemes.

Binary images can be interpreted as subsets of the two-dimensional integer lattice Z2; the field of morphological image processing was largely inspired by this view.

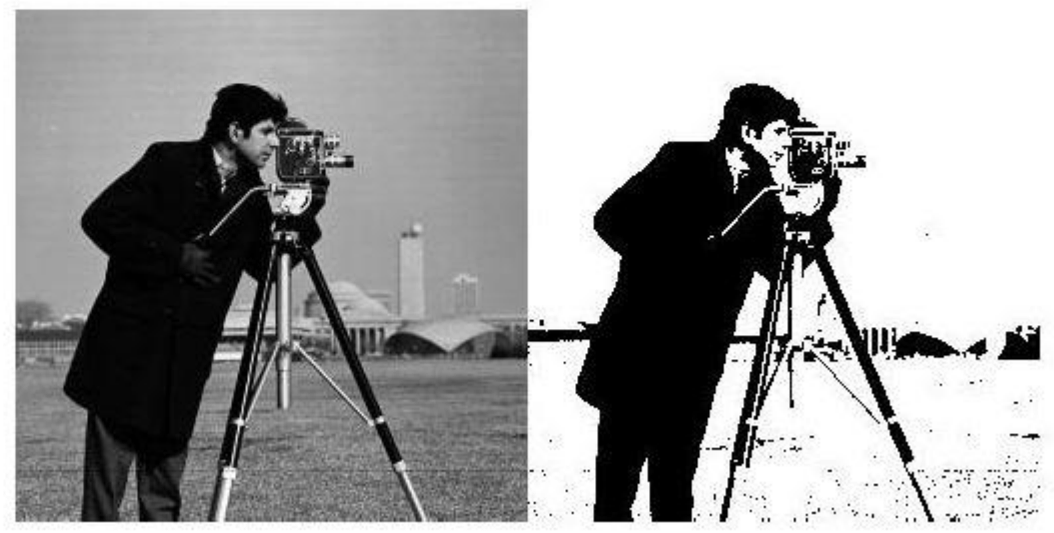


Fig 3.1.Binarization

3.1.2 Pixelation:

Pixelation is caused by displaying a bitmap or a section of a bitmap at such a large size that individual pixels, small single-colored square display elements that comprise the bitmap, are visible. Such an image is said to be pixelated. The pixelated image can be compared with a reference image to find the eye of the driver.

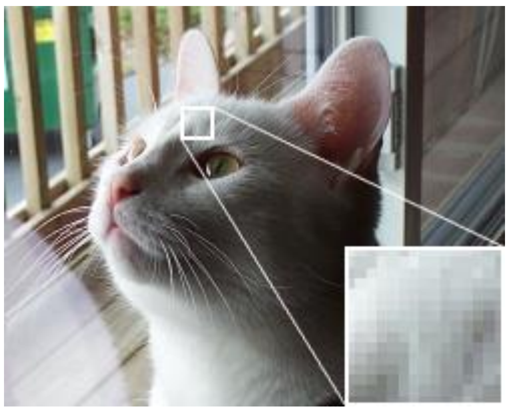


Fig 3.2.Pixelation

3.1.3 Shape model:

3.1.3.a. Tree based SVM:

Decision-tree-based support vector machine which combines support vector machines and decision tree is an effective way for solving multi-class problems. A problem exists in this method is that the division of the feature space depends on the structure of a decision tree, and the structure of the tree relate closely to the performance of the classifier. To maintain high generalization ability, the most separable classes should be separated at the upper nodes of a decision tree. Distance measure is often used as a separability measure between classes, but the distance between class centers cannot reflect the distribution of the classes. After analyzing the tree structure and the classification performance of the decision-tree-based support vector machine, a new separability measure is defined based on the distribution of the training samples in the feature space, the defined separability measure was used in the formation of the decision tree, and an improved algorithm for decision-tree-based support vector machine is proposed. Classification experiments prove the effectiveness of the improved algorithm for decision-tree-based support vector machine.Here we show the mean shape _m and deformation modes (eigenvectors of_m) learned in our tree structured, max-margin model. It captures much of the relevant elastic deformation, but produces some unnatural deformations because it lacks loopy spatial constraints (e.g., the left corner of the mouth in the lower right plot).

3.1.3.b. Active Appearance Model (AAM)

An active appearance model (AAM) is a computer vision algorithm for matching a statistical model of object shape and appearance to a new image. They are built during a training phase. A set of images, together with coordinates of landmarks that appear in all of the images, is provided to the training supervisor.

The model was first introduced by Edwards, Cootes and Taylor in the context of face analysis at the 3rd International Conference on Face and Gesture Recognition, 1998.Cootes, Edwards and Taylor further described the approach as a general method in computer vision at the European Conference on Computer Vision in the same year. The approach is widely used for matching and tracking faces and for medical image interpretation.

The algorithm uses the difference between the current estimate of appearance and the target image to drive an optimization process. By taking advantage of the least squares techniques, it can match to new images very swiftly.

It is related to the active shape model (ASM). One disadvantage of ASM is that it only uses shape constraints (together with some information about the image structure near the landmarks), and does not take advantage of all the available information – the texture across the target object. This can be modelled using an AAM.

Here we show the mean shape and deformation modes of the full-covariance Gaussian shape model used by AAMs. Note we exaggerate the deformations for visualization purposes. Tree based SVM ouotperforms AAM presumably because it is easier to optimize and allows for joint, discriminative training of part appearance models.

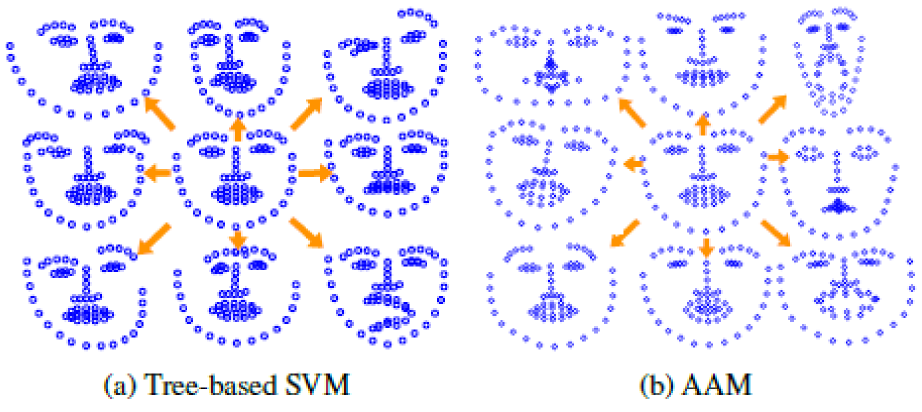


Fig 3.3.Tree based SVM and AAM

3.2 Facial landmark using 68 point shape predictor:

Facial landmarks are used to localize and represent salient regions of the face, such as:

- Eyes
- Eyebrows
- Nose
- Mouth
- Jawline

Detecting facial landmarks is a *subset* of the *shape prediction* problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape.

In the context of facial landmarks, our goal is detect important facial structures on the face using shape prediction methods.

Detecting facial landmarks is therefore a two step process:

- **Step #1:** Localize the face in the image.
- **Step #2:** Detect the key facial structures on the face ROI.

Face detection (Step #1) can be achieved in a number of ways.

We could use OpenCV's built-in Haar cascades.

We might apply a pre-trained HOG + Linear SVM object detector specifically for the task of face detection.

Or we might even use deep learning-based algorithms for face localization.

In either case, the actual algorithm used to detect the face in the image doesn't matter. Instead, what's important is that through some method we obtain the face bounding box (i.e., the *(x, y)*-coordinates of the face in the image).

Given the face region we can then apply **Step #2: detecting key facial structures in the face region.**

There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions:

- Mouth
- Right eyebrow
- Left eyebrow
- Right eye
- Left eye
- Nose
- Jaw

The facial landmark detector included in the dlib library is an implementation of the *One Millisecond Face Alignment with an Ensemble of Regression Trees* paper by Kazemi and Sullivan (2014).

This method starts by using:

1. A training set of labeled facial landmarks on an image. These images are *manually labeled*, specifying **specific** *(x, y)*-coordinates of regions surrounding each facial structure.
2. *Priors*, of more specifically, the *probability on distance* between pairs of input pixels.

Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the *pixel intensities themselves* (i.e., no "feature extraction" is taking place).

The end result is a facial landmark detector that can be used to **detect facial landmarks in *real-time*** with **high quality predictions**.

For more information and details on this specific technique, be sure to read the paper by Kazemi and Sullivan linked to above, along with the official dlib announcement.

3.2.1.Understanding dlib’s facial landmark detector

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of ***68 (x, y)-coordinates*** that map to facial structures on the face.

