Acknowledgement

We would like to extend our sincere thanks to the Department of Computer Science and Engineering, National Institute of Technology Karnataka, and all those associated with it for allowing us to work on the afore-said project and supporting us in every way possible.

We would like to thank our guide, Mr K Vinay Kumar for his continued support and valued inputs throughout the project, without which this project would not have been successful.

We would like to thank our examiner, Dr. K C Shet, and Project Co-ordinator, Mr B R Chandavarkar for being a continuous source of inspiration throughout this past year.

We would like to thank our Project Lab in-charge, Miss Yashaswini, and M. Tech. Teaching Assistants.

Special thanks to everyone who helped us in making this project a success.

Rahul Agrawal

Rishabh M Gupta

Lokesh Kalal

Abstract

This Project aims at building an intelligent application which is capable of detection of vehicles/ cars and tracks them in the various frames of a real- time/ recorded video or image sequence. Once detected, various dynamics of the vehicle, like speed, could be calculated and road traffic can be analysed and recorded. The application will be capable of analysing the traffic density with respect to the time and plot various graphs for traffic analysis which can be used by road authorities as well as car companies.

The project involves two techniques for vehicle detection, recognition and tracking. The first is the blob based detection which detects moving vehicles on the road based on Pixel Based Motion Segmentation. The second is Haar classification and recognition. This method is based on machine learned decision tree to classify the vehicle as car, or other types based on how the classifier is trained.

Keywords: openCV [14], Haar [7], Histograms [3], Image Classifiers [9], Background Subtraction, Blob Tracking.

Contents

Nomenclature

1. List of Abbreviations

SIFT	Scale In-variant Feature Transform
SURF	Speed Up Robust Feature
HOG	Histogram of Oriented Gradients
SVM	Support Vector Machine
XML	Extended Mark-up Language
PBAS	Pixel Based Adaptive Segmentation

2. List of Symbols

∑	Summation

List of Figures

Chapter 1 Introduction

1.1 Image Processing and Object Detection

Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or a set of characteristics or parameters related to the image. Computer vision is considered to be a high-level image processing out of which a machine/software intends to decipher the physical contents of an image or a sequence of images (e.g., videos or 3D full-body magnetic resonance scans).

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. In our project, we focus on vehicular detection and tracking using these techniques.

1.2 Object Detection Tools: openCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing.

Features:

- Image transformations and manipulations
- Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, colour conversion, histograms, image pyramids).
- Structural analysis (connected components, contour processing, distance transform)
- Motion analysis (optical flow, motion segmentation, tracking).
- Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).

1.3 Object Classifiers

Image classification is perhaps the most important part of digital image analysis. It refers to a characterization tool which uses machine learning or neural networks for classification of pixels in a linear or multi-dimensional fashion. The two main classification methods are *Supervised Classification* and *Unsupervised Classification.* Supervised Classification refers to the method where Classifier is trained before-hand using positive and negative samples. Unsupervised learning does not use user training like its counter-method. Multi-classification means that classification is done into more than one type.

These are some of the classification techniques:

1. Decision Tree;
2. Rule-based classifiers;
3. SMO (SVM);
4. Naive Bayes;
5. Neural Networks.

1.4 Application Purpose

This Project aims at building an application which is capable of detection of vehicles/ cars and tracks these them in the various frames of a real- time/ recorded video or image sequence. Once detected, various dynamics of the vehicles could be calculated. The application will also be capable of analysing the traffic density with respect to the time and plot graphs depicting the same.

The main focus of this application is a robust detection scheme, which will be developed using advanced image processing algorithms like Blob tracking and Haar-like features. The implementation of these algorithms will be followed by training a Classifier will help in detection of the required vehicles in a real- time video.

1.5 Report Organization

The following segments describe the various surveys and implementations carried out in this project during the period of August 2012 to April 2013. Chapter 2 describes the Literature surveys carried out for learning various techniques, tools and algorithms used in the project. Chapter 3 describes in detail the project problem definition and objectives set in the project. Chapter 4 describes the theory, algorithms and actual implementation of the work done so far by us. Chapter 5 gives in detail the results obtained and its relevance. Chapter 6 concludes this report along with the future work in this project. References and Appendices follow.

Chapter 2 Literature Survey

The following segment discusses in detail the research papers and information from other sources which have been used in this project thus far. Excerpts from research papers mentioned in section 7- References which are relevant to building of this project have also been shown.

2.1 Haar-like feature based cascade classifier

Haar-like features are digital image features used in object recognition. They owe their name to their intuitive similarity with *Haar wavelets* and were used in the first real-time face detector. Working with only image intensities (i.e., the RGB pixel values at each and every pixel of image) makes the task of feature calculation computationally expensive.

Viola and Jones [9] adapted the idea of using Haar wavelets and developed the so called Haar-like features. A Haar-like feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image.

First, a classifier (namely a cascade of boosted classifiers working with Haar-like features) is trained with a few hundred sample views of a particular object (i.e., a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size.

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. ***The classifier outputs a "1" if the region is likely to show the object (ie.car), and "0" otherwise***. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more

efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales.

The word “cascade” in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word “boosted” means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifers, and are calculated as described below. The current algorithm uses the following Haar-like features:

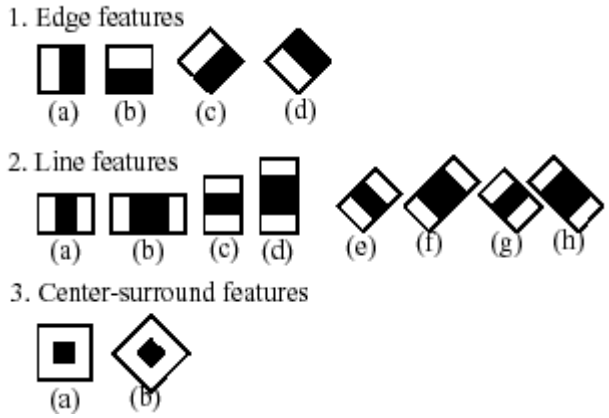

Fig 2.1: Haar Features; these are the features which the classifier uses

The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position within the region of interest and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied). For example, in the case of the third line feature (2c) the response is calculated as the difference between the sum of image pixels under the rectangle covering the whole feature (including the two white stripes and the black stripe in the middle) and the sum of the image pixels under the black stripe multiplied by 3 in order to compensate for the differences in the size of areas. The sums of pixel values over a rectangular region are calculated rapidly using integral images.

The key advantage of a Haar-like feature over most other features is its calculation speed. Due to the use of integral images, a Haar-like feature of any size can be calculated in constant time (approximately 60 microprocessor instructions for a 2-rectangle feature).

2.2 Blob Detection

Blob detection refers to mathematical methods that are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to areas surrounding those regions. Informally, a blob is a region of a digital image in which some properties are constant or vary within a prescribed range of values; all the points in a blob can be considered in some sense to be similar to each other.

One main reason to use this technique is to provide complementary information about regions, which is not obtained from edge detectors or corner detectors. In early work in the area, blob detection was used to obtain regions of interest for further processing. These regions could signal the presence of objects or parts of objects in the image domain with application to object recognition and/or object tracking. In other domains, such as histogram analysis, blob descriptors can also be used for peak detection with application to segmentation. Another common use of blob descriptors is as main primitives for texture analysis and texture recognition. In more recent work, blob descriptors have found increasingly popular use as interest points for wide baseline stereo matching and to signal the presence of informative image features for appearance-based object recognition based on local image statistics. There is also the related notion of ridge detection to signal the presence of elongated objects.

2.3 Histogram of Oriented Gradients

A *histogram* is an array of numbers in which each element (bin) corresponds to the frequency of a range of values in the given data. For instance, each bin counts the

number of pixels having the same colour values in case of an image histogram. Thus, a histogram is a mapping from the set of data values to the set of non-negative real numbers. Histograms are among the most common features used in many computer vision tasks from object based retrieval, to segmentation, to detection, to tracking.

Computational complexity is one major bottleneck of the histogram extraction and comparison based search tasks. It is obvious that only an exhaustive search can provide the global optimum. Although several sub-optimal techniques that are powered by gradient descent methods and application specific constraints have been developed to deliver accelerated alternatives to the basic exhaustive search, computer vision problems that rely on the optimal solutions, such as detection and tracking, still demand a theoretical breakthrough in histogram extraction as much as an powerful computers to crunch the numbers.

To address the computational requirements of detection tasks, a fast method to compute histograms of all possible target regions in a given data is discussed. We took advantage of the spatial positioning of data points in a Cartesian coordinate system, and propagate an aggregated function, which is referred to as the integral histogram [2], starting from an origin point and traversing through the remaining points along a scan-line. We iterate the integral histogram at the current point using the histograms of the previously processed neighbouring data points. At each step, we increase the value of the bin that the current point fits into the bin's range. After the integral histogram is obtained for each data point, histograms of target regions can be computed easily by using the integral histogram values at the corner points of those regions without reconstructing a separate histogram for every single region. Algorithm 3 gives a pseudo implementation of the above technique.

