PROTOTYPE DROWSINESS DETECTION SYSTEM

A THESIS SUBMITTED IN PARALLEL FULFULMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor in Technology

In

Electronics and Instrumentation Engineering

Under the guidance of :- Prof. U.C.Pati

Department of Electronics and Communication Engineering
National Institute of Technology, Rourkela

By:-

Abinash Dash(108EI038) &

Birendra Nath Tripathy(108EI026)

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis titled "Prototype Drowsiness Detection System" submitted by Abinash Dash(108EI038) and Birendra Nath Tripathy(108EI026) in partial fulfilment for the requirements for the award of Bachelor of Technology Degree in Electronics and Instrumentation Engineering at National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

Date:

Prof. U.C. Pati

Department of Electronics and

Communication Engineering

ACKNOWLEDGEMENT

We would like to articulate our deep gratitude to our project guide Prof. U.C.Pati for his guidance, advice and constant support in the project work. We would like to thank him specially for being our advisor here at National Institute of Technology, Rourkela. We would like to thank all faculty members and staff of the Department of Electronics and communication Engineering, N.I.T. Rourkela for their generous help in various ways for this project.

Last but not the least; we give our sincere thanks to all of our friends who have patiently extended all sorts of help in this project.

Abinash Dash

108EI038

Birendra Nath Tripathy

108EI026

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.

CONTENTS

CHAPTER-1

INTRODUCTION

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. [1]

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. It is designed for computational efficiency and with a strong focus on real time applications. It helps to build sophisticated vision applications quickly and easily. OpenCV satisfied the low processing power and high speed requirements of our application.

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

- Gather a data set of face and eye. These should be stored in one or more directories indexed by a text file. A lot of high quality data is required for the classifier to work well.
- The utility application createsamples() 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 reweighted so that more attention is paid to the data points where errors were made. This process continues until the total error over the dataset arising from the combined weighted vote of the decision trees falls below a certain threshold. [6]

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.

CHAPTER 2

Why OpenCV?

2.1 Eye blink detection using Matlab:-

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. [1]

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 Is OpenCV?

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 Intel architectures

[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. [4]

2.3 What Is Computer Vision?

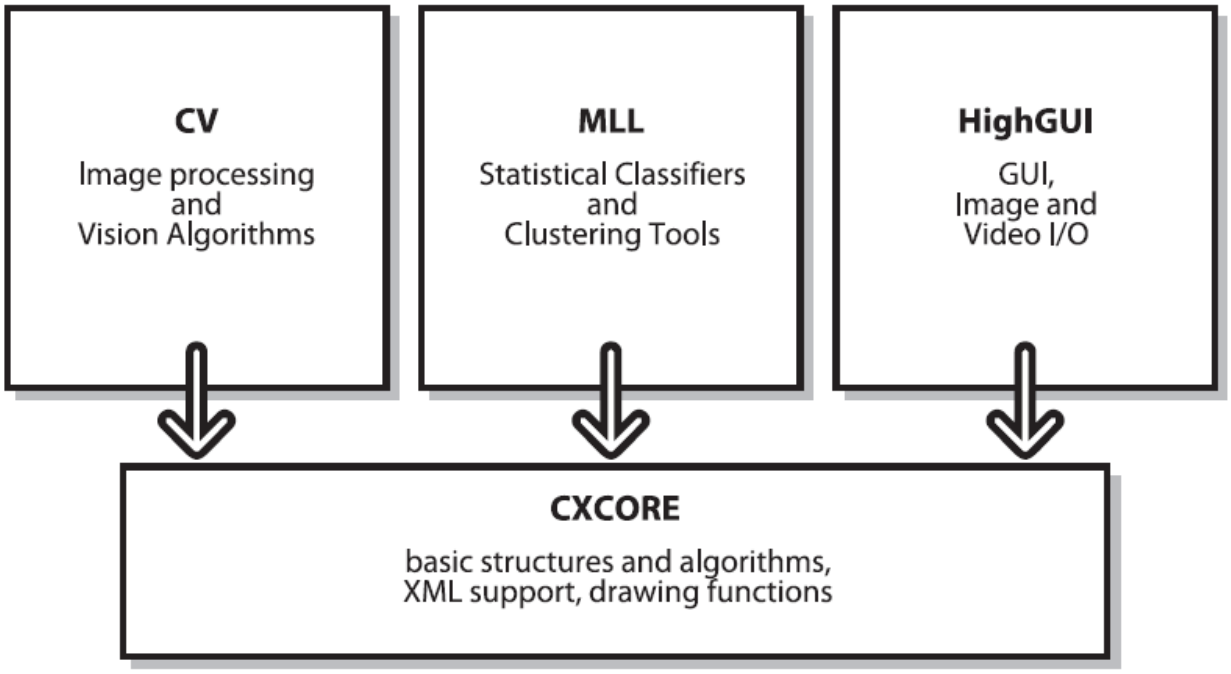
Computer vision is the transforming of data from a still, or video camera into either a representation or a new decision. All such transformations are performed to achieve a particular goal. A computer obtains a grid of numbers from a camera or from the disk, and that's that. Usually, there is no built 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. [6]

2.4 The Origin of OpenCV

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, a new student may start by adding to that which came before. [6]

2.5 OpenCV Structure and Content

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. [6]



2.6 Why OpenCV?

Specific

OpenCV was designed for image processing. Every function and data structure has been designed with an Image Processing application in mind. Meanwhile, Matlab, is quite generic. You can get almost everything in the world by means of toolboxes. It may be financial toolboxes or specialized DNA toolboxes.

Speedy

Matlab is just way too slow. Matlab itself was built upon Java. Also Java was built upon C. So when we run a Matlab program, our computer gets busy trying to interpret and compile all that complicated Matlab code. Then it is turned into Java, and finally executes the code.

If we use C/C++, we don't waste all that time. We directly provide machine language code to the computer, and it gets executed. So ultimately we get more image processing, and not more interpreting.

After doing some real time image processing with both Matlab and OpenCV, we usually got very low speeds, a maximum of about 4-5 frames being processed per second with Matlab. With OpenCV however, we get actual real time processing at around 30 frames being processed per second.

Sure we pay the price for speed – a more cryptic language to deal with, but it's definitely worth it. We can do a lot more, like perform some really complex mathematics on images using C and still get away with good enough speeds for your application.

Efficient

Matlab uses just way too much system resources. With OpenCV, we can get away with as little as 10mb RAM for a real-time application. Although with today's computers, the RAM factor isn't a big thing to be worried about. However, our drowsiness detection system is to be used inside a car in a way that is non-intrusive and small; so a low processing requirement is vital.

Thus we can see how OpenCV is a better choice than Matlab for a real-time drowsiness detection system.

Chapter 3

Machine Learning

3.1 What Is Machine Learning

The goal of **machine learning** is to turn data into information. After having learned from a gathering of data, we want a machine that is able to answer any question about the data:

- What are the other data that are similar to given data? Is there a face in the image?
- What kind of ad will influence the user?
- There is usually a cost component, so the question may become:

  Of the many products that we can make the money from, which one will most likely be bought by the user if an ad is shown for it?

Machine learning converts data into information by detecting rules or patterns from that data. [6]

3.2 Training and Test Set

Machine learning works on data like temperature values or stock prices or color intensities, and in our case face and eye detection. The data is usually preprocessed into features. We might take a database containing 1,000 face images then perform an edge detector on all the faces, and then obtain features such as edge direction, edge intensity, also offset from the face center for every face. We may obtain up to 500 such values for every face or a feature vector of 500 entries. We may then use machine learning to construct some sort of model from the obtained data. If we want to see how the faces fall into various groups (narrow, wide, etc.), after that a clustering algorithm can be the preferred choice. In case we want to learn how to guess the age of a woman from the pattern of the edges that are detected on her face, then we can use a **classifier** algorithm. To reach our goals, machine learning algorithms can analyze our obtained features and hence adjust the weights, the thresholds, and all the other parameters for maximizing performance set by those goals. This method of parameter adjustment for meeting a goal is what is called learning.

It is very important to understand how efficiently machine learning methods can work. This might be a delicate task. Usually, we break up the input data set into a very large training set (i.e. 900 faces, in our project) and a relatively small test set (i.e. remainder 100 faces). Then we can run our classifier on the training set in order to learn for a age prediction model, data feature

vectors given. Once done, we can then test our age prediction classifier obtained on the remainder of the images left in the set.