The indexes of the 68 coordinates can be visualized on the image below:

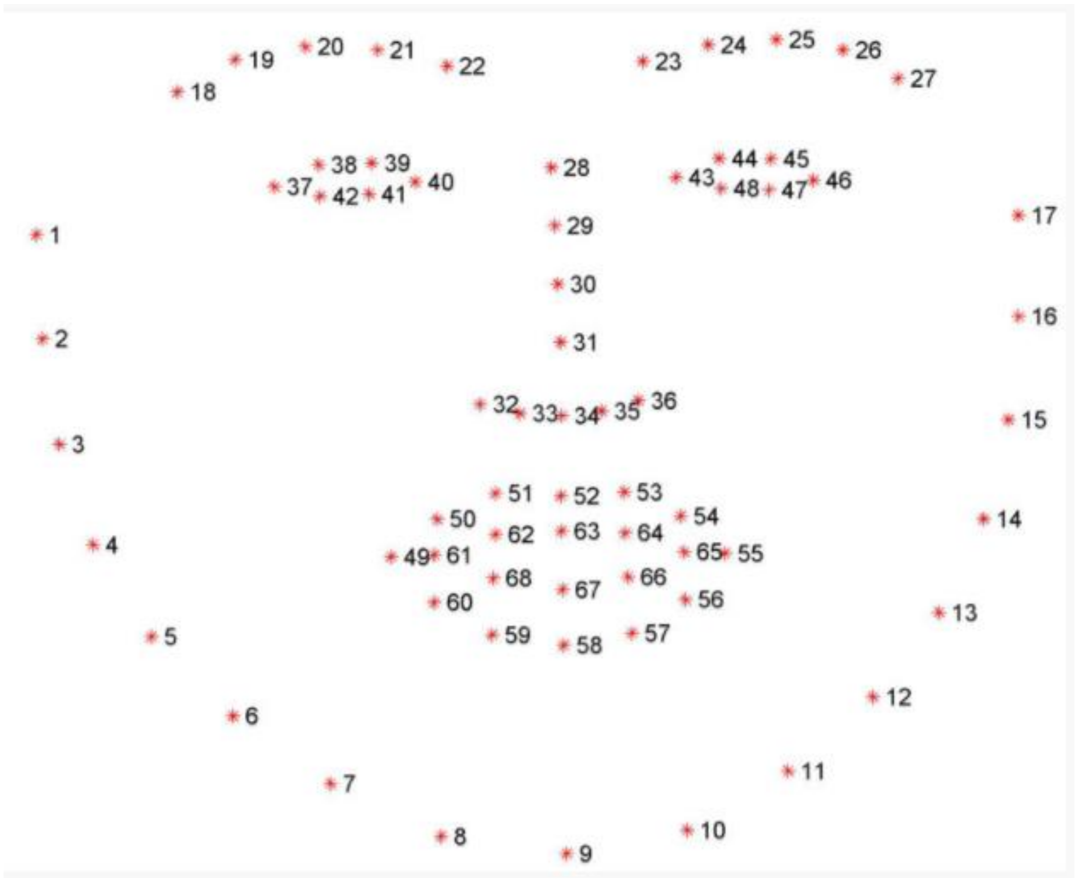


Fig 3.4. Facial landmarks

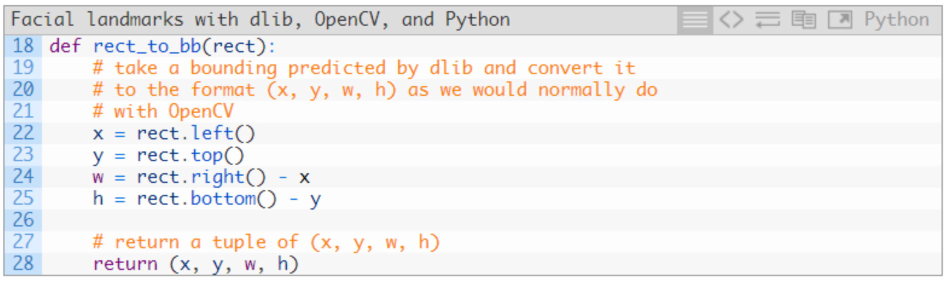
These annotations are part of the 68 point iBUG 300-W dataset which the dlib facial landmark predictor was trained on.

It’s important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the HELEN dataset.

Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data — this is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

3.3 Detecting facial landmarks with dlib, OpenCV, and Python:

The first utility function is `rect_to_bb` , short for “rectangle to bounding box”:



```python
def rect_to_bb(rect):
    # take a bounding predicted by dlib and convert it
    # to the format (x, y, w, h) as we would normally do
    # with OpenCV
    x = rect.left()
    y = rect.top()
    w = rect.right() - x
    h = rect.bottom() - y

    # return a tuple of (x, y, w, h)
    return (x, y, w, h)
```

Fig 3.5. Detecting facial landmarks with dlib, OpenCV, and Python

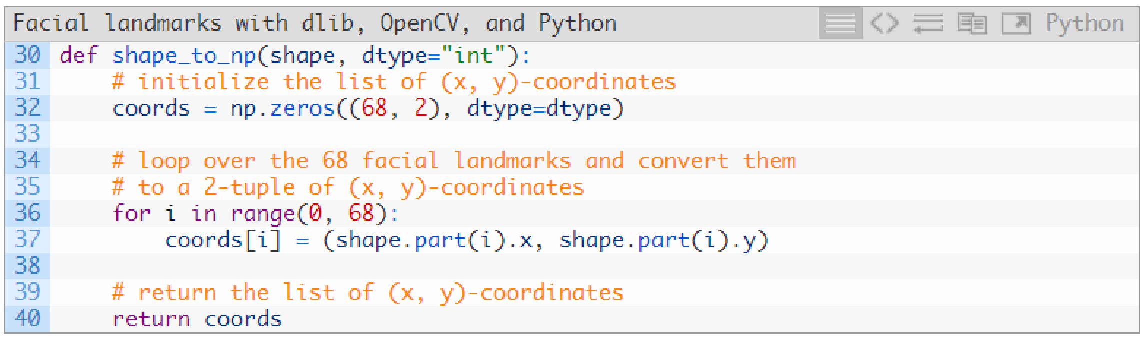
This function accepts a single argument, rect , which is assumed to be a bounding box rectangle produced by a dlib detector (i.e., the face detector).

The rect object includes the *(x, y)*-coordinates of the detection.

However, in OpenCV, we normally think of a bounding box in terms of *“(x, y, width, height)”* so as a matter of convenience, the rect_to_bb function takes this rect object and transforms it into a 4-tuple of coordinates.

Again, this is simply a matter of conveinence and taste.

Secondly, we have the shape_to_np function:



```python
def shape_to_np(shape, dtype="int"):
    # initialize the list of (x, y)-coordinates
    coords = np.zeros((68, 2), dtype=dtype)

    # loop over the 68 facial landmarks and convert them
    # to a 2-tuple of (x, y)-coordinates
    for i in range(0, 68):
        coords[i] = (shape.part(i).x, shape.part(i).y)

    # return the list of (x, y)-coordinates
    return coords
```

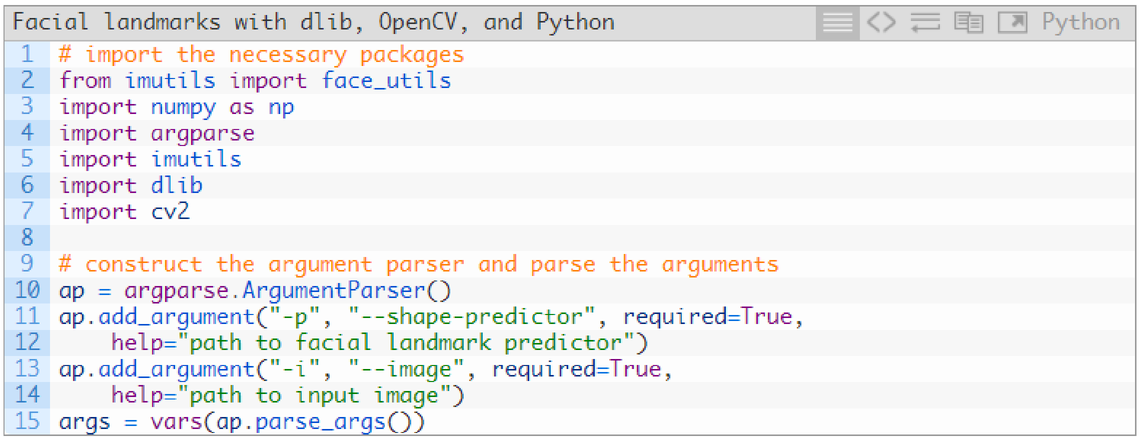
Fig 3.6. shape_to_np function

The dlib face landmark detector will return a shape object containing the 68 *(x, y)*-coordinates of the facial landmark regions.

Using the shape_to_np function, we cam convert this object to a NumPy array, allowing it to “play nicer” with our Python code.

Given these two helper functions, we are now ready to detect facial landmarks in images.