A conventional approach of measuring distances between a given histogram and histograms of all possible target regions is an exhaustive search. This process requires generation of histograms for the regions cantered at every possible point.

To perform histogram comparison, we first generate an integral histogram by propagation, and then compute the histograms of target regions by intersection. There are different scanning and propagation approaches; here we present two of them. One is a string scan method that covers the data space along each dimension e.g. from left to right and top to bottom for an image data. The integral histogram at the current point is obtained by copying the previous values and increasing the corresponding bin with respect to the current value of the point. The string scan requires only update at each step of propagation.

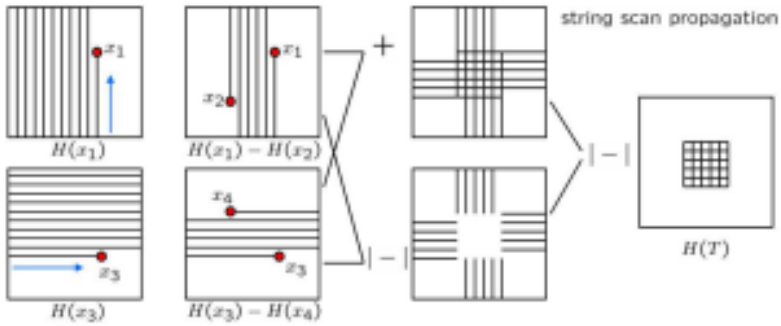

Fig 2.2: Propagation of integral histogram by string scans

The integral histogram at a point is obtained by three arithmetic operations for each bin of using the integral histogram values of the three neighbours as shown in Fig. 2.

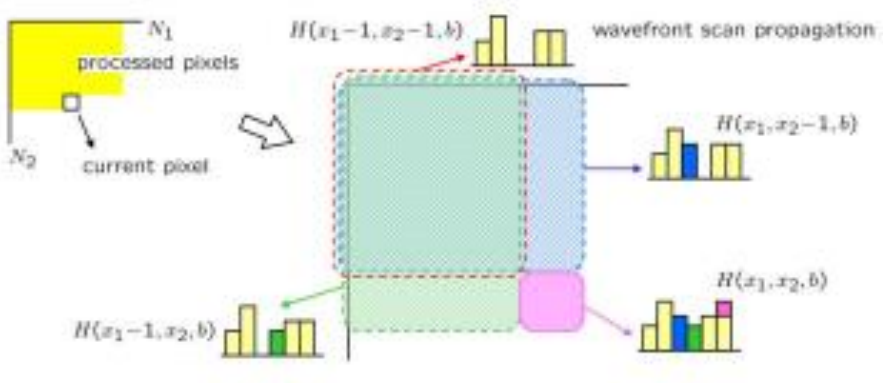

Fig 2.3: Propagation of Integral Histogram by wave-front scan. Grey indicates already traversed points. At each step, the current integral histogram is obtained from the integral histogram values of the three neighbours, and the bin that corresponds to current point's value is increased by one.

2.4 Support Vector Machines (SVM)

SVMs are machine learning techniques which use supervised learning as a means to find the k-th nearest neighbour for an input from given cluster set of data.

Linear Support Vector Machines (SVMs) are used for such high-dimensional and sparse data. Linear SVMs provide good prediction accuracy. While conventional training methods for linear SVMs, in particular decomposition methods like SVM-Light[5], LIBSVM, and SVM-Torch handle problems with a large number of features N quite efficiently, their super-linear scaling behaviour with n makes their use inefficient or even intractable on large datasets. On the other hand, while there are training methods that scale linear in, such methods empirically (or at least in the worst case) scale quadratic-ally with the number of features N.

A multi-dimensional classifier will be further used to identify and classify the positive region of interests into more than two buckets. This will enable us to classify Heavy vehicles or light vehicles separately. This is important because it will be used as the basic building block for the traffic analysis.

2.5 Multi-classification and Multi-classifiers

Combined classifiers are used in multiple classifier source applications like different feature spaces, different training sets, different classifiers applied for example to the same feature space, and different parameter values for the classifiers, e.g. k in k-nearest neighbour (k-NN) classifiers. In this approach, the features obtained from each scale are applied to a base classifier.

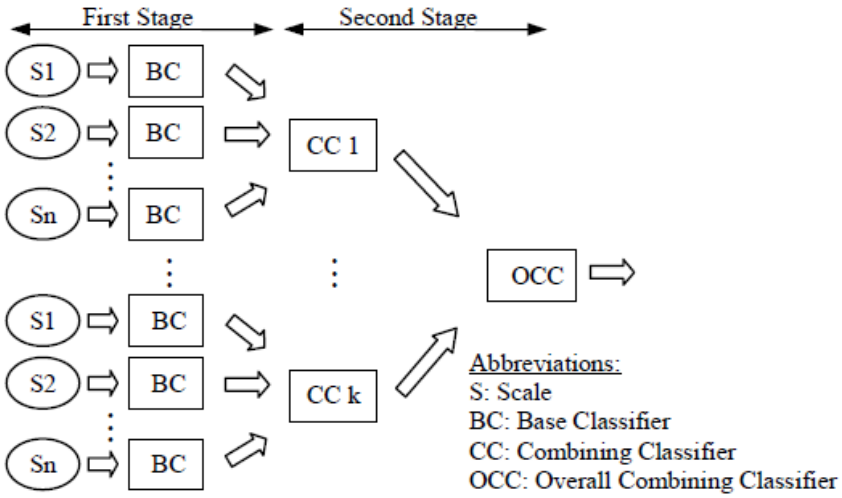

Fig 2.4: Multi-classification using basic classifiers

As the features spaces generated from different scales/derivatives are different in our method, parallel combined classifiers seem to be natural choice. Use of combined classifiers lifts the requirement for feature space fusion. Moreover, each base classifier can be examined separately, e.g. by drawing the corresponding learning curve, to determine the contribution of each scale/derivative towards the overall performance of the classification algorithm as will be shown in section 4. Here, we propose a two-stage parallel combined classifier with the structure shown above. At the output of this stage a fixed combining rule is applied to combine the outputs of different scales for one particular order of derivative. For example scales s1, s2 and s3 for the $\boldsymbol{Lx}$ are combined using a fixed combining rule. In the second stage, the outputs of the first stage, i.e. all different derivatives, are combined to produce the overall decision on the texture classes.

This two-stage parallel combined classifier provides a versatile tool to examine the contribution of each derivative in the overall performance of the classifier. However, it must be noticed that if two combining rules are the same, e.g. if both are fixed 'mean combining' rule, this two stage combined classifier is not different from one stage where all the feature spaces are applied to the base classifiers and the outputs are combined using the same combining rule. Two parameters have to be selected in this combined classifier: the type of the base classifier and the combining rule.

Chapter 3 Problem Definition and Objectives

The following segment describes the problem definition and objectives that are set as a part of the current project with consultation from the Guide.

3.1 Problem definition

Designing and developing an application which detects and recognizes vehicles or a set of vehicles and tracking them in various frames of a video or image sequence in real-time. After tracking, analysis of traffic density with respect to time will be done which will yield statistical data that can be used at traffic junctions for smart traffic light controllers or by the road development authorities.

3.2 Objectives

1. Detect the moving objects on the road. Exclude small instances like people or cycles. This has been done by the use of Blob based detection.
2. Train the Classifier with features corresponding to positive and negative images of the sample space. Recognize the vehicles using Haar classification.
3. Track the vehicles and obtain a good frame rate in real time processing.
4. Use the above developed frame-work for multi classification of traffic. Use various techniques like serial and parallel classification.
5. Analysis of traffic density- plotting of graphs and storage for future reference.

The final work, in the form of an application can be installed on any compatible system and executed. A webcam or IP camera can be used for real time input. Recorded video files can also be analyzed with this application.

4. Methodology/ Work Done

The following section gives the detailed description of the work done so far and describes the algorithms which form the building block of this project

4.1 Phases of the Project and Basic Working Algorithm

The project has been divided into the following phases as these form independent working modules of the project

Phase 1- Vehicular detection and tracking using Linear Classifier
Phase 2- Introducing Multi-Classification in the above frame-work
Phase 3- Traffic Analysis and Inference sub-system development

This Phase and its sub- phases are discussed in this report in later sections. The basic working algorithm of this project is discussed below.

ALGORITHM 1: Training Algorithm

STEP 1: *Store Positive and negative images in separate folders*
STEP 2: *Load image from dataset*
STEP 3: *Calculate Feature vectors for the image*
STEP 4: *Train Classifier with the features*
GOTO STEP 2 until dataset exhausts
// Steps 1 through 4 take place once and generate a training XML file

ALGORITHM 2: Basic Working Algorithm

STEP 1: *Load a frame from recorded or live video.*
STEP 2: *Use Blob detection to detect moving objects*
STEP 3: *Use Haar Classifier to classify the object as vehicles.*
STEP 4: *Use Vehicle counting module for traffic analysis.*
STEP 5: *Detected objects are boxed for output.*
If a key (Quit) is pressed then
GOTO STEP 6
Else
GOTO STEP 1
STEP 9: *Release image frames and destroy tracking windows.*

4.2 Requirements Analysis and Specification

The following segment describes the various requirements of the project from the view point of both, the user and as the application developer.