The test set is not applied for training; also we don't allow the classifier to see the age labels of the test set. We then run the classifier on all of the 100 faces present in the test set and then record how well the ages predicted by the feature vector match the real ages. When the classifier performs poorly, we may try to add new features to the data or consider other types of classifiers. There are many types of classifiers and various algorithms used for training them.

When the classifier performs well, we have a possibly lucrative model that can be used on data in the actual world. This system may be used to set the performance of a video game according to age.

After the classifier has been setup, it sees faces that it could not see before and it makes decisions with regards to what it had learned during training. Finally, when we develop a classification system, we usually use a validation data set.

At times, testing the entire system at the finish is a big step to take. We usually want to change parameters in the way before preparing our classifier for the final testing. We may do this by splitting the initial 1,000-face data set into 3 parts: i.e. a training set consisting of 800 faces and also a validation set with 100 faces, as well as a test set containing 100 faces. While we're running across the training data, we can perform pretests on validation data in order to see our progress. If we are happy with our output for the validation set, we can run the classifier up on the test set for the final result. [6]

3.3 OpenCV ML Algorithms

The machine learning algorithms that are included in OpenCV are given as follows. All the algorithms are present in the *ML* library apart from Mahalanobis and K-means, which are present in *CVCORE*, and the algorithm of face detection, which is present in *CV*. [6]

Mahalanobis: It is a measure of distance that is responsible for the stretchiness of the data. We can divide out the covariance of the given data to find this out. In case of the covariance being the identity matrix (i.e. identical variance), this measure will be identical to the Euclidean distance. [6]

K-means: It is an unsupervised clustering algorithm which signifies a distribution of data w.r.t. *K* centers, *K* being chosen by the coder. The difference between K-means and expectation maximization is that in K-means the centers aren't Gaussian. Also the clusters formed look somewhat like soap bubbles, as centers compete to occupy the closest data points. All these cluster areas are usually used as a form of sparse histogram bin for representing the data. [6]

Normal/Naïve Bayes classifier: It is a generative classifier where features are often assumed to be of Gaussian distribution and also statistically independent from one another. This assumption is usually false. That's why it's usually known as a "naïve Bayes" classifier. That said, this method usually works surprisingly well. [6]

Decision trees: It is a discriminative classifier. The tree simply finds a single data feature and determines a threshold value of the current node which best divides the data into different classes. The data is broken into parts and the procedure is recursively repeated through the left as well as the right branches of the decision tree. Even if it is not the top performer, it's usually the first thing we try as it is fast and has a very high functionality. [6]

Boosting: It is a discriminative group of classifiers. In boosting, the final classification decision is made by taking into account the combined weighted classification decisions of the group of classifiers. We learn in training the group of classifiers one after the other. Each classifier present in the group is called a weak classifier. These weak classifiers are usually composed of single-variable decision trees known as "stumps". Learning its classification decisions from the given data and also learning a weight for its vote based on its accuracy on the data are things the decision tree learns during training. While each classifier is trained one after the other, the data points are re-weighted to make more attention be paid to the data points in which errors were made. This continues until the net error over the entire data set, obtained from the combined weighted vote of all the decision trees present, falls below a certain threshold. This algorithm is usually effective when a very large quantity of training data is available. [6]

Random trees: It is a discriminative forest of a lot of decision trees, each of which is built down to a maximal splitting depth. At the time of learning, every node of every tree is allowed a choice

of splitting variables, but only from a randomly generated subset of all the data features. This ensures that all the trees become statistically independent and a decision maker. In the run mode, all the trees get an unweighted vote. Random trees are usually quite effective. They can also perform regression by taking the average of the output numbers from every tree. [6]

Face detector (Haar classifier): It is an object detection application. It is based on a smart use of boosting. A trained frontal face detector is available with the OpenCV distribution. This works remarkably well. We can train the algorithm for other objects by using the software provided. This works wonderfully for rigid objects with characteristic views. [6]

Expectation maximization (EM): It is used for clustering. It is a generative unsupervised algorithm It fits *N* multidimensional Gaussians to the data, N being chosen by the user. It can act as an efficient way for representing a more complex distribution using only a few parameters (i.e. means and variances). Usually used in segmentation, it can be compared with K-means. [6]

K-nearest neighbors: It is one of the simplest discriminative classifier. The training data is simply stored using labels. Then, a test data point is classified in accordance to the majority vote of the K nearest data points. K-nearest neighbours is probably th simplest algorithm we can use. It is usually effective but it can be slow. It also requires a lot of memory. [6]

Neural networks or Multilayer perceptron (MLP): It is a discriminative algorithm which almost always contains hidden units in between the output and the input nodes for better representation of the input signal. It is slow to train, however it is quite fast to run. It remains the best performer for applications like letter recognition. [6]

Support vector machine (SVM): It is a discriminative classifier that is also capable of doing regression. Here, a distance function in between two data points is defined in a higher-dimensional space. (Projecting data onto higher dimensions helps in making the data more likely for linear separation.) Support vector machine (SVM) learns separating hyperplanes which maximally separate all the classes in the higher dimension. This tends to be the best when there

is limited data. However, when large data sets are available, boosting or random trees are preferred. [6]

3.4 Using Machine Learning in Vision

Usually, most algorithms take a data vector having many features as input. Here, the number of features may number in the thousands. If our task is recognizing a certain kind of object—take for example, a person's face. The first problem that we encounter is obtaining and labeling the training data which falls into positive (i.e. there is a face in the window) and negative (i.e. no face) cases. We soon realize that faces can appear at various scales: i.e. their image might consist of only a few pixels, or we might be looking at an ear which is filling the whole screen. Worse still, faces are usually occluded. We have to define what we actually mean when we say that a face is in the window. [6]

After having labeled the data that was obtained from various sources, we should decide which features we need to extract from these objects. Also, we must know what objects we are after. If the faces always appear upright, there is no reason for using rotation-invariant features and also no reason for trying to rotate the objects before processing.

In general, we must try to find features that express a little invariance in the objects. These can be scale-tolerant histograms of gradients or colors or even the popular SIFT features.

When we have requisite background window information, we can first remove it in order to help other objects stand out. Then we perform our image processing. This may consist of normalizing the image and then computing the various features. The resulting data vectors are all given the label that is associated with the object, action, or window. [6]

Once the data is obtained and converted into feature vectors, we break up the data into training sets, validation sets and test sets. It is advisable to do our learning, validation, and testing using a cross-validation framework. Here, the data is split into K subsets and we run various training (maybe validation) as well as test sessions. Each session consists of various sets of data that take on the roles of training (validation) and test. The test results obtained from these separate

sessions are used for averaging to get the final performance result. A more accurate picture of how the classifier performs when deployed in operation can be given by cross-validation.

Now that our data is ready, we must choose a classifier. Usually the choice of the classifier is determined by computational, data, and memory requirements. For certain applications, like online user preference modeling, we need to train the classifier quickly. In such a case, nearest neighbors, normal Bayes, or decision trees should be a good choice. When memory is the primary consideration, decision trees or neural networks are used for their space efficiency. When we have time to train our classifier but it needs to run quickly, neural networks can be a good choice, as with normal Bayes classifiers & support vector machines. When we have time to train but require high accuracy, then boosting and random trees are good choices. When we just want an easy and understandable check weather our features are chosen well or not, then decision trees or nearest neighbors should be used. For a good out of the box classification performance, boosting or random trees are tried. [6]

3.5 Variable Importance

This is the importance of a particular variable in a dataset. One of the uses of variable importance is for reducing the number of features the classifier needs to consider. After starting with a number of features, we train our classifier to find the importance of each feature in relation to all the other features. We then get rid of unimportant features. Eliminating unimportant features helps in improving speed performance (since it can eliminate the processing taken for computing those features) and also makes training and testing faster. When we don't have sufficient data, which is regularly the case, then eliminating unimportant variables helps in increasing classification accuracy, which in turn yields faster processing and better results. [6]

Breiman's algorithm for variable importance is as follows.

1. A classifier is trained on the training set.

2. A validation or test set is used for determining the accuracy of the classifier.

3. For each data point and a chosen feature, a new value for that feature is randomly chosen from among the values that the feature has in the remainder of the data set (known as "sampling with replacement"). This helps in ensuring that the distribution of that feature remains the same as in the original data set, however, the actual structure or meaning of that feature is removed (as its value is chosen at random from the remainder of the data).