Open up a new file, name it facial_landmarks.py , and insert the following code:



```
1  # import the necessary packages
2  from imutils import face_utils
3  import numpy as np
4  import argparse
5  import imutils
6  import dlib
7  import cv2
8  
9  # construct the argument parser and parse the arguments
10 ap = argparse.ArgumentParser()
11 ap.add_argument("-p", "--shape-predictor", required=True,
12     help="path to facial landmark predictor")
13 ap.add_argument("-i", "--image", required=True,
14     help="path to input image")
15 args = vars(ap.parse_args())
```

Fig 3.7. Imported libraries

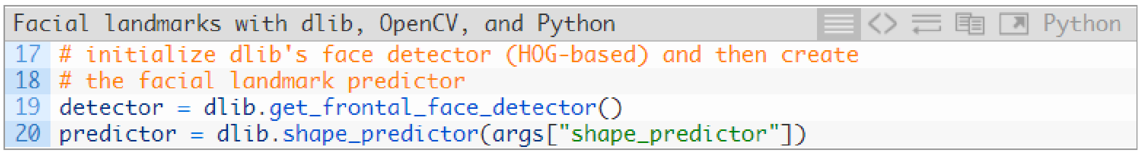
Lines 2-7 import our required Python packages.

We'll be using the face_utils submodule of imutils to access our helper functions detailed above.We'll then import dlib.

Lines 10-15 parse our command line arguments:

- --shape-predictor : This is the path to dlib's pre-trained facial landmark detector. We can download the detector model here or you can use the ***"Downloads"*** section of this post to grab the code + example images + pre-trained detector as well.
- --image : The path to the input image that we want to detect facial landmarks on.

Now that our imports and command line arguments are taken care of, now initialize dlib's face detector and facial landmark predictor:



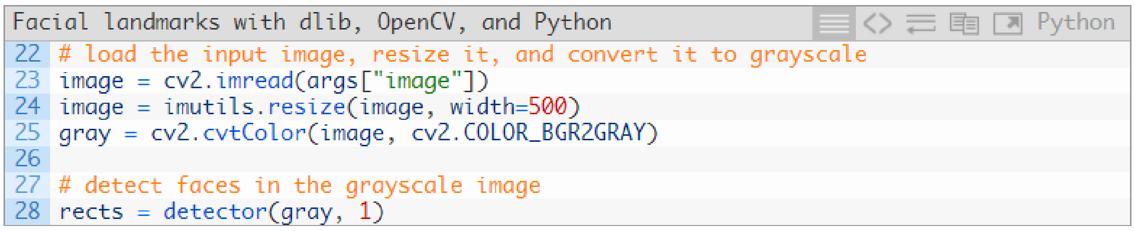
```
17 # initialize dlib's face detector (HOG-based) and then create
18 # the facial landmark predictor
19 detector = dlib.get_frontal_face_detector()
20 predictor = dlib.shape_predictor(args["shape_predictor"])
```

Fig 3.8. Initializing dlib's face detector and facial landmark predictor

Line 19 initializes dlib's pre-trained face detector based on a modification to the standard Histogram of Oriented Gradients + Linear SVM method for object detection.

Line 20 then loads the facial landmark predictor using the path to the supplied --shape-predictor .

But before we can actually detect facial landmarks, we first need to detect the face in our input image:



```python
22 # load the input image, resize it, and convert it to grayscale
23 image = cv2.imread(args["image"])
24 image = imutils.resize(image, width=500)
25 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
26
27 # detect faces in the grayscale image
28 rects = detector(gray, 1)
```

Fig 3.9. Detecting the facial landmarks

Line 23 loads our input image from disk via OpenCV, then pre-processes the image by resizing to have a width of 500 pixels and converting it to grayscale (**Lines 24 and 25**).

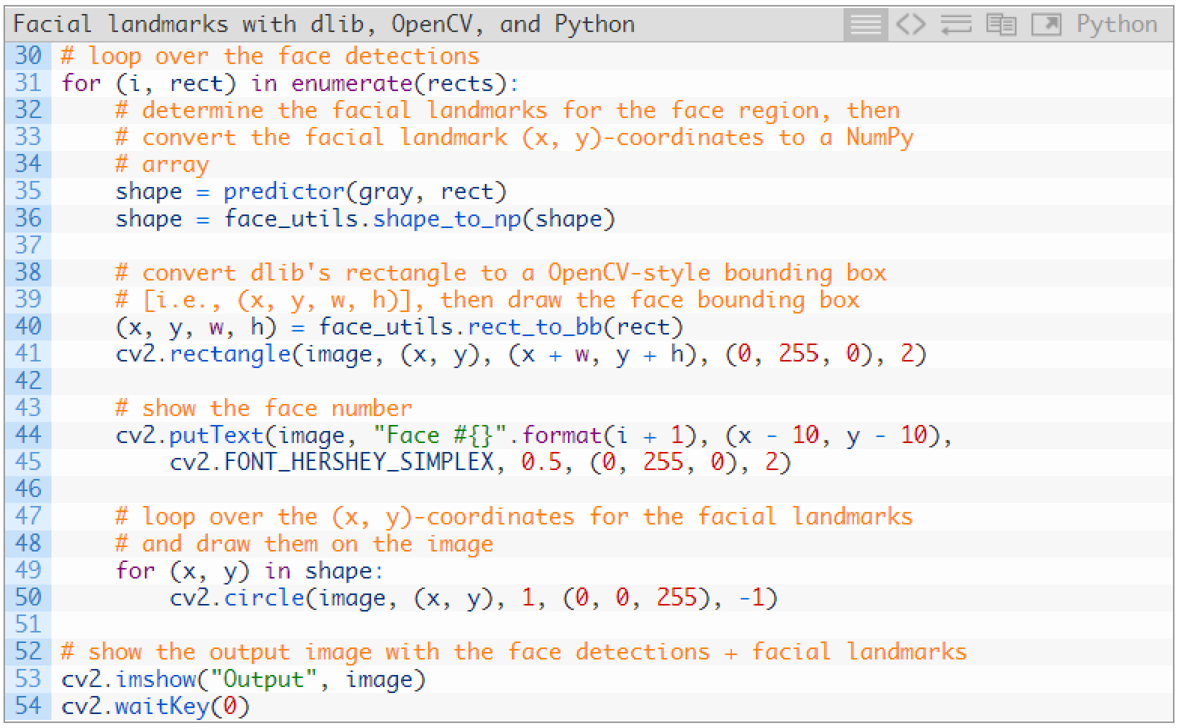
Line 28 handles detecting the bounding box of faces in our image.

The first parameter to the detector is our grayscale image (although this method can work with color images as well).

The second parameter is the number of image pyramid layers to apply when upscaling the image prior to applying the detector (this it the equivalent of computing cv2.pyrUp *N* number of times on the image).

The benefit of increasing the resolution of the input image prior to face detection is that it may allow us to detect *more* faces in the image — the downside is that the larger the input image, the more computationally expensive the detection process is.

Given the *(x, y)*-coordinates of the faces in the image, we can now apply facial landmark detection to each of the face regions.



```
30 # loop over the face detections
31 for (i, rect) in enumerate(rects):
32     # determine the facial landmarks for the face region, then
33     # convert the facial landmark (x, y)-coordinates to a NumPy
34     # array
35     shape = predictor(gray, rect)
36     shape = face_utils.shape_to_np(shape)
37 
38     # convert dlib's rectangle to a OpenCV-style bounding box
39     # [i.e., (x, y, w, h)], then draw the face bounding box
40     (x, y, w, h) = face_utils.rect_to_bb(rect)
41     cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
42 
43     # show the face number
44     cv2.putText(image, "Face #{}".format(i + 1), (x - 10, y - 10),
45         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
46 
47     # loop over the (x, y)-coordinates for the facial landmarks
48     # and draw them on the image
49     for (x, y) in shape:
50         cv2.circle(image, (x, y), 1, (0, 0, 255), -1)
51 
52 # show the output image with the face detections + facial landmarks
53 cv2.imshow("Output", image)
54 cv2.waitKey(0)
```

Fig 3.10. Obtaining face with bounding box

We start looping over each of the face detections on **Line 31**.

For each of the face detections, we apply facial landmark detection on **Line 35**, giving us the 68 *(x, y)*-coordinates that map to the specific facial features in the image.

Line 36 then converts the dlib shape object to a NumPy array with shape *(68, 2)*.

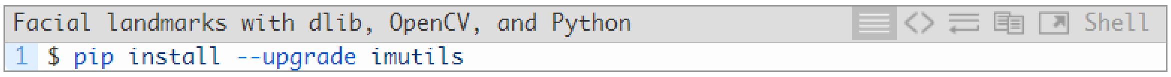
Lines 40 and 41 draw the bounding box surrounding the detected face on the image while **Lines 44 and 45** draw the index of the face.

Finally, **Lines 49 and 50** loop over the detected facial landmarks and draw each of them individually.

Lines 53 and 54 simply display the output image to our screen.

3.4 Facial landmark visualizations:

Before we test our facial landmark detector, make sure you have upgraded to the latest version of imutils which includes the face_utils.py file:

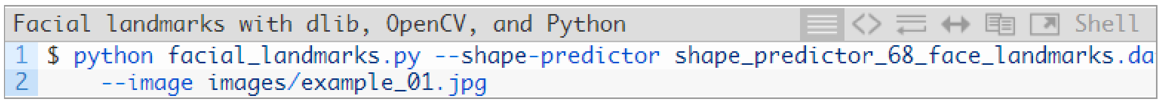


```
$ pip install --upgrade imutils
```

Fig 3.11. Upgrading imutils

Note: *If you are using Python virtual environments, make sure you upgrade the imutils inside the virtual environment.*

Once you've downloaded the .zip archive, unzip it, change directory to facial-landmarks , and execute the following command:



```
$ python facial_landmarks.py --shape-predictor shape_predictor_68_face_landmarks.da
    --image images/example_01.jpg
```

Fig 3.12. Execution command

3.5 Positioning the camera

The car's front is imagined to be a parabola and the camera is placed such that it points to the focus of parabola.