4.2.1 Requirements Analysis

1. Input by user/ administrator: During the training period, a large set of images (both positive and negative images) are to be input to the SVM/ learning machine in order to train them. The end- user will give a real- time video as input to the application.

2. Output: The Output will be in the form of detected objects being boxed in the video. The analysis data is to be stored in data file and graphs plotted so that useful inference can be drawn.

3. The object to be detected is a 3-D object – here vehicles: This project is aimed to detect vehicles in particular, which is an unsymmetrical 3D object.

4. User Interface Requirements: A simple Graphical User Interface.

5. Hardware Requirements analysis: The hardware that is required for this project is a webcam which would be used to grab image sequence and transfer it to application.

6. Purpose/ Usage analysis: This application can be used in monitoring the traffic. It can be used to find the vehicles that offend the traffic rules. On a higher-level, it can be used as a tool to analyse the traffic density for road strength and pollution issues.

4.2.2 Requirements Specification

1. Acquisition: Webcam or USB camera to take input video
2. Processing: OpenCV 2.2.0 libraries (or higher) for processing/detecting the object in the frames

3. Video Fragmenter: FFMPEG 4:0.6.2-1, Software used for splitting frames of streamed video (To be used in initial training phases).

4. Platform specification: The application should be compatible with Linux operating systems such as Ubuntu as well as Windows. The library files including openCV and standard libraries should be preinstalled on the system

5. Hardware specifications:
   a. A USB Webcam compatible to the system
   b. System with USB (Universal Serial Bus) slot.

6. System specification:
   a. Processor: 2 GHz +, Dual Core preferred (due to the high operational needs).
   b. Operating system: Linux OS or Windows OS.

4.3 Application Architecture

The flow charts given below describe the working of the program in brief. Note that this is a single classification program.

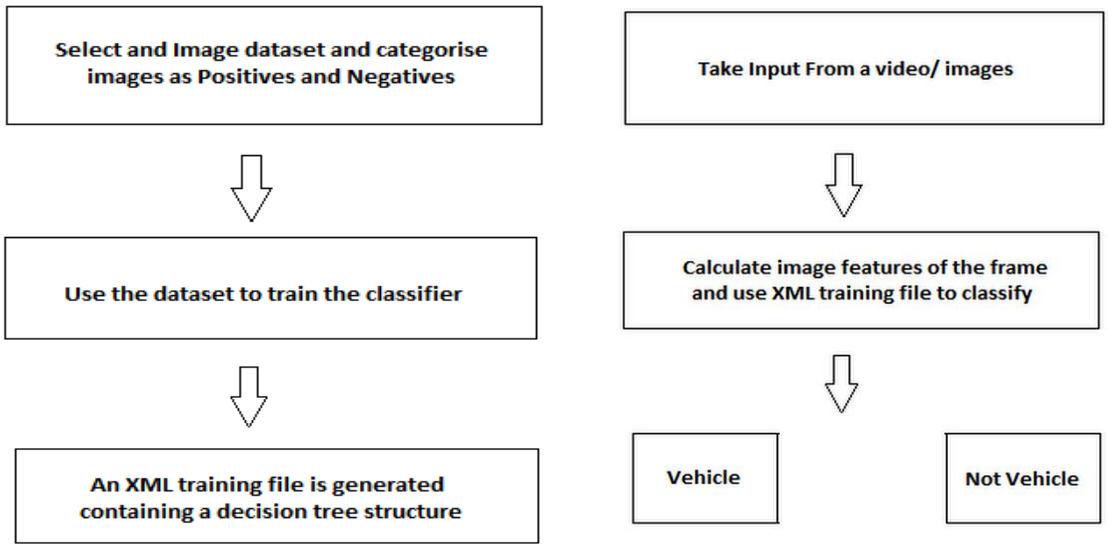

Fig 4.1: Flow-charts depicting the TRAINING and CLASSIFY stages

The learning images containing both positive and negative samples in the data set are used to train the Classifier. So this can be referred to as TRAINING/ LEARNING module. The Second module known as CLASSIFY is used to take the actual input image from the live/ real- time video feed and detect if the object (for which the machine is trained) is present or not in the input image.

4.4 Vehicle Detection

The XML file generated after training the classifier is used for classification in the actual/ real time feed when the main program is under execution.

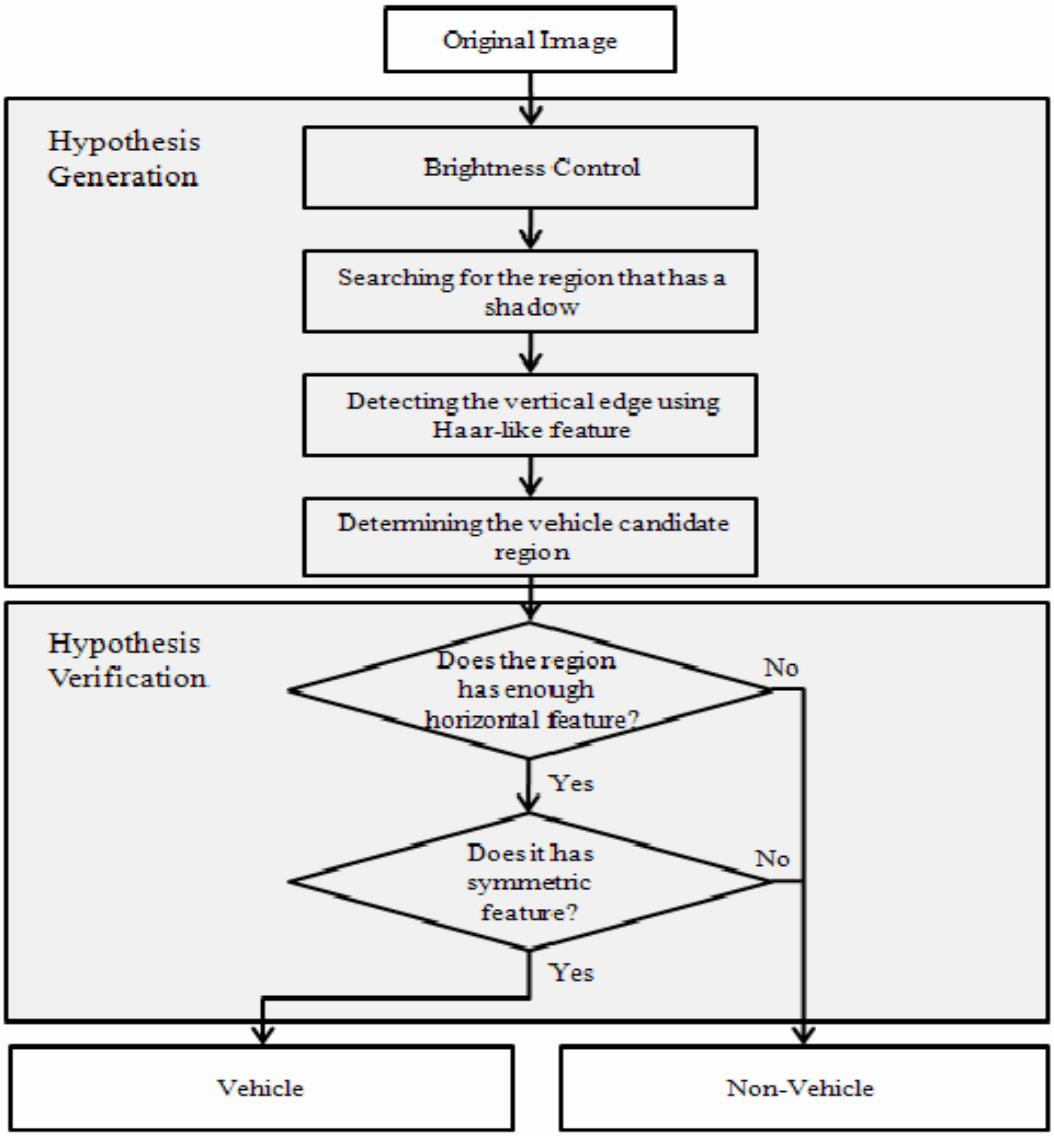

Fig 4.2: Flow-chart depicting actual vehicle detection

4.5 Multi-classification

The cascade classifiers have to be used to classify the vehicles into various types. Let us say that XML 1 is the training file for one type of vehicles, say Type A, and XML 2 denote the training file for another type of classifier, say type B. We propose two different methods for multi-classification of vehicles.

The first method is Serial Multi-classification. Here, two or more classifiers are arranged in a serial manner to classify the objects. The diagram given below explains the classification in detail.