4. The classifier is trained on the altered set of the training data and then the accuracy of classification measured on the changed test or validation data set. When randomizing of a feature hurt accuracy a lot, then it is implied that the feature is vital. When randomizing of a feature does not hurt accuracy much, then the feature is of little importance and is a suitable candidate for removal.

5. The original test or validation data set is restored and the next feature is tried until we are finished. The result obtained orders each feature by its importance. This procedure used above is built into random trees and decision trees and boosting. Thus, we can use these algorithms to decide which variables we will finally use as features; then we can use the optimized feature vectors to train the classifier. [6]

3.6 Tools of Machine Learning

There are a few basic tools which are used in machine learning to know the results. In supervised learning, one of the fundamental problems is knowing just how well the algorithm has performed: i.e. How accurate is it at fitting or classifying the data?

For real problems, we must take into account noise, fluctuations and errors in sampling, and so on. In other words, our test or validation data set may not reflect accurately the actual distribution of data. To come closer to predicting the actual performance of the classifier, we use the technique of *cross-validation* and/or the closely related technique of bootstrapping.

In its most rudimentary form, cross-validation involves the division of data into K different subsets. We train on $K - 1$ of these subsets and then test on the final subset of data (i.e. the "validation set") that we didn't train. We do this K times, in which each of the K subsets gets its turn at becoming the validation set. Then we average the results.

Bootstrapping is somewhat similar to cross-validation, but here the validation set is randomly selected from the training data. The selected points for the round are then used only for testing, not training. Then the process is started again from scratch. We do this *N* times. Each time we randomly select a new set of validation data and then average the results in the end. This means some and/or many of the data points are reused in various validation sets, hence the results are usually superior compared to cross-validation. [6]

There are two other very useful ways of assessing, tuning and characterizing classifiers. One is to plot the receiver operating characteristic (ROC) and the other is filling in a confusion matrix; see the figure. The ROC curve is used to measure the response of the performance parameter of the classifier through the full range of settings of that parameter. Here, the parameter is a threshold. Just to make this more relevant to our project, suppose we are want to recognize blue eyes in an image and we are using a threshold on the color blue as our detector. Setting the blue threshold extremely high would result in the classifier failing to recognize any blue eyes, thereby yielding a false positive rate of 0 but also having a true positive rate also at 0 (i.e. lower left part of the curve in the figure). On the other side, if the blue threshold is set to 0 then all signals count as a recognition. This means that all true positives (the blue eyes) are recognized including all the false positives (brown and green eyes); thus we end up having a false positive rate of 100% (i.e. up for every right part of the curve in the figure). The best possible ROC curve would be the one that follows the *y*-axis up to 100% and then cuts horizontally over to the up for every right corner. Failing that, the closer that the curve comes to the up for every left corner, the better. We can compute the fraction of area under the ROC curve versus the total area of the ROC plot. It can be a statistic of merit: The closer that the ratio is to 1, the better the classifier.

The given figure also shows a confusion matrix. It is just a chart of true and false positives along with true and false negatives. It is a quick way to evaluate the performance of a classifier. In an ideal case, we would see 100% along the NW-SE diagonal and 0% elsewhere. If we have a classifier that is capable of learning more than one class, then the confusion matrix is generalized for many classes and we just have to keep track of the class to which each labeled data point was assigned. [6]

3.7 Learning Objects

A trained classifier cascade stored in an XMLfile can be used by the cvLoad() function to load it and then cvHaarDetectObjects() to find objects similar to the ones it was trained on to train our own classifiers to detect other objects such as faces and eyes.

We do this with the OpenCV *haartraining* application, which creates a classifier from a training set of positive and negative samples. Steps are described below. [2][3]

3.7.1 Dataset gathering:-

Collect a data set consisting of examples of the object you want to learn These may be stored in one or more directories indexed by a text file in the following format:

<path>/img_1 count_1 x11 y11 w11 h11 x12 y12 . . .

<path>/img_2 count_2 x21 y21 w21 h21 x22 y22 . . .

Each of these lines contains the path (if any) and file name of the image containing the object(s). This is followed by the count of how many objects are in that image and then a list of rectangles containing the objects. The format of the rectangles is the *x*- and *y*-coordinates of the upper left corner followed by the width and height in pixels.

For optimum performance of the classifier you have to gather a lot of positive samples with very less unnecessary variance in data. The output of this process finally creates a file in the format.idx (e.g face.idx). [2][3]

3.7.2 Building vector output file:-

The utility application createsamples is used to build a vector output file of the positive samples. Using this file, you can repeat the training procedure below on many runs, trying different parameters while using the same vector output file. For example:

createsamples –vec faces.vec –info faces.idx –w 30 –h 40

This takes the *faces.idx* file and outputs a formatted training file, *faces.vec*. Then createsamples extracts the positive samples from the images before normalizing and resizing them to the specified width and height (here, 30-by-40). [2][3]

3.7.3 Object detection:-

The Viola-Jones cascade is a binary classifier: It simply 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 a negative sample. It is useful to take the negative images from the same type of data. That is, if we want to learn faces in online videos, for best results we should take our negative samples from comparable frames .However, comparable results can still be obtained using negative samples taken from any other source. Again we put the images into one or more directories and then make an index file consisting of a list of image filenames, one per line. For example, an image index file called background.idx. [2][3]

3.7.4 Training:-

Training call cn be initialized either by a batch file or using the following command lines in command prompt:

```
Haartraining /
  –data face_classifier /
  –vec faces.vec –w 30 –h 40 /
  –bg backgrounds.idx /
  –nstages 20 /
  –nsplits 1 /
  [–nonsym] /
  –minhitrate 0.998 /
  –maxfalsealarm 0.5
```

Now the resulting classifier will be stored in *face_classifier.xml*. Here faces.vec is the set of positive samples (sized to width-by-height = 30-by-40), and random images extracted from backgrounds.idx will be used as negative samples. The cascade is having 20 (-nstages) stages, where every stage is trained to have a detection rate (-minhitrate) of 0.998 or higher. The false hit rate (-maxfalsealarm) is set at 50% (or lower) each stage to allow for the overall hit rate of 0.998. The weak classifiers are specified in this case as "stumps", which means they can have

only one split (-nsplits);. For more complex object detection as many as six splits can be used, but mostly you no more than three splits are used.

Even on a fast machine, training may take a long time in the range of several hours to a day, depending on the size of the data samples. The training procedure must test approximately 100,000 features within the training window over all positive and negative samples. [2][3]

3.7.5 Creation of .xml files:-

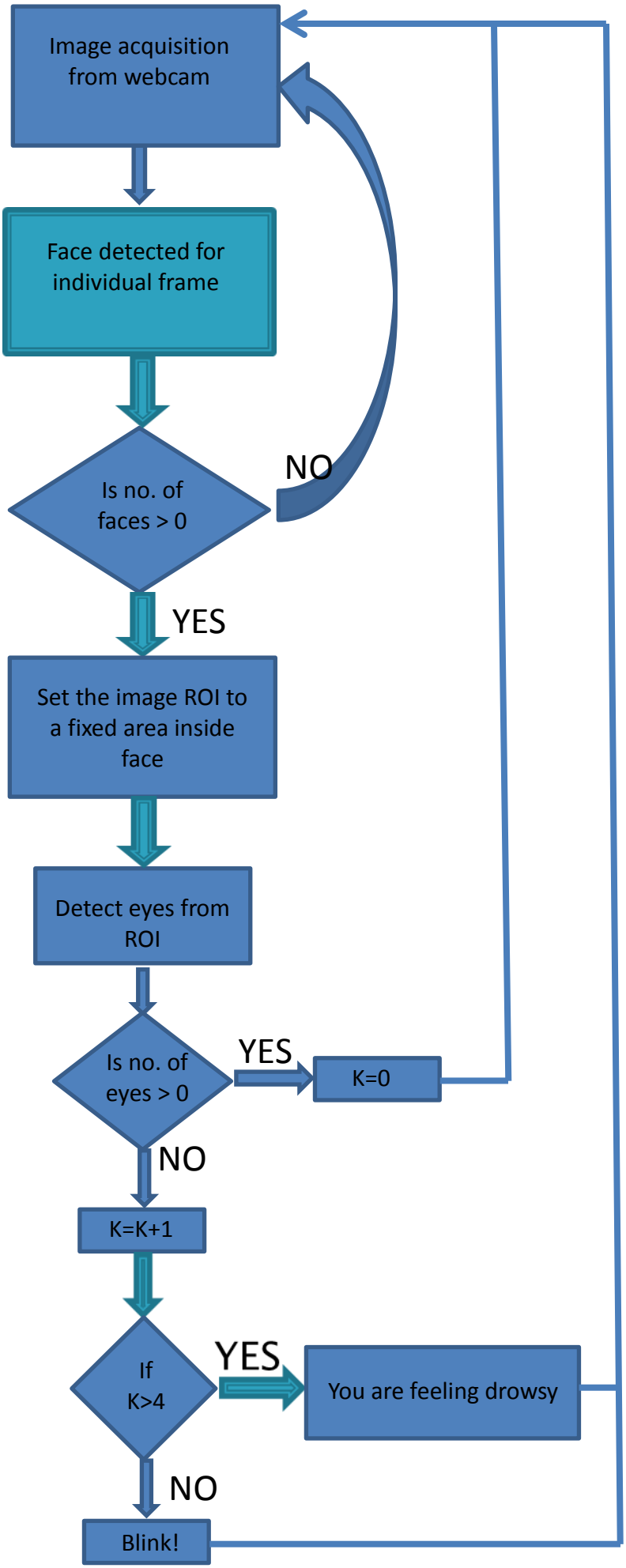
For our project face and eye classifiers are required. So we used the learning objects method to create our own haarclassifier .xml files. The Haarcascade files are one for the face and one for the eyes.