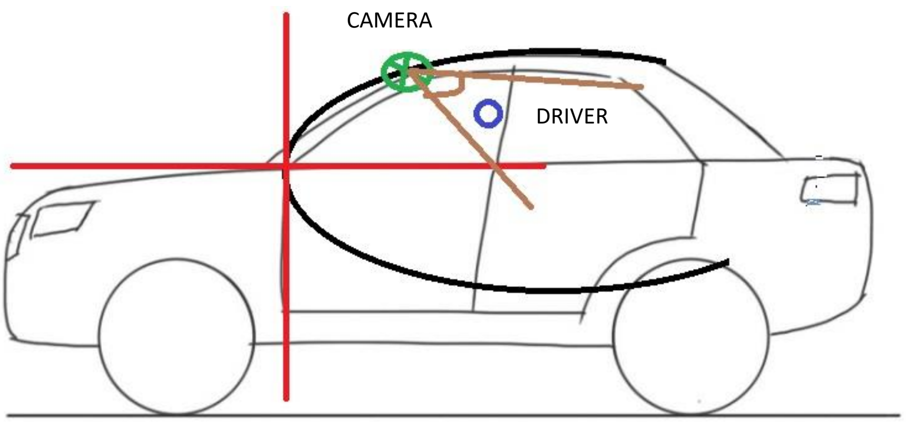


Fig 3.13. Positioning the camera

3.6 Eye Aspect Ratio:

The ratio between vertical and horizontal points indicated around the eye is known as Eye Aspect Ratio.

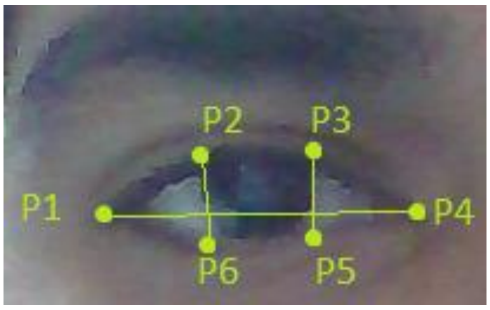


Fig 3.14. EAR points

There is a relation between the *width* and the *height* of these coordinates.

we can then derive an equation that reflects this relation called the *eye aspect ratio* (EAR).

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

Where *p1, p2, p3, p4, p5 and p6* are 2D facial landmark locations.

The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighting the denominator appropriately since there is only *one* set of horizontal points but *two* sets of vertical points.

CHAPTER 4
WORKING

4.1 FLOWCHART:

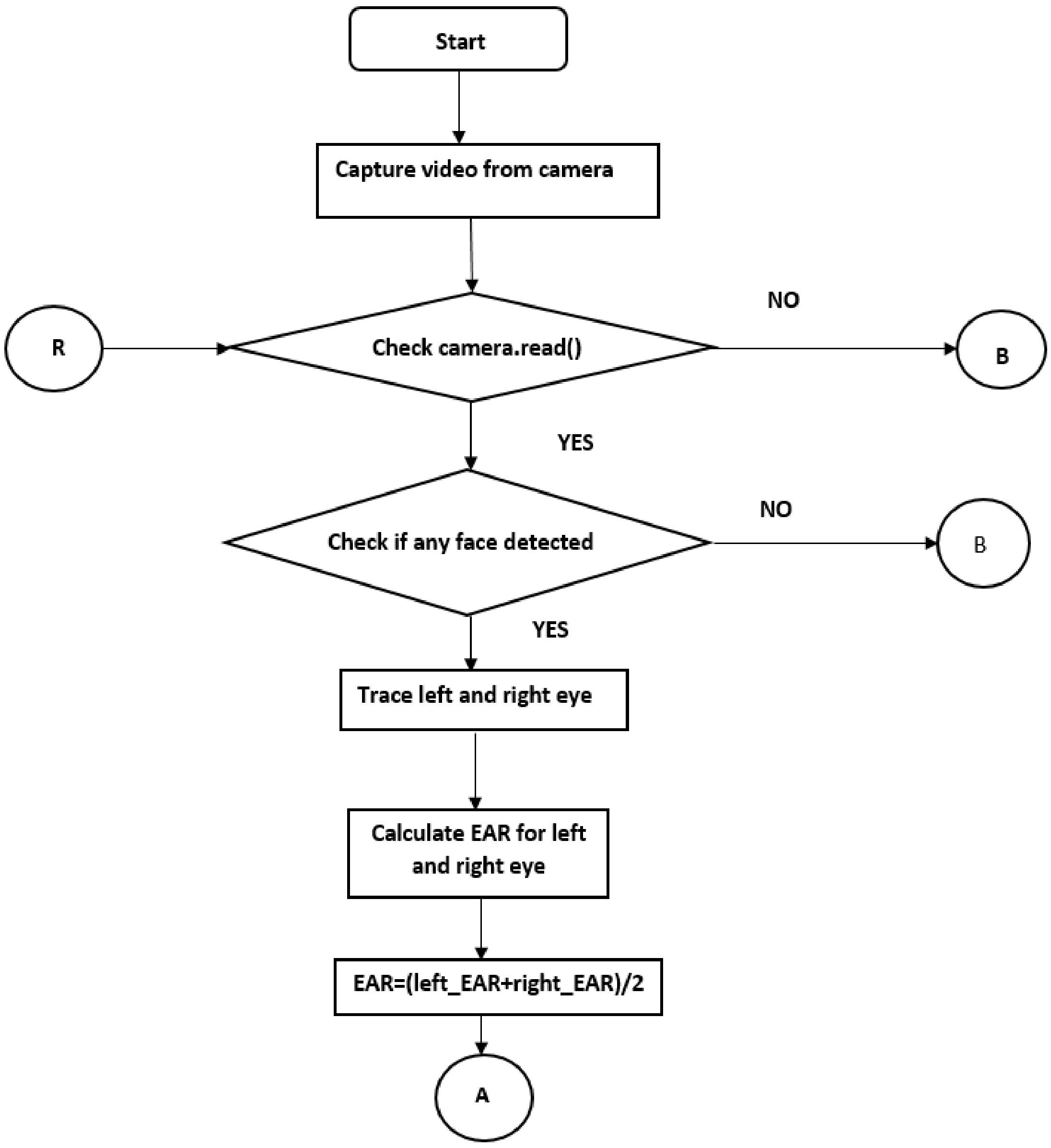

Start
Capture video from camera
R
Check camera.read()
NO
B
YES
Check if any face detected
NO
B
YES
Trace left and right eye
Calculate EAR for left and right eye
EAR=(left_EAR+right_EAR)/2
A