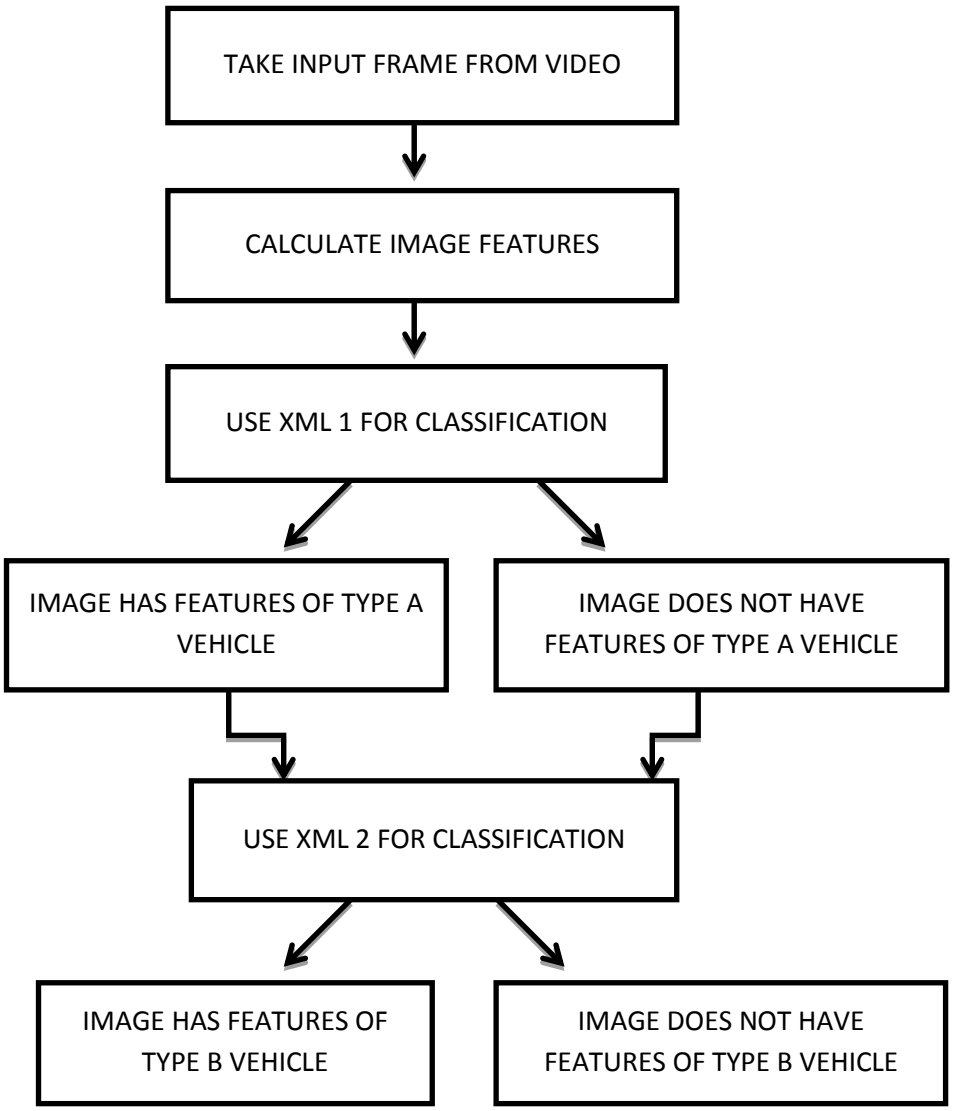

Fig 4.3: Flow-charts depicting serial multi-classification

Another approach is Parallel Multi-classification. Here, we create two or more threads/ child processes which separately classify the image. The positive detections can then be combined to generate the given output.

It is difficult to analyse which of the above method is better before implementation, as division and combination of threads is a costly process if the number of classifications is small. So, both the methods will be implemented and the better process will be chosen to continue further work.

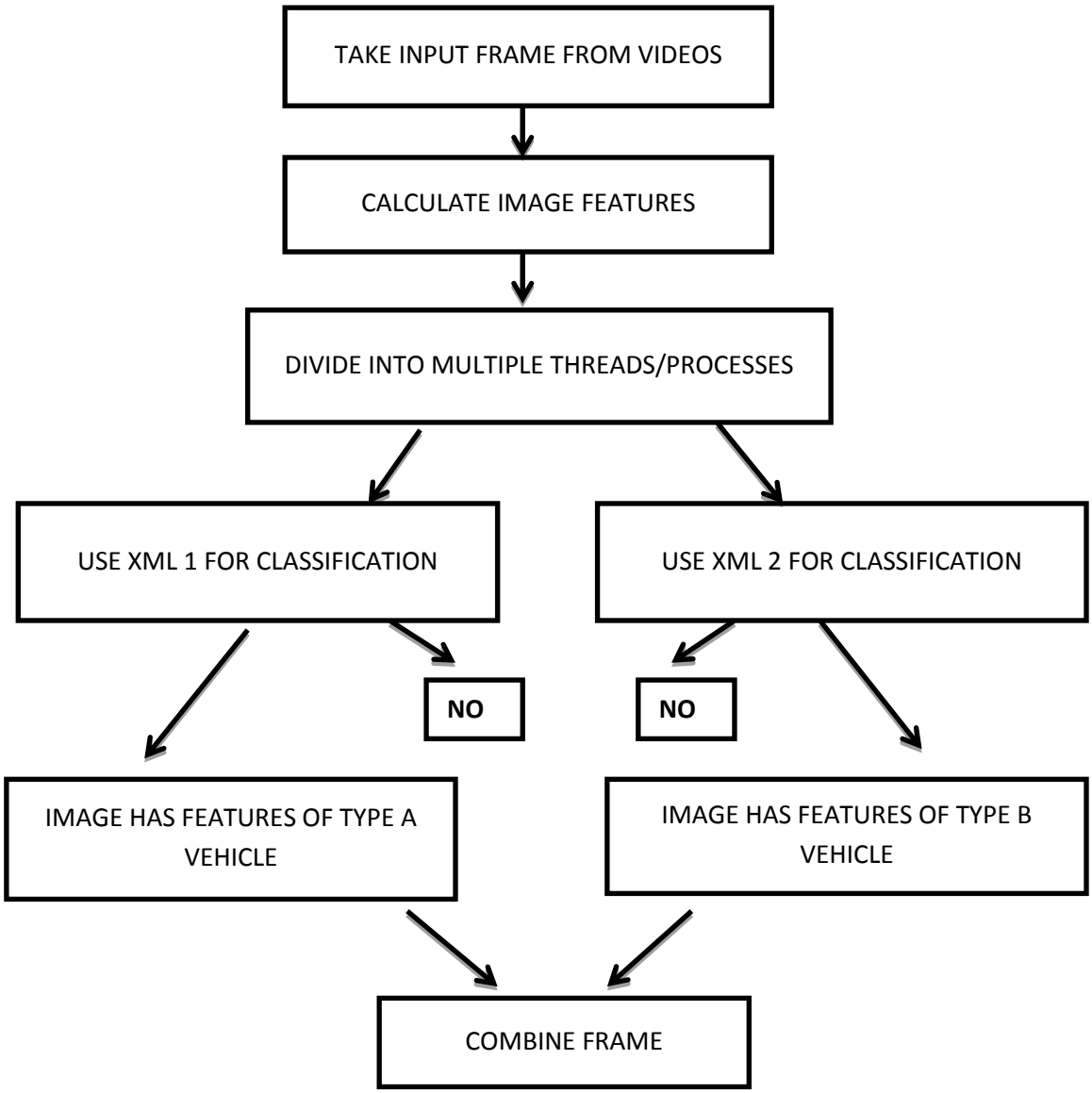

Fig 4.4: Flow-chart depicting parallel multi-classification

4.6 Using Haar-like Features for Training

The steps for training a Haar Classifier and detecting an object can be divided into:

- Creating the description file of positive samples
- Creating the description file of negative samples
- Packing the positive samples into a *vec* file
- Training the classifier
- Converting the trained cascade into a xml file

All the steps have been explained in detail. First of all, we need a large number of images of our object. We can shoot a small video (if possible 360 degree) of our object and then use ffmpeg to extract all the frames. A 25fps video of 1 minute will yield around 1500 pictures. For our purpose and testing, we have used an already assembled MIT image dataset.

```
$ ffmpeg  -i  Video.mpg  Pictures%d.bmp
```

Once the positive and negative images are prepared, it is put into two different folders named *positive* and *negative*. The next step is the creation of description files for both positive and negative images. The description file is just a text file, with each line corresponding to each image. The fields in a line of the positive description file are: the image name, followed by the number of objects to be detected in the image, which is followed by the x, y coordinates of the location of the object in the image. Some images may contain more than one object.