Around 1000 positive and 1500 negative samples were taken. It took a long time to train them. Finally face.xml and Haarcascade_eye.xml files are created.

These xml files are directly used for object detection using haardetectobjects() function. It detects a sequence of objects (in our case our 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. [2][3]

Chapter 4

Algorithm and Implementation



4.1 Algorithm

Drowsiness of a person can be measured by the extended period of time for which his/her eyes are in closed state. In our system, primary attention is given to the faster detection and processing of data.

The number of frames for which eyes are closed is monitored. If the number of frames exceeds a certain value, then a warning message is generated on the display showing that the driver is feeling drowsy.

In our algorithm, first the image is acquired by the webcam for processing. Then we use the Haarcascade file face.xml to search and detect the faces in each individual frame. If no face is detected then another frame is acquired. If a face is detected, then a region of interest in marked within the face. This region of interest contains the eyes. Defining a region of interest significantly reduces the computational requirements of the system. After that the eyes are detected from the region of interest by using Haarcascade_eye.xml.

If an eye is detected then there is no blink and the blink counter K is set to '0'. If the eyes are closed in a particular frame, then the blink counter is incremented and a blink is detected. When the eyes are closed for more than 4 frames then it is deducible that the driver is feeling drowsy. Hence drowsiness is detected and an alarm sounded. After that the whole process is repeated as long as the driver is driving the car.

4.2 Image acquisition:-

The function cvCaptureFromCAM allocates and initialized the CvCapture structure for reading a video stream from the camera.

CvCapture* cvCaptureFromCAM(int index);

Index of the camera to be used. If there is only one camera or it does not matter what camera to use , -1 may be passed.

cvSetCaptureProperty Sets camera properties For example

cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 280);

cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 220);

The function cvQueryFrame() grabs a frame from a camera or video file, decompresses it and returns it. This function is just a combination of *GrabFrame* and *RetrieveFrame*, but in one call. The returned image should not be released or modified by the user. In the event of an error, the return value may be NULL. [4][5][7]

4.3 Face detection:-

Cascade is loaded by:

```
cascade = (CvHaarClassifierCascade*)cvLoad( cascade_name,
0, 0, 0 );
```

storage is allocated:

```
CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* cvHaarDetectObjects(
const CvArr* image,
CvHaarClassifierCascade* cascade,
CvMemStorage* storage,
double scale_factor = 1.1,
int min_neighbors = 3,
int flags = 0,
```

CvSize min_size = cvSize(40,40)
);

CvArr image is a grayscale image. If region of interest (ROI) is set, then the function will respect that region. Thus, one way of speeding up face detection is to trim down the image boundaries using ROI. The classifier cascade is just the Haar feature cascade that we loaded with cvLoad() in the face detect code. The storage argument is an OpenCV "work buffer" for the algorithm; it is allocated with cvCreateMemStorage(0) in the face detection code and cleared for reuse with cvClearMemStorage(storage). The cvHaarDetectObjects() function scans the input image for faces at all scales. Setting the scale_factor parameter determines how big of a jump there is between each scale; setting this to a higher value means faster computation time at the cost of possible missed detections if the scaling misses faces of certain sizes. The min_neighbors parameter is a control for preventing false detection. Actual face locations in an image tend to get multiple "hits" in the same area because the surrounding pixels and scales often indicate a face. Setting this to the default (3) in the face detection code indicates that we will only decide a face is present in a location if there are at least three overlapping detections. The flags parameter has four valid settings, which (as usual) may be combined with the Boolean OR operator. The first is CV_HAAR_DO_CANNY_PRUNING. Setting flags to this value causes fl at regions (no lines) to be skipped by the classifier. The second possible flag is CV_HAAR_SCALE_IMAGE, which tells the algorithm to scale the image rather than the detector (this can yield some performance advantages in terms of how memory and cache are used). The next flag option, CV_HAAR_FIND_BIGGEST_OBJECT, tells OpenCV to return only the largest object found (hence the number of objects returned will be either one or none).* The final flag is CV_HAAR_DO_ROUGH_SEARCH, which is used only with CV_HAAR_FIND_BIGGEST_OBJECT. This flag is used to terminate the search at whatever scale the first candidate is found (with enough neighbors to be considered a "hit"). The final parameter, min_size, is the smallest region in which to search for a face. Setting this to a larger value will reduce computation at the cost of missing small faces.

CvRect *face = (CvRect*)cvGetSeqElem(face, 0);

The above fnction gets a sequence of objects from the image given and stores them as a rectangular region in the image.

```
cvRectangle(
img,

cvPoint(face->x, face->y),
cvPoint(
face->x + face->width,
face->y + face->height
),
CV_RGB(0, 255, 0),
1, 8, 0
)
```

The above function draws a rectangle in the image for the corresponding corner points .the other parameter are for drawing colour and thickness and type of lines in the rectangle. [4][5][7]

4.4 Set Image Region of interest:-

```
cvSetImageROI(
    img,                /* the source image */
    cvRect(
        face->x,        /* x = start from leftmost */
        face->y + (face->height)/5, /* y = a few pixels from the top */
        face->width,    /* width = same width with the face */
        (face->height)/3   /* height = 1/2 of face height */
    )
```

It sets the ROI for the corresponding image. We have taken from the left most point in the face to a few pixels from the top to half of face height. Width of the region is same as that of the face. The rectangular region is now used to get eyes. [4][5][7]

4.5 Eye Detection:-

```
CvSeq *eyes = cvHaarDetectObjects(
                        img,
                        cascade,
                        storage,
                        1.1,
                        5,
                        0 /*CV_HAAR_DO_CANNY_PRUNNING*/,
                        cvSize( 10, 5 ) );
```

It is the same function as that of the face detection .Here eye cascade is used and minimum size of the object is decreased and optimized to detect eyes of various sizes in the image.

As described earlier cvRectangle() is used to draw rectangle for sequences of eyes detected in a given frame .

then cvResetImageROI(img) is used to reset the ROI so that the whole image can be used to be displayed in a window using cvShowImage("video", img) function.

Since the cascade is constructed for only open eyes ,so when eyes are closed even for momentarily they are not detected. The closing state of eyes are detected as blink and if the closing state continues for more than 7 consecutive frames then drowsiness condition is displayed using cvPutText() function . We can also use batch file command system(“abc.mp3”) to play any audio file at the instant drowsiness is detected .

The detection process is continued until a key is pressed .When the key is pressed the program stops executing the detection function and stops capture form camera, releases memory and closes video window using the functions given below. [4][5][7]

```
cvReleaseCapture( &capture );
cvDestroyWindow( "video" );
cvReleaseMemStorage( &storage );
```

CHAPTER-5

RESULT

5.1 Summary:-

To obtain the result a large number of videos were taken and their accuracy in determining eye blinks and drowsiness was tested.

For this project we used a 5 megapixel USB webcam connected to the computer. The webcam had inbuilt white LEDs attached to it for providing better illumination. In real time scenario, infrared LEDs should be used instead of white LEDs so that the system is non-intrusive. An external speaker is used to produce alert sound output in order to wake up the driver when drowsiness exceeds a certain threshold.