Fig 4.1. Flowchart part 1

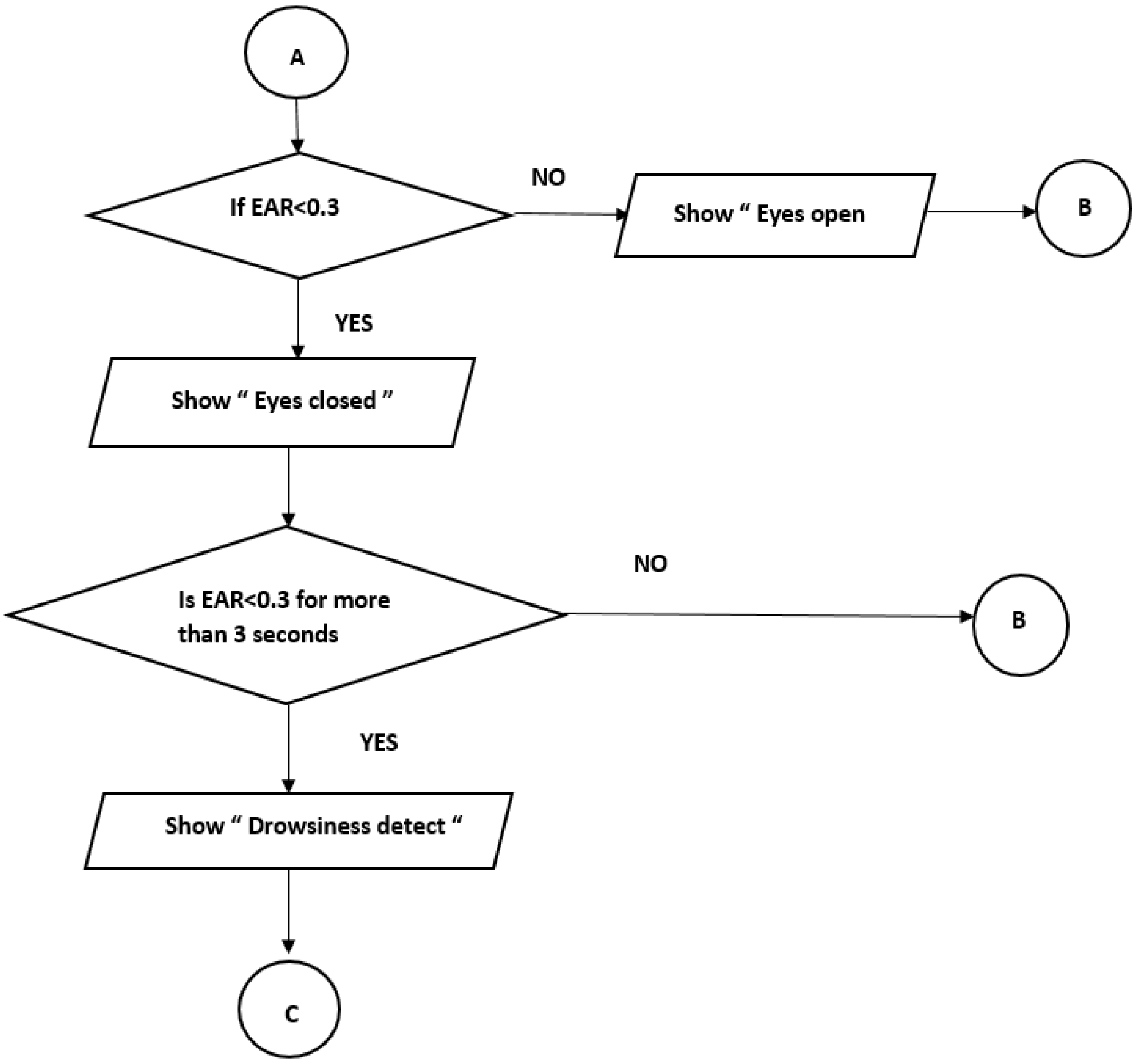

A
If EAR<0.3
NO
Show “ Eyes open
B
YES
Show “ Eyes closed ”
Is EAR<0.3 for more than 3 seconds
NO
B
YES
Show “ Drowsiness detect “
C

Fig 4.2. Flowchart part 2

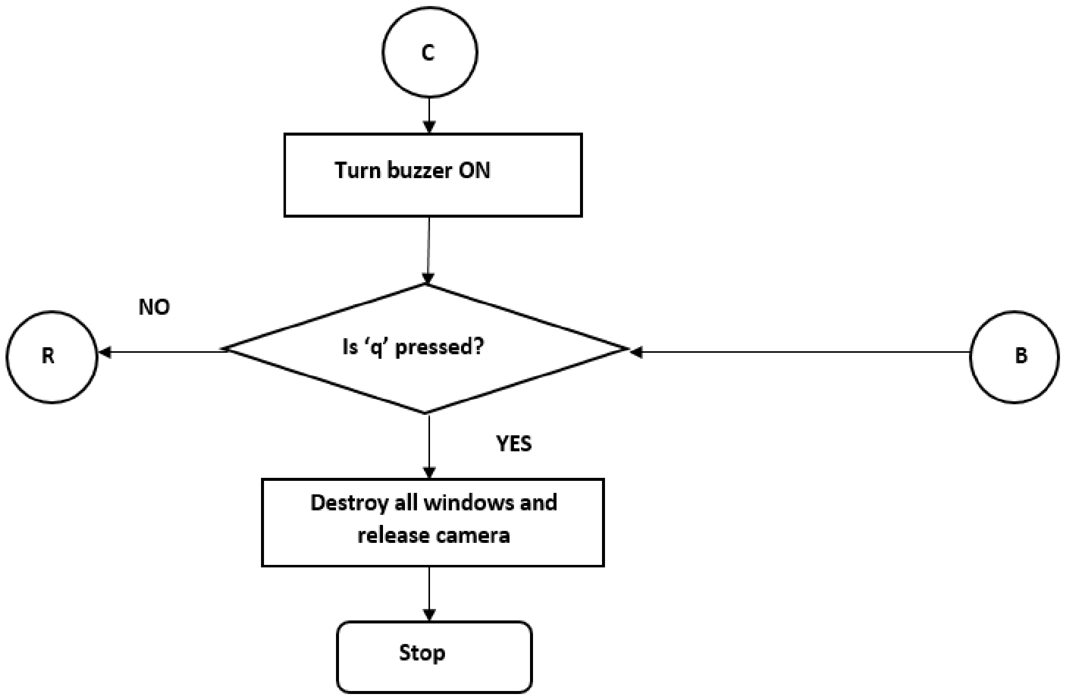


Fig 4.3. Flowchart part 3

4.2 Working:

A Camera is used to monitor the driver for drowsiness. The video signals from the camera are converted into frames. The number of frames produced per second from the video is 60 fps. Each frame is then processed. The frame is processed for finding any human face in its vicinity. If a human face is detected then the eyes of that human face is marked using 12 points,6 points for each eye. Using those points EAR is then calculated for finding the drowsiness. A threshold is set, if the calculated EAR is greater then the threshold then the eyes are open and if the calculated EAR is lower then the threshold then the eyes are closed.

If the EAR is below the threshold for more than 3 seconds then the driver is in a state of drowsiness. An alert system plays an alarm to warn the driver. If there is no drowsiness detected in the driver the system continuously checks for drowsiness until the exit protocol is executed. The system checks for drowsiness even at the time of warning the driver of his drowsiness and hence the system has excellent performance when it comes in detecting the drowsiness.

CHAPTER 5
OUTPUT AND RESULT

5.1 Output:

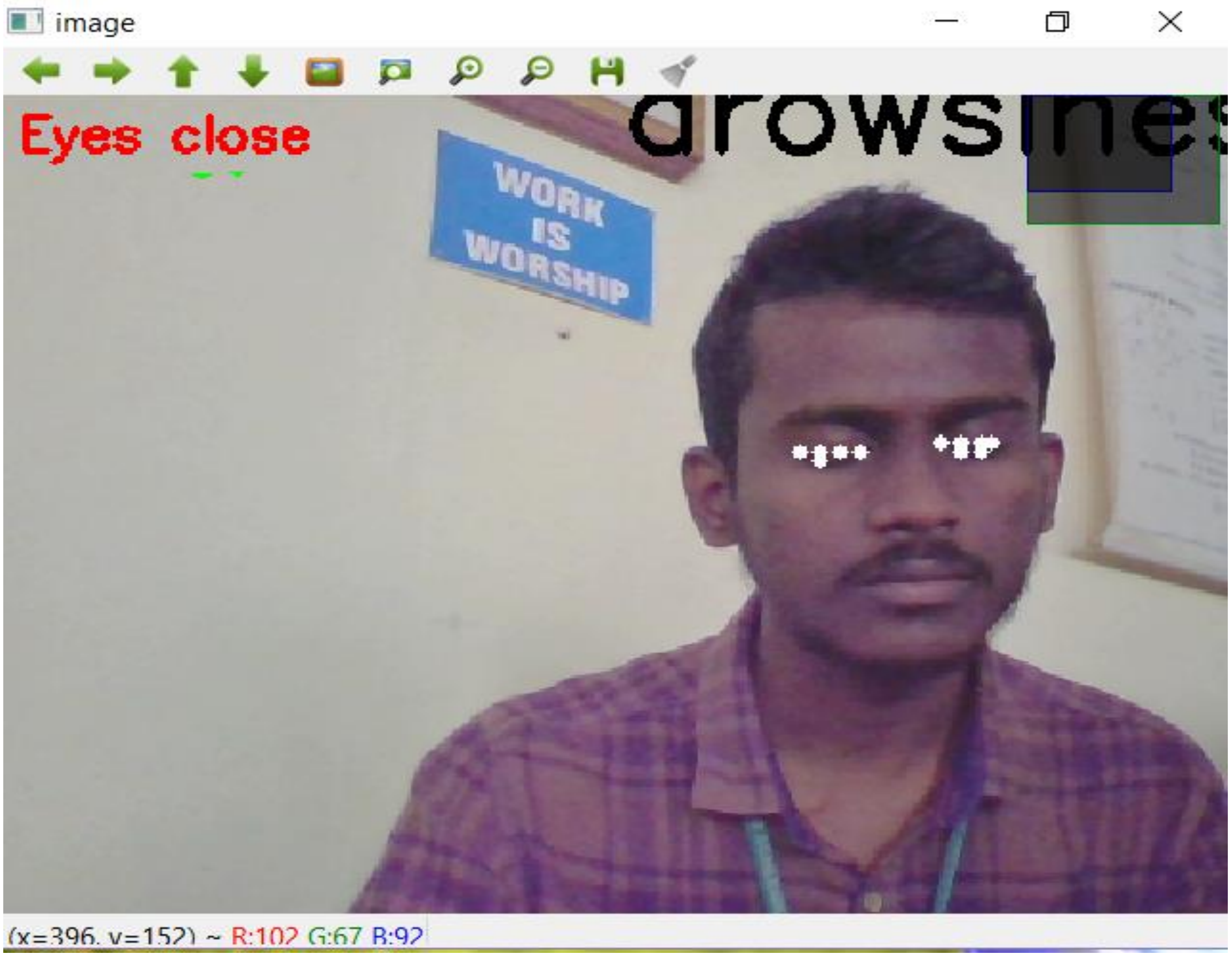


Fig 5.1. Straight view – Eyes close

This shows the result obtained from STRAIGHT view (i.e.,)the camera is in level with driver’s eye. The results were satisfactory for eyes close condition.