Next we create the description file of negative samples. The description file of negative samples contains only the filenames of the negative images. It can be easily created by listing the contents of the negative samples folder and redirecting the output to a text file, i.e., using the command:

```
$ ls > negative.txt
```

Now we can create the samples for training. All the positive images in the description file are packed into a .vec file. It is created using the createsamples utility provided with opencv package. Its synopsis is:

```
$opencv-createsamples -info positive.txt -vec posvec.vec –w20 -h20
```

Our positive image descriptor file was named positive.txt and the name chosen for the vec file was posvec.vec. Since a car was taken as a sample, minimum width of the

object was selected as 20 and height as 20 (as specified in MIT dataset). The above command yielded a vec file. The contents of a vec file can be seen using the following command:

```
$opencv-createsamples -vec vecfile.vec -show
```

Now everything is ready to start the training of the classifier. For that, we use the opencv-haar training utility. Its synopsis is:

```
$opencv-haartraining -data haar -vec posvec.vec -bg
negative.txt -nstages 30 -mem 2000 -mode all -w 20 -h 20
```

It creates a directory named Haar and puts the training data into it. The arguments given defines the name of vec file, background descriptor file, number of stages which is given here as 30, memory allocated which is 2 Gb, mode, width, height etc. This step is the most time consuming one. It takes about 5- 7 hours to complete the training of a sample of 50 images.

Once the training is over, we are left with a folder full of training data (named Haar in this case). The next step is to convert the data in that directory to an xml file. It is done using the convert_cascade program given in the openCV samples directory.

A sample tree from the training file is shown below (Discussed more in results)

```
-0.0161712504923344
0.5866637229919434
-0.5909150242805481

      7 18 5 2 -1.
      7 19 5 1 2.

   0

0.0119725503027439
-0.3645753860473633
0.8175076246261597

      5 12 11 4 -1.
      5 14 11 2 2.

   0

0.0554178208112717
-0.5766019225120544
0.8059020042419434
```

4.7 Blob Tracking and Pixel Based Adaptation Segmentation

The tracking consists of two phases: vehicle detection and vehicle recognition. Once the vehicle is detected or recognized in a frame of video, it is tracked by looping the capture and detection. So, each frame is grabbed, detected and recognized for vehicles. The first detection scheme used is the Blob detection. This is a motion based detection scheme whereby, any moving object on the road is segmented using background subtraction methods [16]. Hence, the moving blobs or spaces in the consecutive images are detected as vehicles. The image is smoothened first so that none of the small moving things like pedestrians are detected. The picture below shows how the background is subtracted. The white spaces depicting the moving vehicles on the road are shown in the figure below.

Fig 4.5 Pixel Based Adaptive Segmentation (PBAS) of images

4.8 Analysis Framework

The field of view of the camera is divided into two segments: State A and State B. A line (vertical- depending on the camera position) demarcates the two states. Whenever a moving blob or any detected vehicle crosses over that line, the count has to be increased by 1. This information is stored and updated periodically so that the statistical data for analysis is stored along with the current time stamp.

We have named this analysis framework as ‘vehicle counting’ framework, as the basic purpose of this module is to analyse the density of the traffic. The collected data is then used for plotting static graphs using Gnuplot software. The figure below shows how the process of counting vehicles is performed.

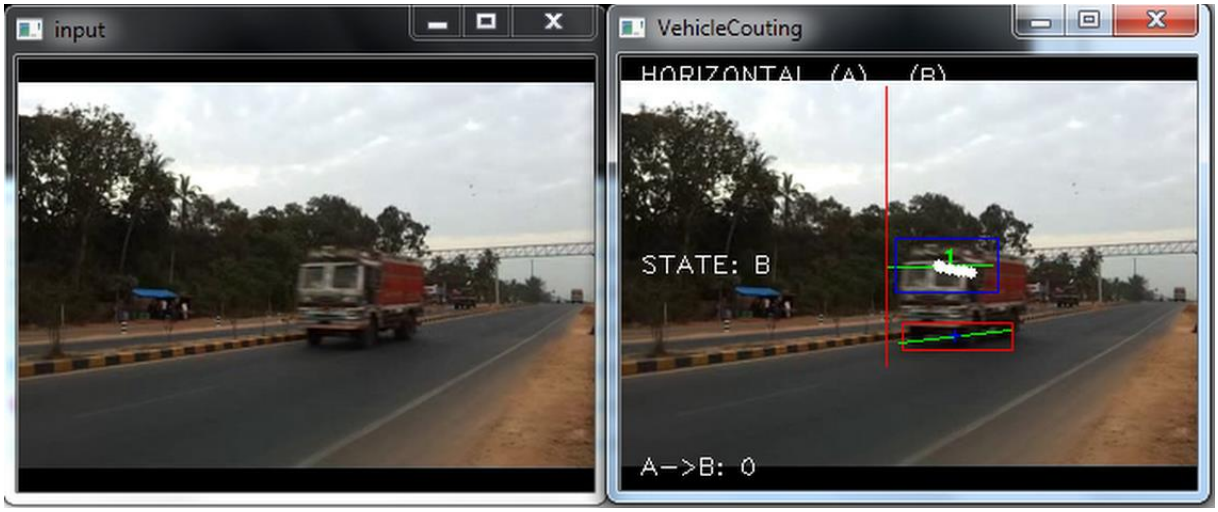

Fig 4.6 Vehicle counting framework

Once, the analysis is done, we can add an inference sub-system which gives actions to the raw data obtained, like control of traffic lights.

4.9 Implementation and Code Details

These describe the main function used in actual implementation for calculation of integral histograms. The various openCV functions available for this are described in this section. Actual implementation uses these APIs to achieve the required functionality:

1. **Converting image to grayscale**

   cvCvtColor(input_image, grey_image, CV_BGR2GRAY)
   It is used to convert an image from RGB to Grey format or division of images into plane images (1- dimensional image)

   cvEqualizeHist(src, dst)
   The function equalizes the histogram of the input image using the following algorithm:

a. Calculate the histogram H for src .

b. Normalize the histogram so that the sum of histogram bins is 255.

c. Compute the integral of the histogram:

$$H'_i = \sum_{0 \le j < i} H(j)$$

d. Transform the image using H' as a look-up table:

$$\texttt{dst}(x,y) = H'(\texttt{src}(x,y))$$

e. The algorithm normalizes the brightness and increases the contrast of the image.

2. Generate Sobel image in x and y direction (Sobel is an advanced edge detection technique)

void cvSobel(const CvArr* *src*, CvArr* *dst*, int *xorder*, int *yorder*, int *apertureSize=3*)

Calculates the first, second, third or mixed image derivatives using an extended Sobel operator. Parameters are as follows:

- *src* – Source image of type CvArr*
- *dst* – Destination image
- *xorder* – Order of the derivative x
- *yorder* – Order of the derivative y
- *apertureSize* – Size of the extended Sobel kernel, must be 1, 3, 5 or 7

3. Detection frame-work

```
void detectvehicles(IplImage *frame)
{
int i;
CvSeq
*vehicles=cvHaarDetectObjects(frame,cascade,Membuffer,1.1
,3,0,cvSize(30,30));//Maximum 3 neighbourhood //images
and the last cvsize gives minimum size of //vehicle
```

```
for(i=0;i<(vehicles?vehicles->total:0);i++)

{CvRect *r=(CvRect *)cvGetSeqElem(vehicles,i);

cvRectangle(frame,cvPoint(r->x,r->y),cvPoint(r->x+r-
>width,r->y+r->height),CV_RGB(255,0,0),1,8,0);



//Draws a rectangle with two points given with color //of
red and thickness as 8 and line type - 1

}
```

4. Loading the Haar Cascade Classifier

```
CvHaarClassifierCascade *cascade=(CvHaarClassifierCascade
*)cvLoad ("filename",0,0,0); //Loading Haar file
```

5. Vehicle Counting Framework

Here the main working function, process() has been shown. This function keeps the track of vehicles and counts them using other functions in the vehicleCounting class.

```
void VehicleCouting::process()
{
  // Initialize config and set the counting line

  for(std::map<cvb::CvID,cvb::CvTrack*>::iterator it =
tracks.begin() ; it != tracks.end(); it++)
  {
    cvb::CvID id = (*it).first;
    cvb::CvTrack* track = (*it).second;

    CvPoint2D64f centroid = track->centroid;

    if(track->inactive == 0)
    {
      if(positions.count(id) > 0)
      {
        std::map<cvb::CvID, VehiclePosition>::iterator it2 =
                                          positions.find(id);
        VehiclePosition old_position = it2->second;
```

```
        VehiclePosition current_position =
                                getVehiclePosition(centroid);

        if(current_position != old_position)
        {
          if(old_position == VP_A && current_position == VP_B)
            countAB++;
          if(old_position == VP_B && current_position == VP_A)
            countBA++;
           positions.erase(positions.find(id));
        }
      }
      else
      {
        VehiclePosition vehiclePosition =
        getVehiclePosition(centroid);

        if(vehiclePosition != VP_NONE)
          positions.insert(std::pair<cvb::CvID,
          VehiclePosition>(id,vehiclePosition));
      }
    }
    else
    {
      if(positions.count(id) > 0)
        positions.erase(positions.find(id));
    }
// Other not so important functions condensed in the report
  if(showAB == 0)
  {
    cv::putText(img_input, "A->B: " +
boost::lexical_cast<std::string>(countAB), cv::Point(10,img_h-
20), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
    cv::putText(img_input, "B->A: " +
boost::lexical_cast<std::string>(countBA), cv::Point(10,img_h-
8), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));
  }

  if(showAB == 1)
    cv::putText(img_input, "A->B: " +
boost::lexical_cast<std::string>(countAB), cv::Point(10,img_h-
8), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));

  if(showAB == 2)

cv::putText(img_input, "B->A: " +
boost::lexical_cast<std::string>(countBA), cv::Point(10,img_h-
8), cv::FONT_HERSHEY_PLAIN, 1, cv::Scalar(255,255,255));

}
```