The system was tested for different people in different ambient lighting conditions(daytime and nighttime). When the webcam backlight was turned ON and the face is kept at an optimum distance, then the system is able to detect blinks as well as drowsiness with more than 95% accuracy. This is a good result and can be implemented in real-time systems as well.

Sample outputs for various conditions in various images is given below. Two videos were taken; one in which only the eyes were detected and the other in which both face and eyes were detected. Though the two processes have relatively equal accuracy, the computational requirements of the former and lesser than that of the latter.

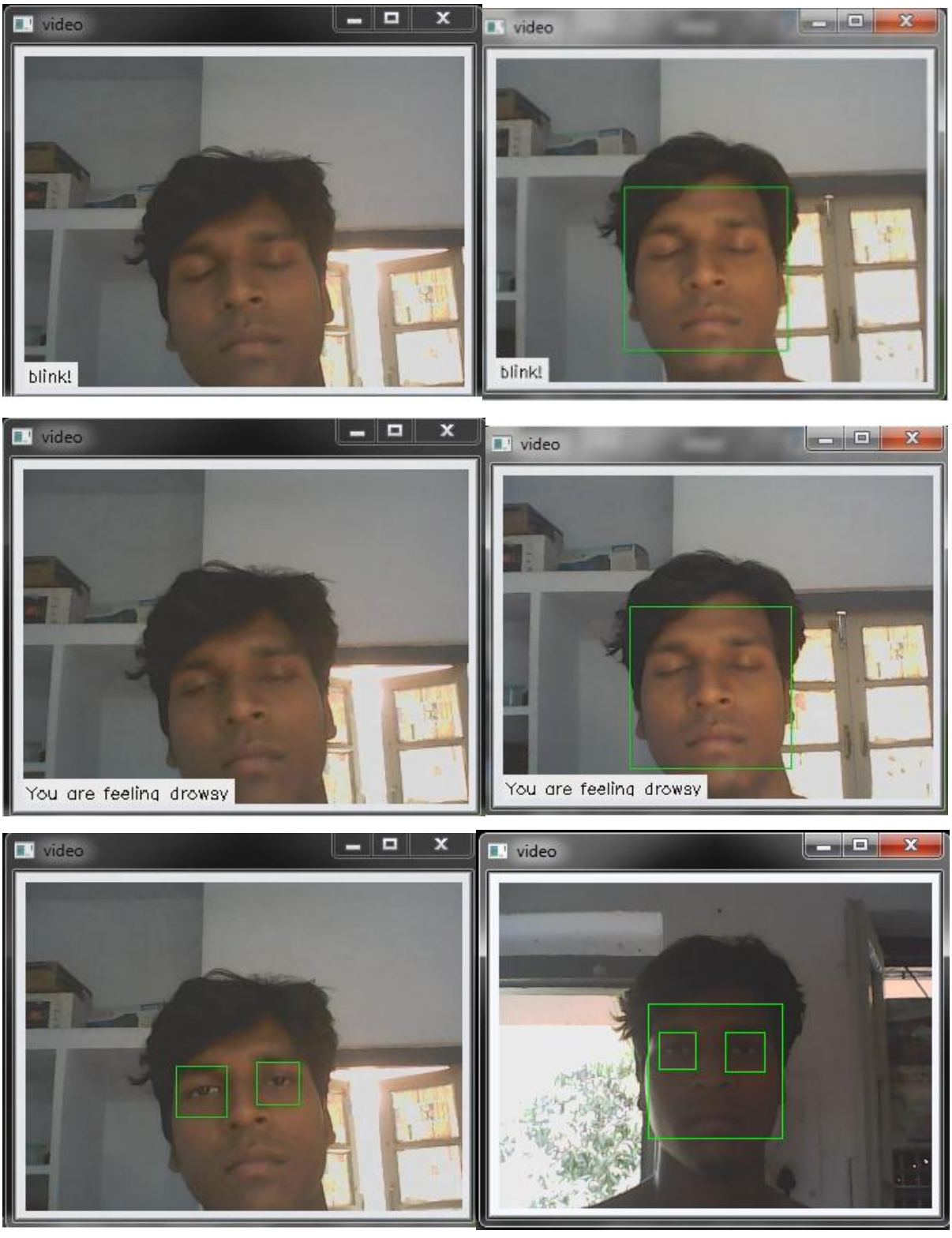
5.2 Sample Images:-

Sample 1

Only eyes detected face and eyes detected

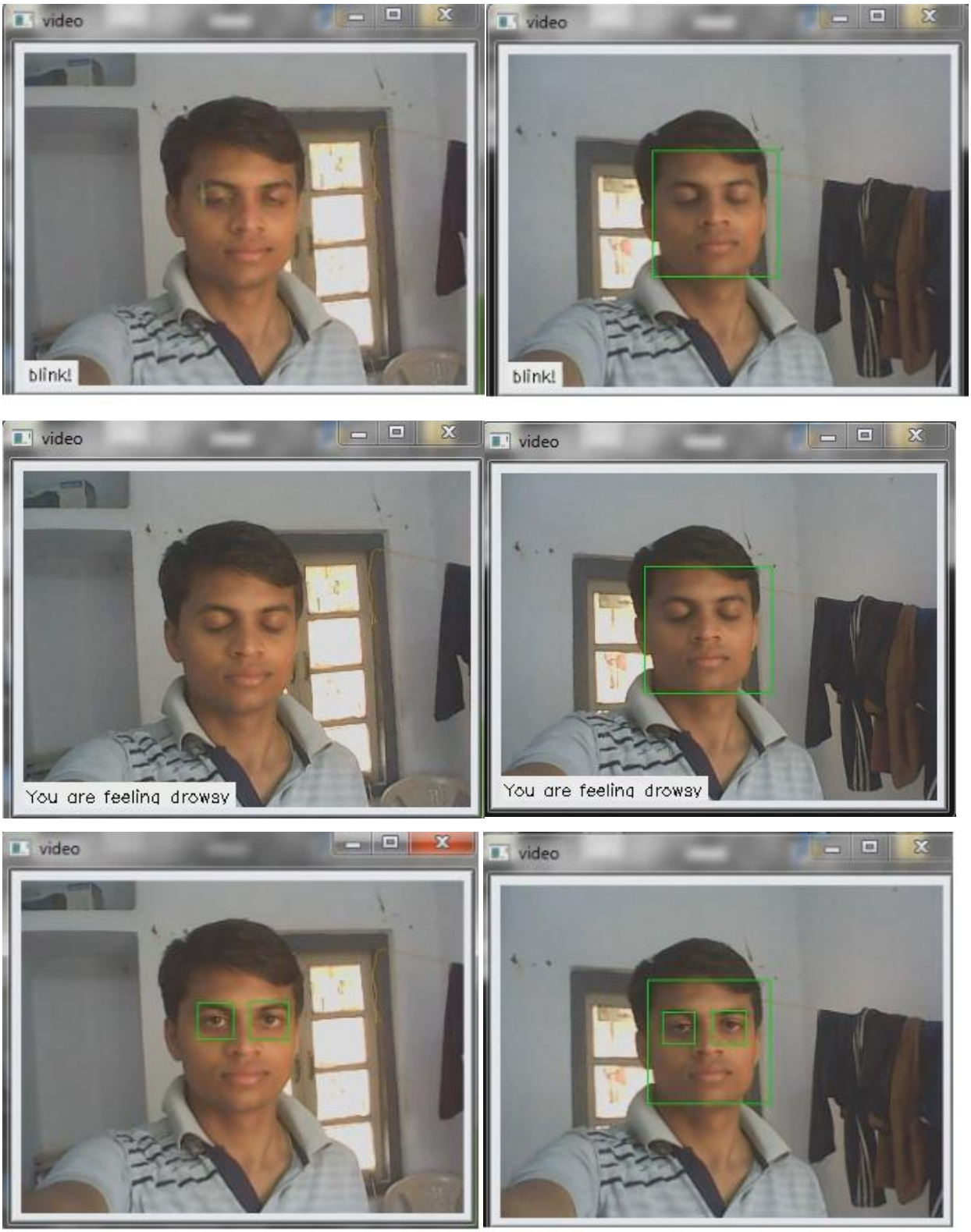
Sample 2

Only eyes detected face and eyes detected



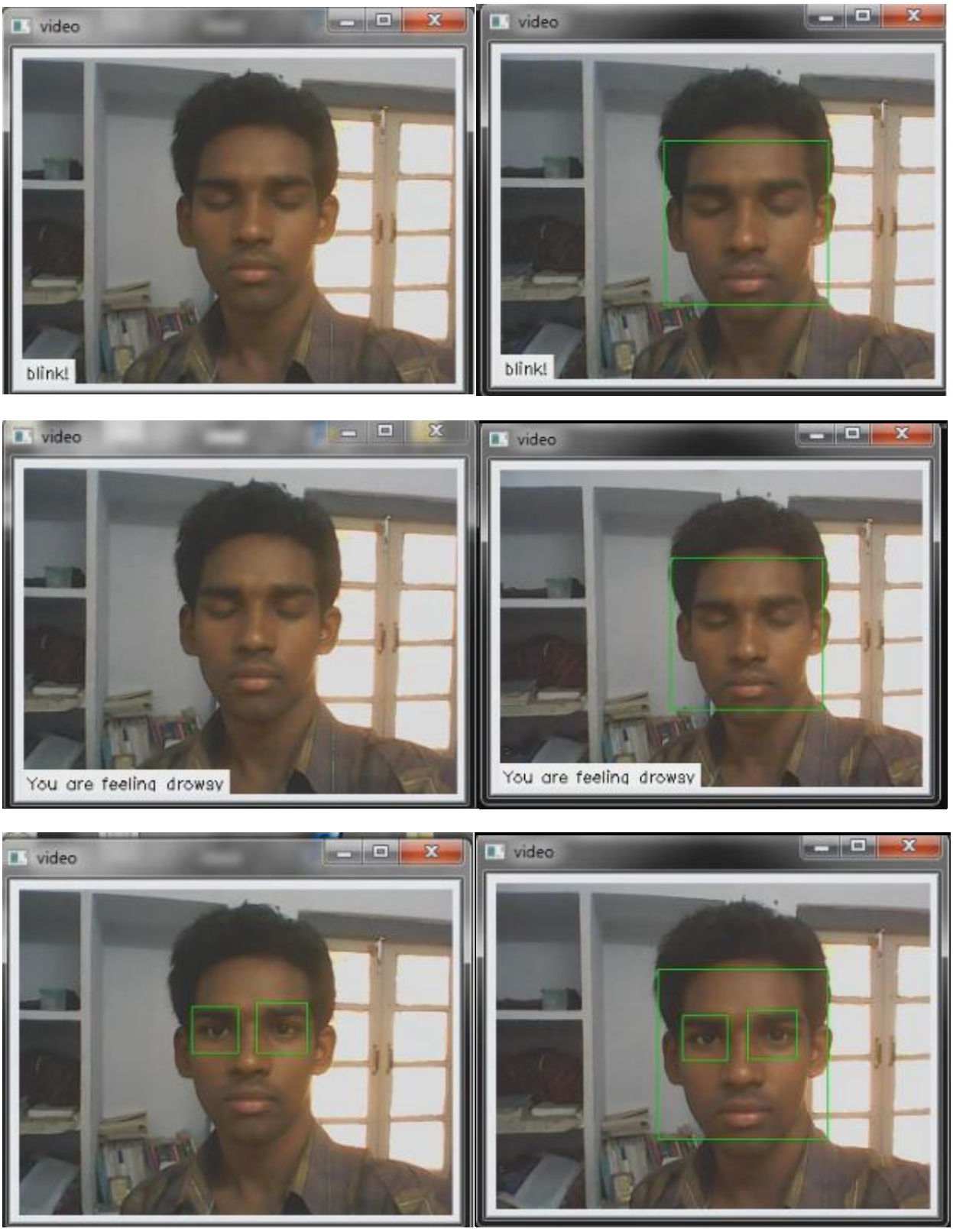
Sample 3

Only eyes detected face and eyes detected



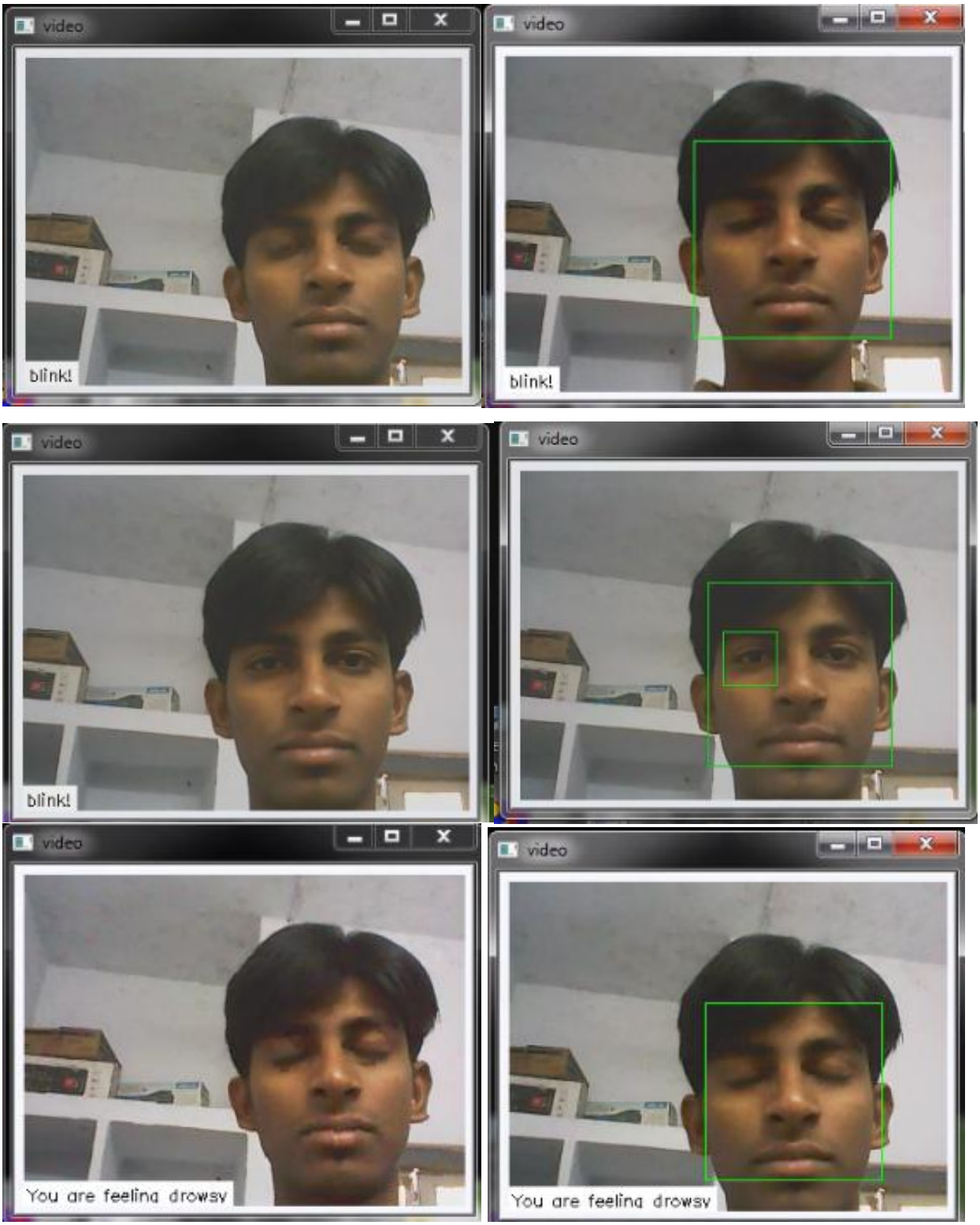
Sample 4

Only eyes detected face and eyes detected



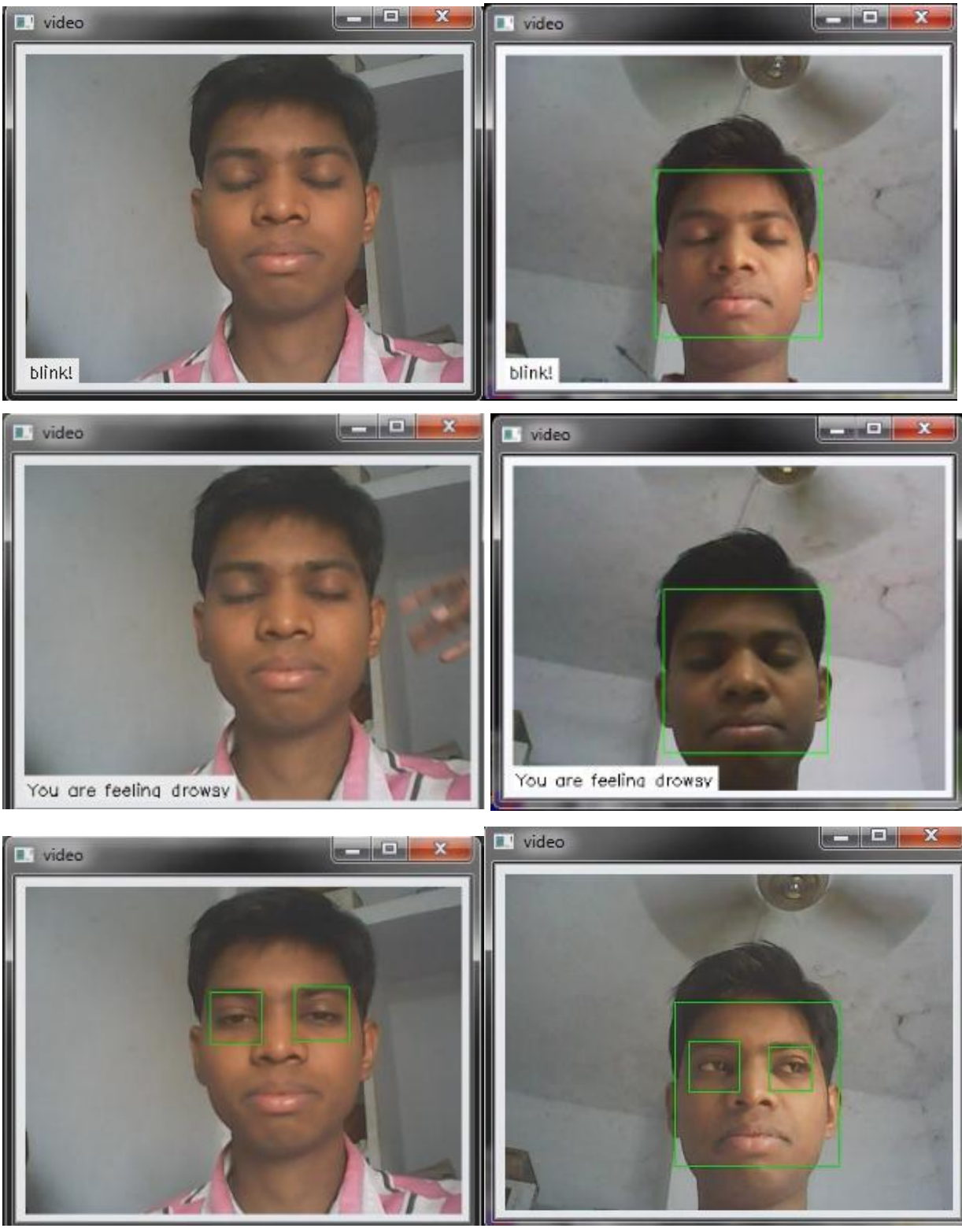
Sample 5

Only eyes detected face and eyes detected



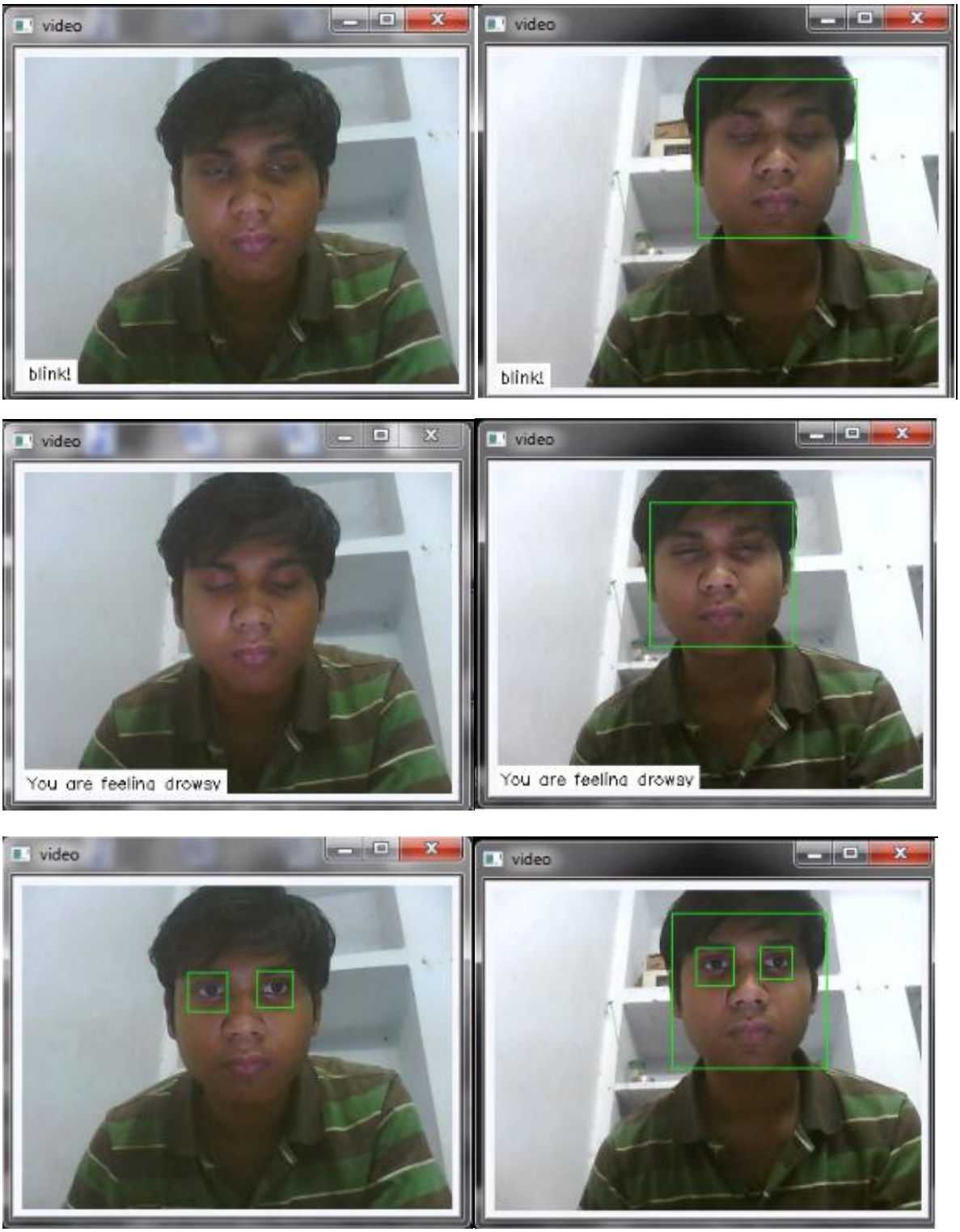
Sample 6

Only eyes detected face and eyes detected



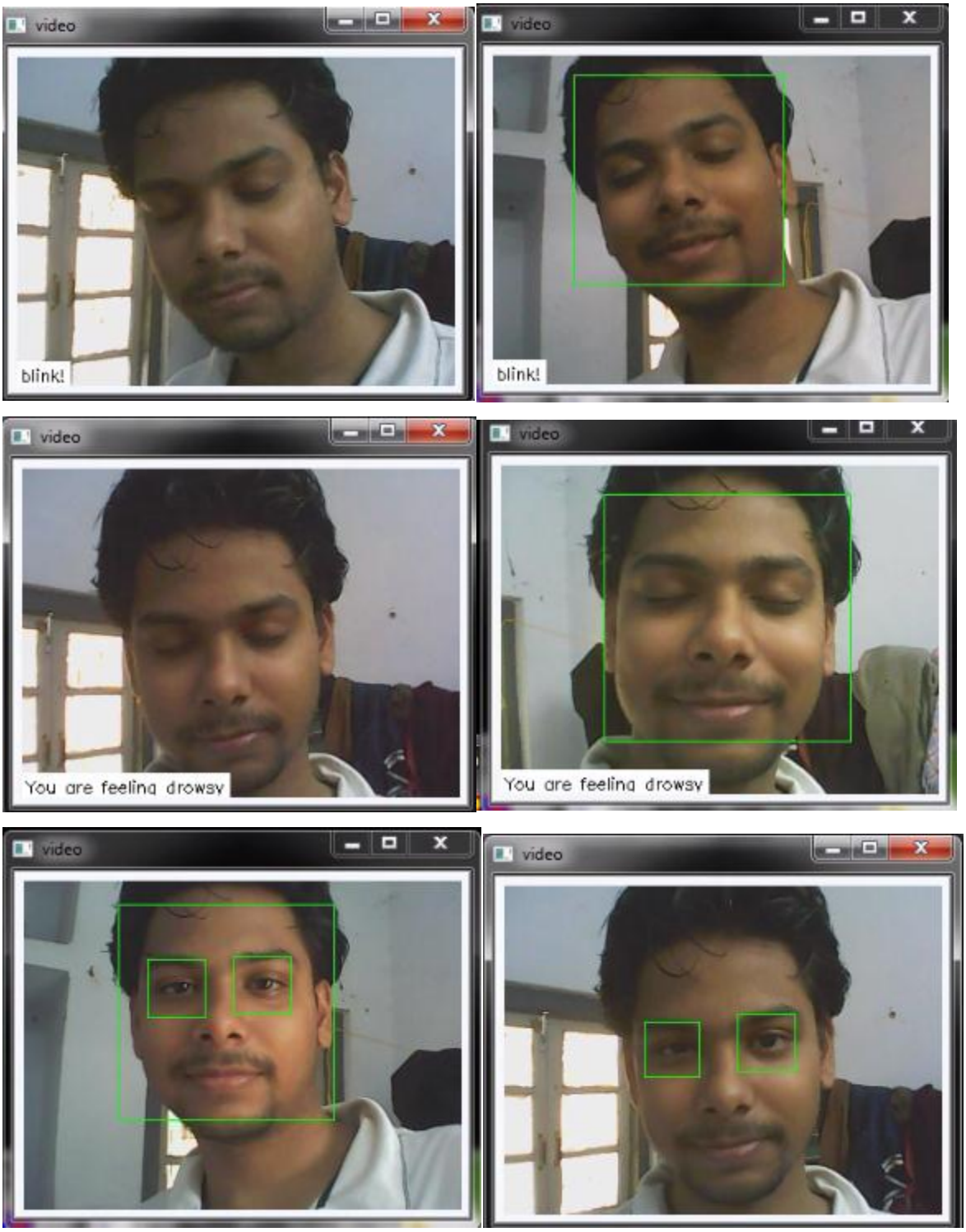
Sample 7

Only eyes detected face and eyes detected



Sample 8

Only eyes detected face and eyes detected



5.3 Table and Analysis:-

Various samples with various accuracies were taken and a table plotted for them.

Input	Eye blink accuracy for only eyes	Eye blink accuracy for face and eyes	Drowsiness accuracy for only eyes	Drowsiness accuracy for face and eyes
Sample 1	100%	100%	100%	100%
Sample 2	100%	95%	100%	100%
Sample 3	95%	95%	100%	100%
Sample 4	95%	100%	100%	100%
Sample 5	65%	45%	50%	33%
Sample 6	100%	100%	100%	100%
Sample 7	90%	95%	100%	100%
Sample 8	100%	100%	100%	100%
Total	93.125%	91.25%	93.75%	91.6%

Each volunteer was asked to blink 20 times and become drowsy 6 times during the testing process. The accuracy for eye blink was calculated by the formula