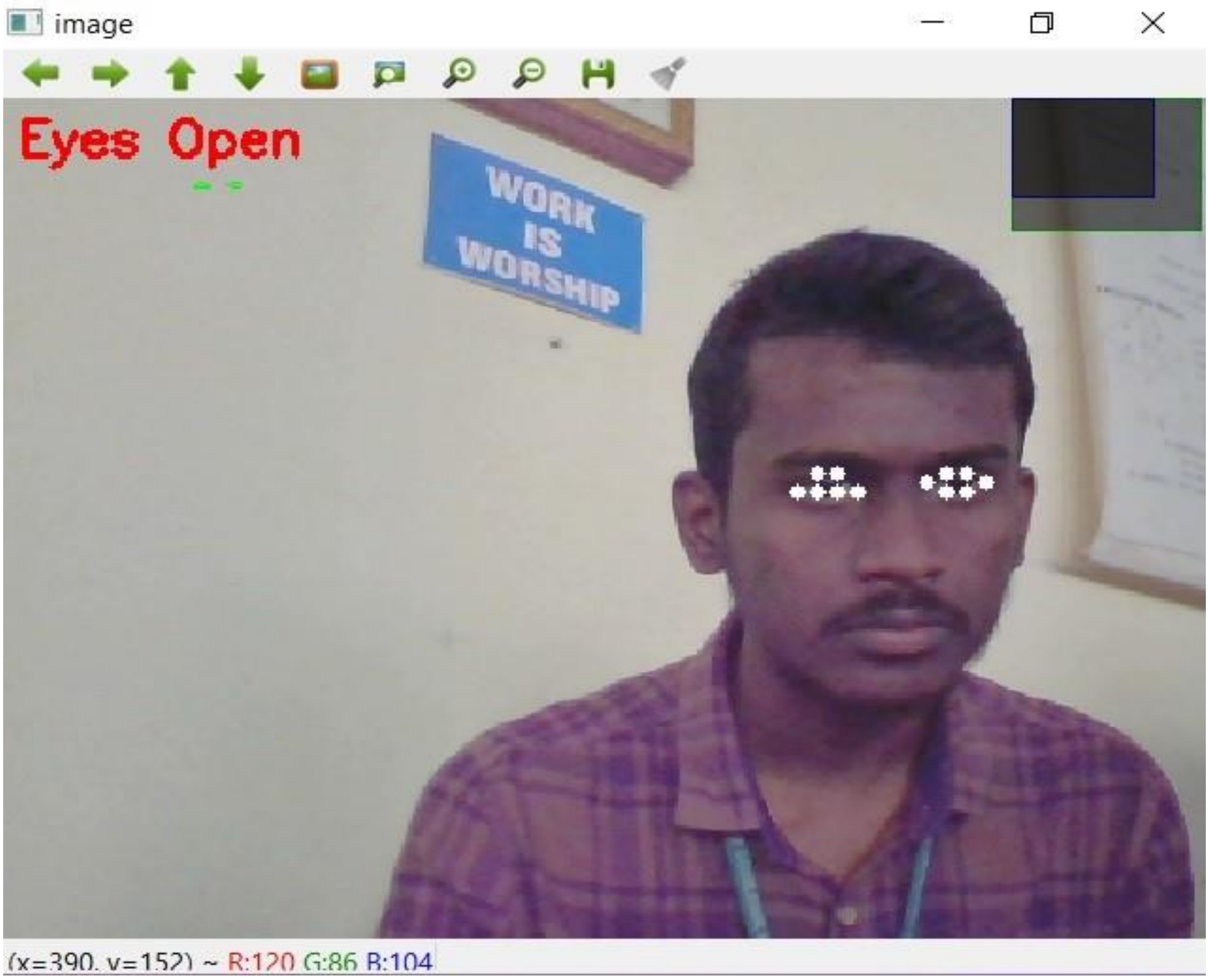


Fig 5.2. straight view – Eyes open

This shows the result obtained from STRAIGHT view (i.e.,)the camera is in level with driver’s eye. The results were satisfactory for eyes open condition.

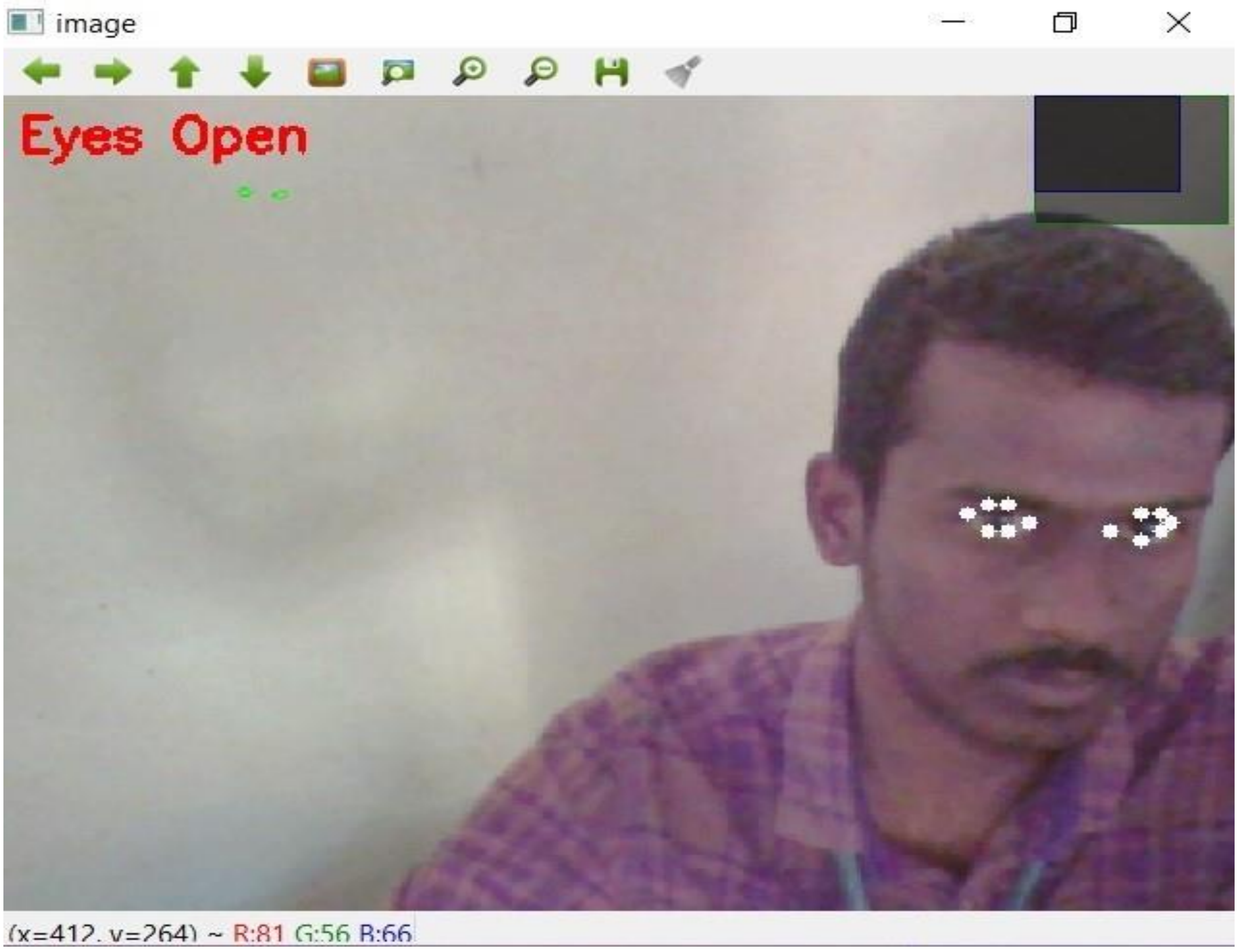


Fig 5.3. Top view – Eyes open

This shows the result obtained from TOP view (i.e.,)the camera is above the eye level of driver. The results were satisfactory for eyes open condition.