6. Blob Tracking

The following code snippet describes the implementation details of blob based tracking scheme as described in the previous section.

```
void BlobTracking::process()
{
  IplConvKernel* morphKernel = cvCreateStructuringElementEx(5,
                               5, 1, 1, CV_SHAPE_RECT, NULL);
  cvMorphologyEx(segmentated, segmentated, NULL, morphKernel,
                                          CV_MOP_OPEN, 1);

  if(showBlobMask)
    cvShowImage("Blob Mask", segmentated);

  IplImage* labelImg = cvCreateImage(cvGetSize(frame),
                                  IPL_DEPTH_LABEL, 1);

  cvb::CvBlobs blobs;
  unsigned int result = cvb::cvLabel(segmentated, labelImg,
                                                  blobs);

  cvb::cvFilterByArea(blobs, minArea, maxArea);

  if(debugBlob)
    cvb::cvRenderBlobs(labelImg, blobs, frame, frame,
CV_BLOB_RENDER_BOUNDING_BOX|CV_BLOB_RENDER_CENTROID|CV_BLOB_RE
NDER_ANGLE|CV_BLOB_RENDER_TO_STD);
  else
    cvb::cvRenderBlobs(labelImg, blobs, frame, frame,
CV_BLOB_RENDER_BOUNDING_BOX|CV_BLOB_RENDER_CENTROID|CV_BLOB_RE
NDER_ANGLE);

  cvb::cvUpdateTracks(blobs, tracks, 200., 5);

  if(debugTrack)
    cvb::cvRenderTracks(tracks, frame, frame,
CV_TRACK_RENDER_ID|CV_TRACK_RENDER_BOUNDING_BOX|CV_TRACK_RENDE
R_TO_STD);
  else
    cvb::cvRenderTracks(tracks, frame, frame,
CV_TRACK_RENDER_ID|CV_TRACK_RENDER_BOUNDING_BOX);

}
```

7. Traffic Analysis framework

The following code snippet processes the imput frame through the frameworks described above, i.e. Blob detection, Haar Classification and Vehicle Counting. The results are stored in a file for analysis.

```
if(!img_mask.empty())
{
// Blob tracking each frame
blobTracking->process(img_input, img_mask, img_blob);

// Vehicle counting in each frame
vehicleCouting->setInput(img_blob);
vehicleCouting->setTracks(blobTracking->getTracks());
vehicleCouting->process();

// Haar Classification
detect(frame);

//Writing data to a file
myfile <<(((atoi(getDtTm(buff))-sec)< 0)
?(atoi(getDtTm(buff))+ 60 -sec):(atoi(getDtTm(buff))-sec))<<
"\t" <<(vehicleCouting->countAB +vehicleCouting->countBA) <<
"\n" ;

}

//-------------------------------------------------------------
// Time Storage function

static char *getDtTm (char *buff)
{
    time_t t = time (0);
    strftime (buff, DTTMSZ, DTTMFMT, localtime (&t));
    return buff;
}
```

Chapter 5 Results

The main parts of application: Vehicle detection, Vehicle Counting, Vehicle feature Recognition and Traffic Analysis were implemented and tested before and after integration. The application is detecting almost every vehicle, but only recognizing 75-85% cars as these are feature based recognitions depending highly on training data. Two different ways in which multi-classification can be done is serial and parallel. It is better to use serial classification here as division and combination of threads is a costly process if the number of classes are small.

The following rooted tree structure was obtained when the Haar cascade classifier was trained using a set of 1000 images.

```
        // openCV Haar Classifier
        20 20                                   // size of images
         //<root node- tree 0>
                                6 12 8 8 -1.        // rectangle feature vector
                                6 16 8 4 2.         // rectangle feature vector

                            0

                        0.0452074706554413    // threshold value
                        -0.7191650867462158   // left node value
                        0.7359663248062134    // right node value

         // <tree 1>
                                1 12 18 1 -1.       // rectangle feature vector
                                7 12 6 1 3.         // rectangle feature vector

                            0

                        -0.0161712504923344    // threshold value
                        0.5866637229919434     // threshold value
                        -0.5909150242805481    // threshold value
         // <tree 2>
                                7 18 5 2 -1.        // rectangle feature vector
                                7 19 5 1 2.         // rectangle feature vector

                            0

                        0.0119725503027439      // threshold value
                        -0.3645753860473633     // threshold value
…
```

We have chosen openCV as the library to implement image processing techniques mainly due to the reason that it has real time applications and is faster than other software such as MATLAB. We have used Sobel edge detection as the edge feature to be detected scheme and Gaussian smoothing has been eliminated for faster processing.

The following snapshots have been taken while testing the above classifier with random images from over the internet.

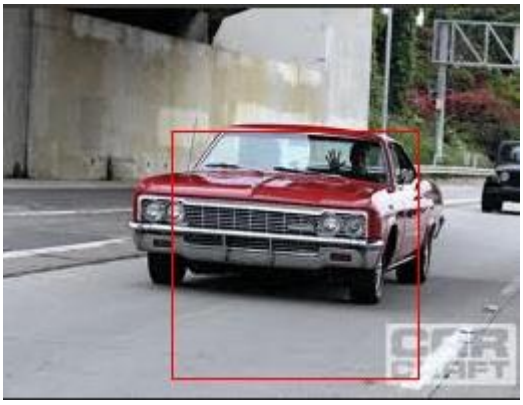

Fig 5.1: Detection of a single instance

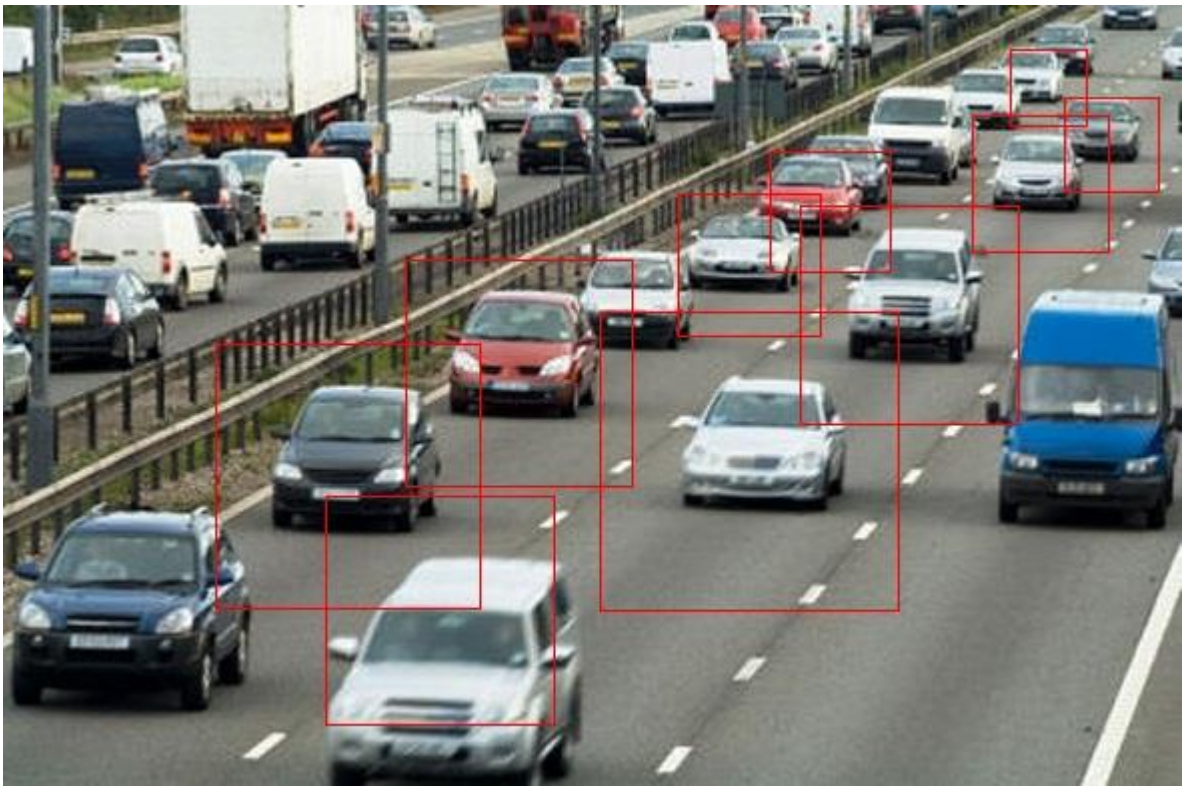

Fig 5.2: Detection of multiple instances (13 of 16 detected)

When lighting is poor or the image is dull, the detection ratio falls as shown. Here, the detection is not efficient and falls below 50%.