Accuracy = 1 - |total no. of blinks -no. of blinks detected| / total no. of blinks.

The same formula was used for calculating accuracy of drowsiness detection.

It can be seen from the above table that if sample 5 is not taken into consideration then the system has an accuracy of nearly 100%. That said; the high amount of errors in sample 5 shows that the system is prone to error and has certain limitations which we will discuss in the next section. In sample 5 we did not use the backlight of the webcam. The resulting poor lighting conditions gave a highly erroneous output.

5.4 Limitations:-

The limitations of the system are as follows.

1. **Dependence on ambient light:-** With poor lighting conditions even though face is easily detected, sometimes the system is unable to detect the eyes. So it gives an erroneous result which must be taken care of. In real time scenario infrared backlights should be used to avoid poor lighting conditions.
2. **Optimum range required:-** when the distance between face and webcam is not at optimum range then certain problems are arising.

   When face is too close to webcam(less than 30 cm) , then the system is unable to detect the face from the image. So it only shows the video as output as algorithm is designed so as to detect eyes from the face region.

   This can be resolved by detecting eyes directly using haardetectobjects functions from the complete image instead of the face region. So eyes can be monitored even if faces are not detected.

   When face is away from the webcam(more than 70cm) then the backlight is insufficient to illuminate the face properly. So eyes are not detected with high accuracy which shows error in detection of drowsiness.

   This issue is not seriously taken into account as in real time scenario the distance between drivers face and webcam doesn't exceed 50cm. so the problem never arises.

   Considering the above difficulties, the optimum distance range for drowsiness detection is set to 40-70 cm