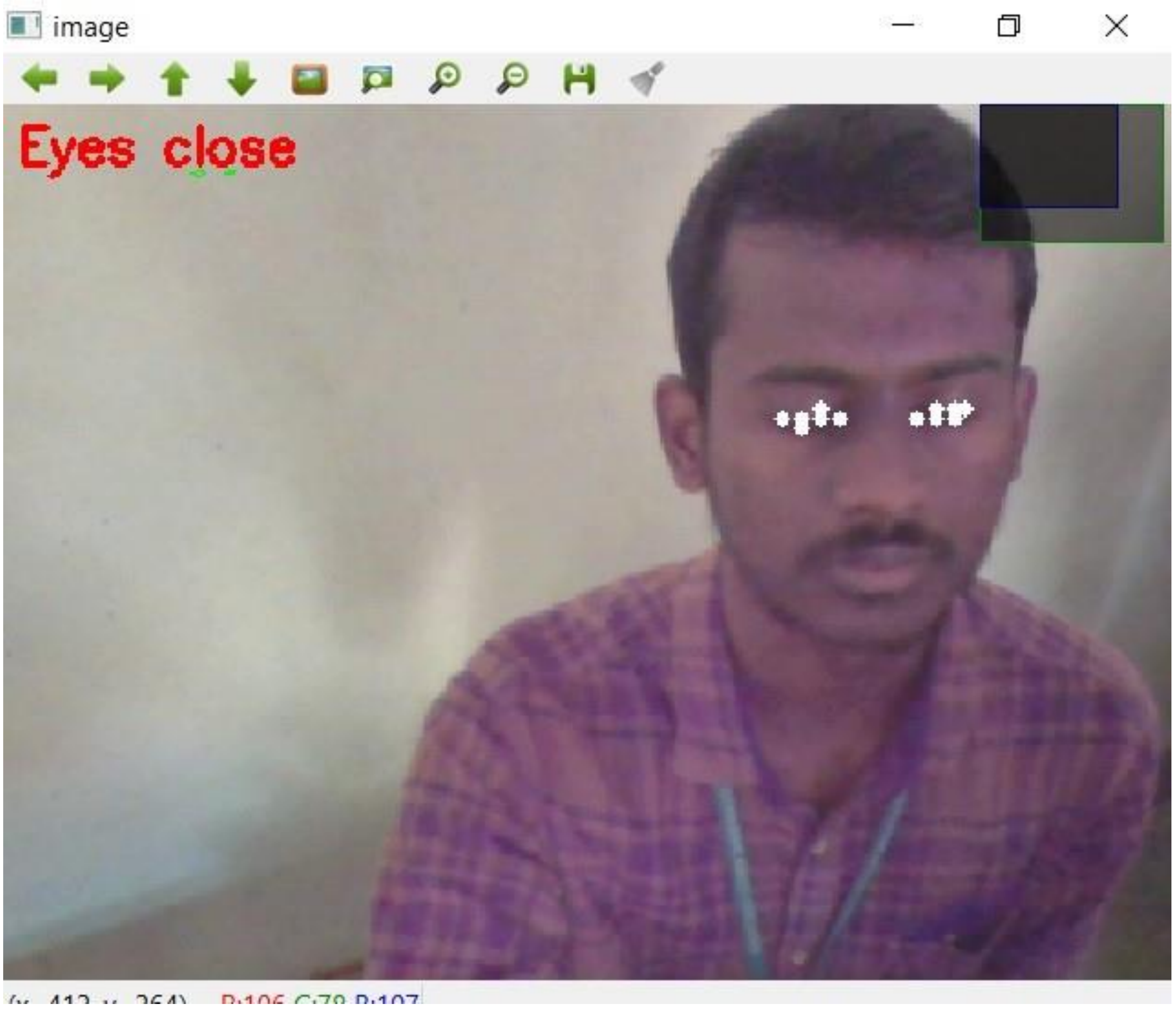


Fig 5.4. Top view – Eyes close

This shows the result obtained from TOP view (i.e.,)the camera is above the eye level of driver. The results were satisfactory for eyes close condition.

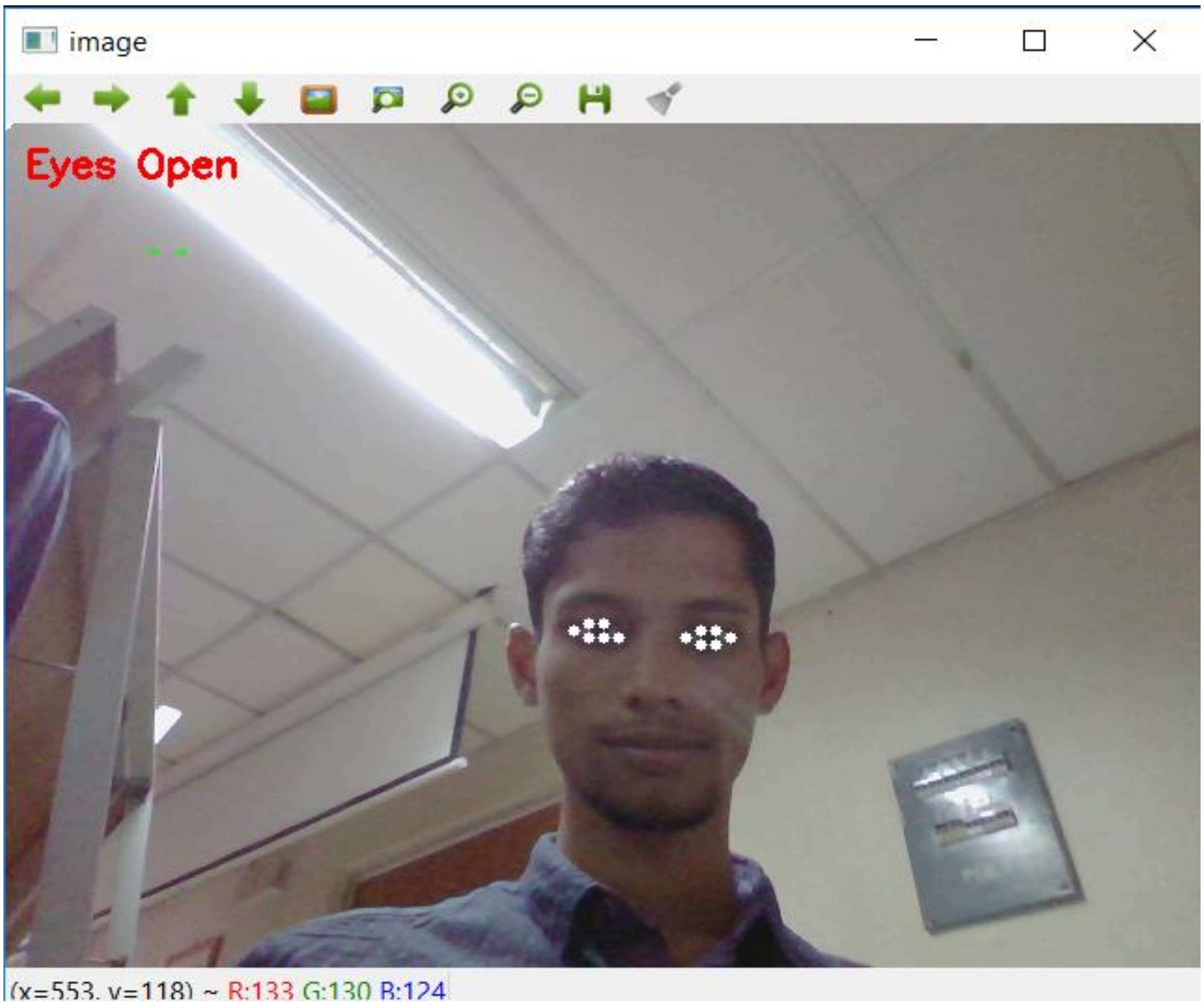


Fig 5.5. Bottom view – Eyes open

This shows the result obtained from BOTTOM view (i.e.,)the camera is below the eye level of driver. The results were satisfactory for eyes open condition.