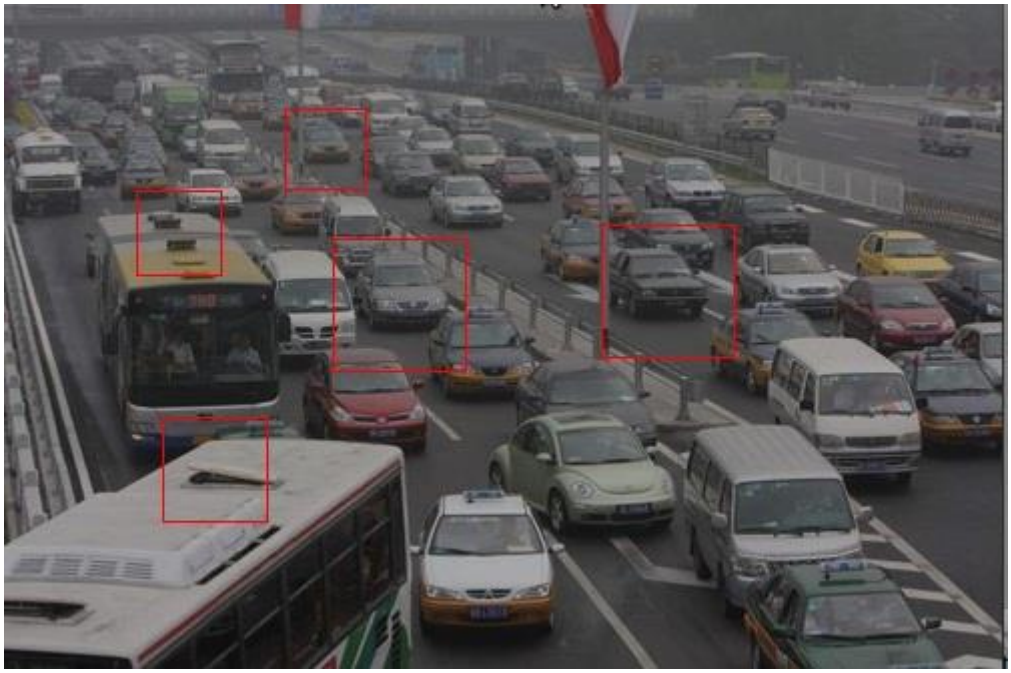

Fig 5.3: Detection in poor lighting

The following table illustrates the positive to negative hits during testing. The data is collected over a testing of 25 images taken randomly. Please note that the detection percentage may change depending on the images and training data and the number of images tested with.

Table 5.1: Statistical data obtained after testing with random vehicle images

Conditions	**Positive Recognition**	**Negative Recognition**
Light	~ 40%	NIL (<10%)
Normal	~ 85%	NIL
Dark	~ 40%	~ 20 %

The following video was taken outside NITK campus and describes the application working in detail.

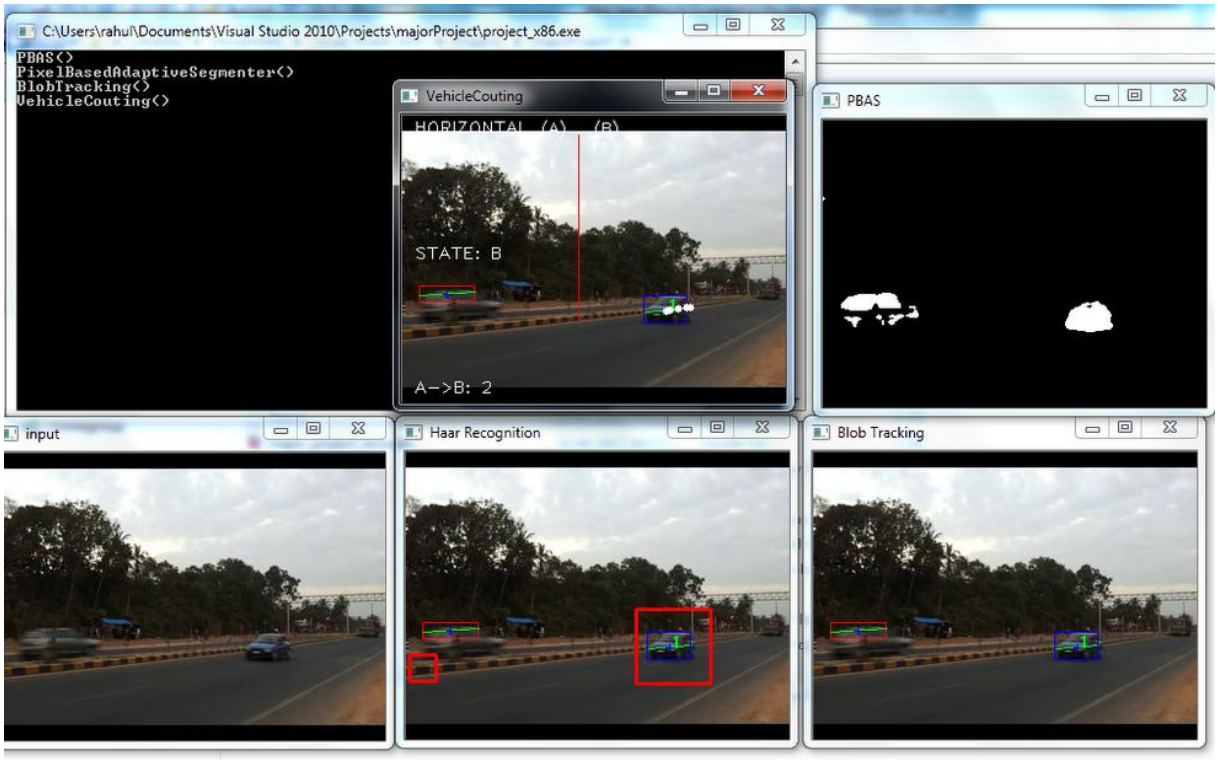

Fig 5.4: Overall application at work. From top- The main execution screen, the vehicle counting window, PBAS window, input screen, HAAR Recognition and Blob tracking window

The statistical data obtained is stored in a file and Gnuplot was used to plot the graph as shown in next few pages.

Table 5.2: Statistical data with Blob Tracking, Haar recognition using sampling videos

Conditions	**Positive Recognition**	**Negative Recognition**	**Positive Blob Detection**	**Negative Blob Detection**
Light	~ 40%	NIL (<10%)	~ 80%	NIL (<10%)
Normal	~ 85%	NIL	100%	NIL
Dark	~ 40%	~ 20 %	~ 60%	~ 20%

Note: During varied lighting conditions, sometimes, a single vehicle is detected as two blobs rather than one. This is the source of negative detection.