3. **Hardware requirements:-** Our system was run in a PC with a configuration of 1.6GHz and 1GB RAM Pentium dual core processor.
   Though the system runs fine on higher configurations, when a system has an inferior configuration, the system may not be smooth and drowsiness detection will be slow.

The problem was resolved by using dedicated hardware in real time applications, so there are no issues of frame buffering or slower detection.

4. **Delay in sounding alarm:-** When drowsiness level exceeds a certain threshold, an alarm is produced by a system speaker. It requires a media player to run the audio file. There is a significant delay between when drowsiness is detected and when the media player starts and generates the alarm. But in real time, drowsiness is a continuous phenomenon rather than a one off occurrence. So the delay is not that problematic.
5. **Orientation of face:-** when the face is tilted to a certain extent it can be detected, but beyond this our system fails to detect the face. So when the face is not detected, eyes are also not detected.

   This problem is resolved by using tracking functions which track any movement and rotation of the objects in an image. A trained classifier for tilted face and tilted eyes can also be used to avoid this kind of problem.
6. **Poor detection with spectacles:-** When the driver wears glasses the system fails to detect eyes which is the most significant drawback of our system. This issue has not yet been resolved and is a challenge for almost all eye detection systems designed so far.
7. **Problem with multiple faces:-** If more than one face is detected by the webcam, then our system gives an erroneous result. This problem is not important as we want to detect the drowsiness of a single driver.

5.5 Future works:-