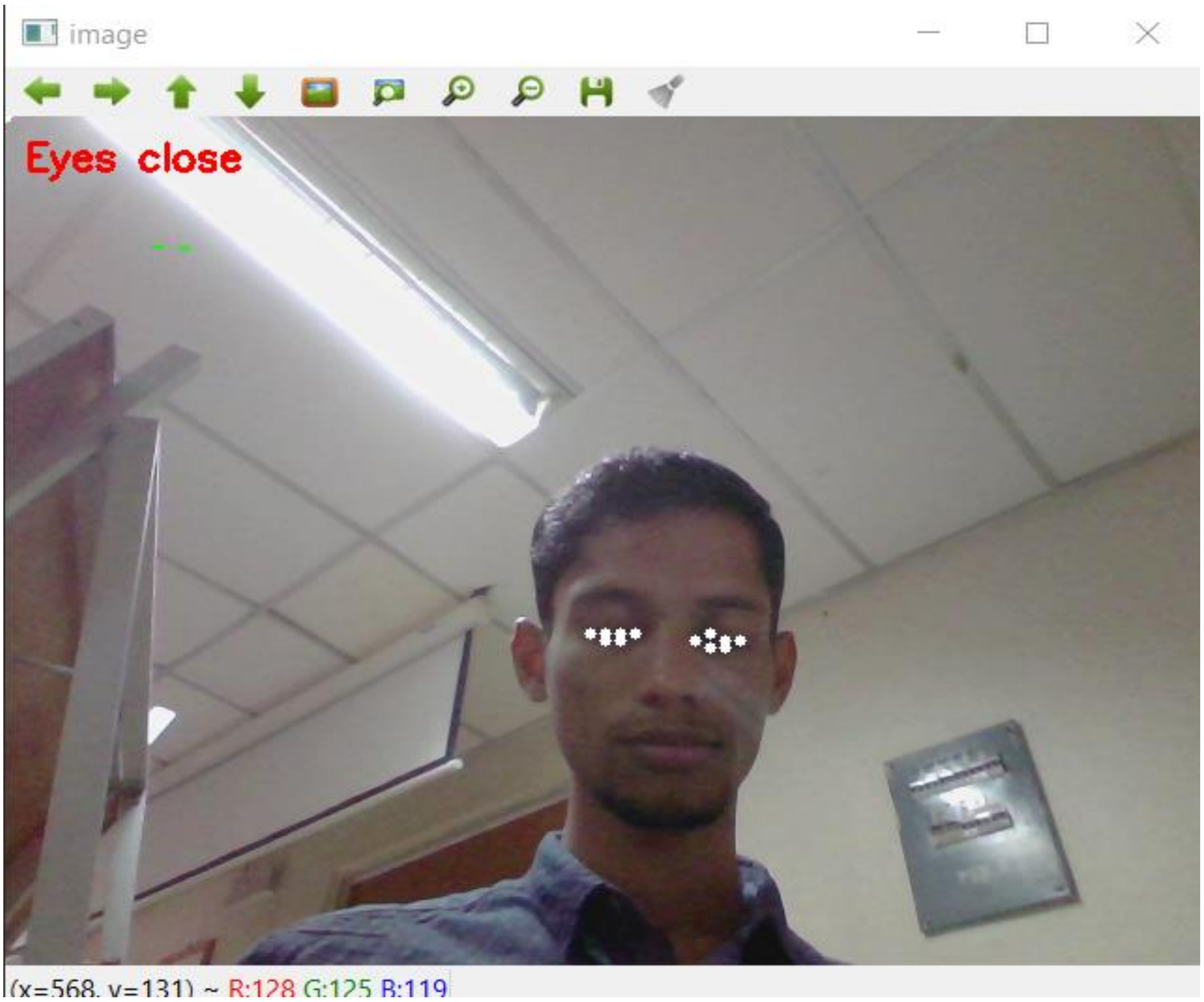


Fig 5.6. Bottom view – Eyes close

This shows the result obtained from BOTTOM view (i.e.,)the camera is below the eye level of driver. The results were satisfactory for eyes close condition.

5.2 Advantages:

- Cheaper and efficient due to avoidance of sensors.
- The response time in image processing using python is faster than Matlab image processing technique.
- The memory for reference image is reduced by using 68-point shape predictor library.

5.3 Applications:

- This project can be used in all Light MovingVehicles(LMV) and Heavy Moving Vehicles(HMV).
- Used by ophthalmologist in finding out eye aspec ratio(EAR) for research purposes.
- Used for long distant car and truck drivers.

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

6.1 Conclusion:

In Driver Assistance System, the scope of using openCV for image processing are immense. It is highly useful in processing images and also shows reliable performance in real time conditions. The problems associated with human drowsiness and other human errors can probably be overcome with openCV. Driving can be done better and smarter with the help of driver assistance system which uses openCV. This is compatible with many operating systems thus making it suitable for future improvements also.

This project "**REAL TIME EYE DETECTION AND TRACKING METHOD FOR DRIVER ASSISTANCE SYSTEM**" has been completed successfully and the output results are verified. The results are in line with the expected output. From this project, Drivers can easily be alerted if there is a chance for accident due to their drowsiness. This project is further adaptive for upgradation also.

6.2 Future scope:

This project can further be developed into an android application which can run in most of the smartphones and thus it can be made available free of cost for the driving community.

References:

[1] D’Orazio T, Leo M, Guaragnella C, Distante A (2007), “A visual approach for driver inattention detection”,Pattern Recogn 40(8):2341–2355.

[2] Qiang J, Xiaojie Y (2002), “Real-Time Eye, Gaze, and face pose tracking for monitoring driver vigilance”. Real-Time Imag 8:357–377

[3] Bakic, V., and Stockman, G., “Menu Selection by Facial Aspect, Proceedings Vision Interface”, Quebec, Canada, May, 1999.

[4] Kim, K., and Ramakrishna, R.S., “ Vision- Based Eye-Gaze Tracking for Human Computer Interface, Systems, Man and Cybernetics”, IEEE SMC’99 Conference Proc., No.2, pp.324-329, 1999.

[5] Mariani, R. “Face Learning Using a Sequence of Images, International Journal of Pattern Recognition and Artificial Intelligence”, Vol.14, No.5, pp.631-648, 2000.

[6] Morimoto, C., Koons, D., Amir, A., and Flickner, M., “Pupil Detection and Tracking Using Multiple Light Sources, Image and Vision Computing, Special issue on Advances in Facial Image Analysis and Recognition Technology”, Vol.18, No.4, pp.331-335, 2000.

[7] Pappu, R., and Beardsley, P.A., “A Qualitative Approach to Classifying Gaze Direction”, Proceedings of the Third IEEE International Conference on Automatic Face and Gesture Recognition, Nara, Japan, April, 1998.

[8] Singh,S., and Papanikolopoulos, P., “ Advances in Vision-Based Detection of Driver Fatigue”, Proceedings of the ITS America Ninth Annual Meeting, 1999.

[9] Smith, P., Shah, M., and Lobo, M.V., “Monitoring Head/Eye Motion for Driver Alertness with One Camera”, Proceedings of the International Conference on Pattern Recognition, 2000.

[10] Talmi, K., and Liu, J., “Eye and Gaze Tracking for Visually Controlled Interactive Stereoscopic Displays, Signal Processing: Image Communication”, No.14, pp.799-810, 1999.

APPENDIX

PROGRAMMING

```
from __future__ import division

import dlib

from imutils import face_utils

import cv2

import numpy as np

from scipy.spatial import distance as dist

import threading

import pygame

def start_sound():

    pygame.mixer.init()

    pygame.mixer.music.load("C:/Users/Acchu/Miniconda3/alert.ogg")

    pygame.mixer.music.play()


def resize(img, width=None, height=None, interpolation=cv2.INTER_AREA):

    global ratio

    w, h = img.shape

    if width is None and height is None:

        return img

    elif width is None:

        ratio = height / h

        width = int(w * ratio)
        resized = cv2.resize(img, (height, width), interpolation)

        return resized

    else:

        ratio = width / w

        height = int(h * ratio)

        resized = cv2.resize(img, (height, width), interpolation)

        return resized

######

def shape_to_np(shape, dtype="int"):

    coords = np.zeros((68, 2), dtype=dtype)

    for i in range(36,48):

        coords[i] = (shape.part(i).x, shape.part(i).y)

    return coords

def eye_aspect_ratio(eye):

    A = dist.euclidean(eye[1], eye[5])

    B = dist.euclidean(eye[2], eye[4])



            # compute the euclidean distance between the horizontal

            # eye landmark (x, y)-coordinates

    C = dist.euclidean(eye[0], eye[3])



            # compute the eye aspect ratio

    ear = (A + B) / (2.0 * C)
```

```
        # return the eye aspect ratio
    return ear
camera = cv2.VideoCapture(0)

predictor_path = 'C:/Users/Acchu/Miniconda3/shape_predictor_68_face_landmarks.dat_2'

detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(predictor_path)
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
total=0
alarm=False
while True:
    ret, frame = camera.read()
    if ret == False:
        print('Failed to capture frame from camera. Check camera index in cv2.VideoCapture(0)
\n')
        break

    frame_grey = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_resized = resize(frame_grey, width=120)
```

```
# Ask the detector to find the bounding boxes of each face. The 1 in the
# second argument indicates that we should upsample the image 1 time. This
# will make everything bigger and allow us to detect more faces.
    dets = detector(frame_resized, 1)


    if len(dets) > 0:
        for k, d in enumerate(dets):
            shape = predictor(frame_resized, d)
            shape = shape_to_np(shape)


            leftEye= shape[lStart:lEnd]
            rightEye= shape[rStart:rEnd]
            leftEAR= eye_aspect_ratio(leftEye)
            rightEAR = eye_aspect_ratio(rightEye)
            ear = (leftEAR + rightEAR) / 2.0
            leftEyeHull = cv2.convexHull(leftEye)


            rightEyeHull = cv2.convexHull(rightEye)
            cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
            cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
            if ear>.25:
                print (ear)
                total=0
                alarm=False
                cv2.putText(frame,"Eyes Open", (10, 30),cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0, 0, 255), 2)
            else:
                total+=1
                if total>20:
                    if not alarm:
                        alarm=True
                        d=threading.Thread(target=start_sound)
                        d.setDaemon(True)
                        d.start()
                        print ("drowsiness detect")
                        cv2.putText(frame, "drowsiness detect" ,(250,
30),cv2.FONT_HERSHEY_SIMPLEX, 1.7, (0, 0, 0), 4)
                cv2.putText(frame, "Eyes close".format(total), (10,
30),cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
            for (x, y) in shape:
                cv2.circle(frame, (int(x/ratio), int(y/ratio)), 3, (255, 255, 255), -1)
    cv2.imshow("image", frame)


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