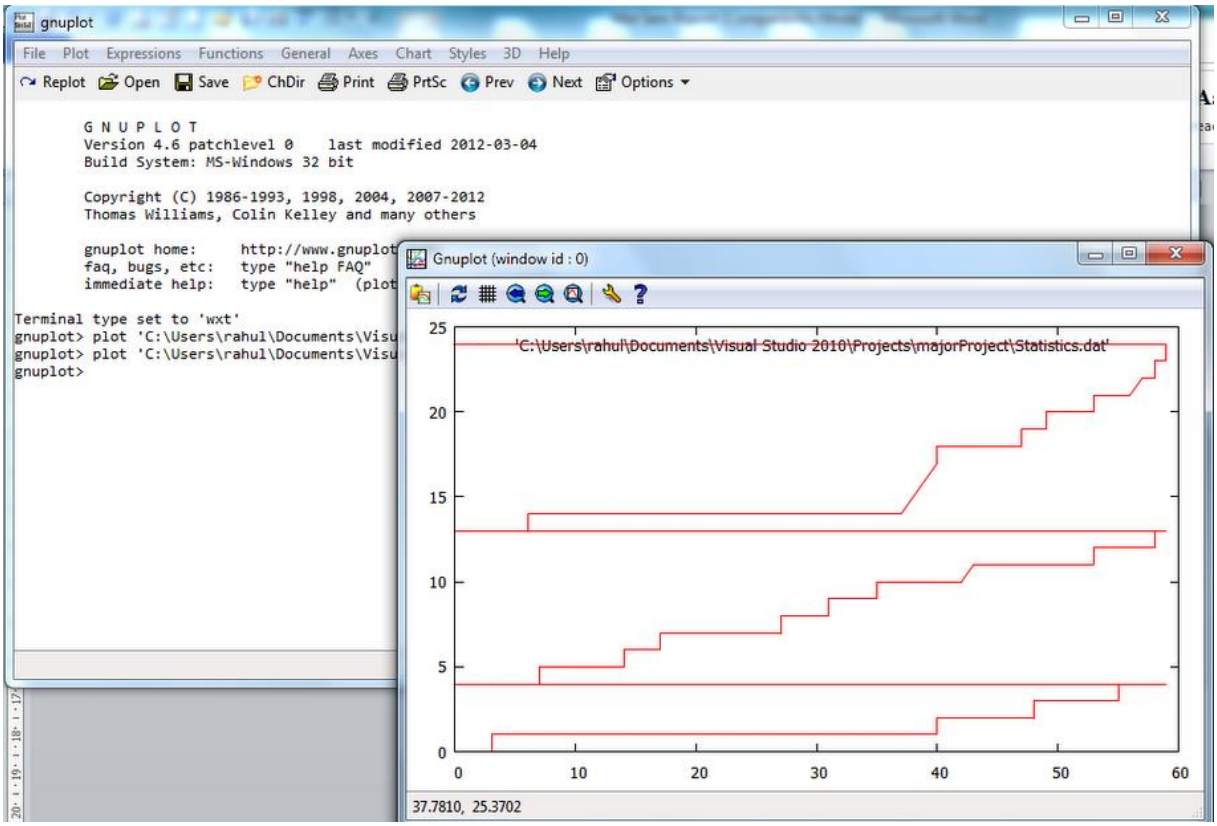

Fig 5.5: The graph depicting cumulative traffic density over a 3 minute video.

The Highway NH 17 which passes through NITK campus was analysed for traffic on Friday, April 19, 2013 and the results obtained were: During office hours, rush is maximum with maximum number of busses and cars.

Time	Density
10	32
11	26
12	19
13	10
14	12
15	11
16	17
17	25
18	32
19	28

Note: redundant entries have been removed and normalized to make the data presentable and meaningful.

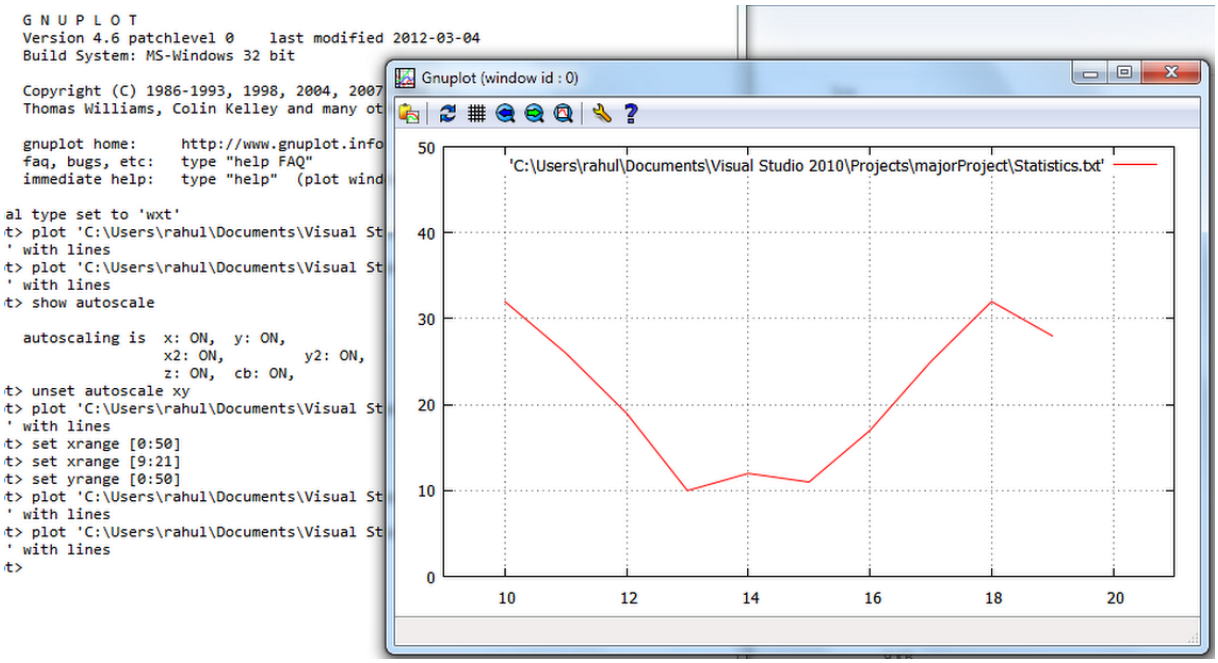

Fig 5.5: The graph absolute traffic density on NH-66, over a period of 8 hours with 2 minute sample video recordings.

Chapter 6 Conclusion and Scope for Future Work

The following section describes the conclusions of this report and proposed future work (major).

6.1 Conclusion

The working of the application is as follows: The input images/ frames of the video are processed with blob tracking. Once moving vehicles are detected, the vehicle counting module updates and stores the count value in a file in real time. Then, the moving vehicles are classified as cars, trucks using a classifier (Haar classifier based decision tree). The Stored counting data is further used to plot density-time graphs using Gnuplot.

The Haar cascade classifier has been trained and used to detect vehicles in still images as well as recorded video. The recognition rate under normal conditions is about 80%. The detection scheme which uses Blob tracking and Pixel Based Adaptive Segmentation has a very good accuracy and detects/ tracks almost 95- 100% of moving vehicles. However, if vehicles are stationary on the road, only Haar recognition works and Blob tracking fails. Both serial and parallel multi classification techniques have been tested, but, as the numbers of classes are less, serial classification is done.

The entire application is coded using C++ language with the help of external libraries like openCV, Boost, BGS (Background Subtraction) apart from other standard libraries.

6.2 Scope for Future Work

The future applications of this software are many, including but not limited to the following:

1. The data collected can be used to build smart traffic light controllers using fuzzy logic rules to automatically adjust traffic light timings for optimal performance.

2. With the help of more powerful cameras, traffic monitoring tool can be made which uses speed and licence plate number to automatically send tickets or challans to defaulters.

3. The data collected can be used to build applications for scheduled maintenance od roads. i.e. if the congestion in certain part of city is more, the authorities can focus only on maintaining or developing these sites first

4. A user application, which collects congestion information from many such systems running our application, so that the user can select the least congested path for travel. This application along with GPS technology may help in real time traffic monitoring too.

7. References

Papers

[1] Weiming Hu et al., "A survey on visual surveillance of object motion and Behaviours," in *IEEE Transaction on Systems, Man and Cybernetics*, 2004, pp. 34-3.

[2] Fatih Porikli, "Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005

[3] B. Triggs, Navneet Dalal, "Human Detection Using Histograms of Flow and Appearance," in *European Conference on Computer Vision*, Graz, Austria, 2006

[4] M. J. Gangeh, B. M. Romeny, "Scale-Spaced Texture Classification Using Combined Classifiers," in *SCIA*, 2007

[5] Thorsten Joachims, "Training Linear SVMs in Linear time," unpublished, 2007.

[6] Norbert Buch et al., "A Review of Computer Vision techniques for the Analysis of Urban Traffic," in *IEEE Transactions on Intelligent Transportation Systems*, Vol. 12, No. 3, 2011

[7] Sungji Han et al., "Vehicle Detection Method using Haar-like feature on Real Time System," in *World Academy of Science, Engineering and Technology*, 2009

[8] R. Lienhart, J. Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection", Intel Labs, Intel Corporation, Santa Clara, CA 95052, USA

[9] P. Voila, M. Jones, "robust real-time Object Detection", in *Second International workshop on Statistical and Computational Theories of Vision*, Vancouver, Canada, July 2001

[10] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Detecting moving objects, ghosts and shadows in video streams", in *IEEE Trans. on Patt. Anal. and Machine Intell.*, vol. 25, no. 10, Oct. 2003, pp. 1337-1342.

Online Sources

[11] Support Vector Machines.
Available: http://en.wikipedia.org/wiki/Support_vector_machine

[12] Histogram of Oriented Gradients.
Available: http://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

[13] Wei Zeng. (2012, April). A Generic Rapid Object Detection in Complex Scenes. NFC Labs, China. Available: http://optimal.opt.ac.cn/VALSE2012/data/%E6%9B%BE% E7%82%9C.pdf

Books

[14] Ian Somerville, *Software Engineering*, 8th ed. China: Pearson Education Limited, 2007.

[15] Gary Bradski, A. Kaehler, *Learning openCV*. California, USA: O'Reilly Media Inc., 2008.

Documentation

[16] *OpenCV 2.0- Reference*, Willow Garage Inc., USA. Available: http://opencv.willowgarage.com/documentation

[17] BGSlibrary: A OpenCV C++ Background Subtraction Library, Sobral A.C., 2012 Available: http://code.google.com/p/bgslibrary/

[18] Gnuplot, 2013. Available: http://www.gnuplot.info/

Thesis

[19] Rodrigo Verschae, "Object Detection Using Nested Cascades of Classifiers: A learning framework and its extension to multi-class case," Dept. of Electrical Engineering, University of Chile, 2010