In the real time driver fatigue detection system it is required to slow down a vehicle automatically when fatigue level crosses a certain limit. Instead of threshold drowsiness level it is suggested to design a continuous scale driver fatigue detection system. It monitors the level of drowsiness continuously and when this level exceeds a certain value a signal is generated which controls the hydraulic braking system of the vehicle.

Hardware components required-

Dedicated hardware for image acquisition processing and display

Interface support with the hydraulic braking system which includes relay, timer, stepper motor and a linear actuator.

Function

When drowsiness level exceeds a certain limit then a signal is generated which is communicated to the relay through the parallel port(parallel data transfer required for faster results).The relay drives the on delay timer and this timer in turn runs the stepper motor for a definite time period .The stepper motor is connected to a linear actuator.

The linear actuator converts rotational movement of stepper motor to linear motion. This linear motion is used to drive a shaft which is directly connected to the hydraulic braking system of the vehicle. When the shaft moves it applies the brake and the vehicle speed decreases.

Since it brings the vehicle speed down to a controllable limit , the chances of accident occurrence is greatly reduced which is quite helpful for avoiding crashes caused by drowsiness related cases.

5.6 Conclusion:-

Thus we have successfully designed a prototype drowsiness detection system using OpenCV software and Haar Classifiers. The system so developed was successfully tested, its limitations identified and a future plan of action developed.

References

[1] http://www.ee.ryerson.ca/~phiscock/thesis/drowsy-detector/drowsy-detector.pdf

[2] http://www.cnblogs.com/skyseraph/archive/2011/02/24/1963765.html

[3] http://www.scribd.com/doc/15491045/Learning-OpenCV-Computer-Vision-with-the-OpenCV-Library

[4]http://opencv.willowgarage.com/documentation/reading_and_writing_images_and_video.html

[5] http://www.scribd.com/doc/46566105/opencv

[6] Learning OpenCV by Gary Bradski and Adrian Kaehler

[7] http://note.sonots.com/SciSoftware/haartraining.html