iJRASET
International Journal For Research in
Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 5 **Issue:** VIII **Month of publication:** August 2017

DOI: http://doi.org/10.22214/ijraset.2017.8253

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VIII, August 2017- Available at www.ijraset.com

HAAR Cascade Based Dynamic Traffic Scheduling System

Rishabh Sachdeva[1], Sricheta Ruj[2]
[1]*Computer Engineering, Shri Govindram Seksaria Institute of Technology and Science*
[2]*Computer Engineering, Nirma University*

Abstract: Object Detection has become an important feature in field of Computer Science. Benefits of object detection are varied and are not restricted to any specific area. Instead, object detection technique is growing rapidly and used widely in Information Industry. This paper intends to address one such possibility with the help of Haar-cascade classifier. The focus of the paper will be on the case study to develop a Smart and Dynamic Traffic Management System which is based on vehicle detection mechanism.
Keywords: Haar cascades, OpenCV, Pixel, Image Feature, Adaboost

I. INTRODUCTION

A. Digital Image Processing

An image may be defined as two dimensional entity F(x,y), where x and y are spatial coordinates, and the amplitude of Function F at any pair of coordinates x,y is called intensity or gray level of image. When x, y and amplitude values of function F are finite and discrete entities, we call image a digital image.

The field of digital image processing refers to processing digital images by means of digital computer. A digital image is composed of finite number of elements, each of which has a particular location and value. These elements are referred to as picture elements, image elements and pixels.

B. RGB and Gray Scale Image

RGB (red, green, and blue) refers to a system for representing the colours to be used on a computer display. Red, green, and blue can be combined in various proportions to obtain any colour in the visible spectrum. Levels of R, G, and B can each range from 0 to 100 percent of full intensity. Gray scale images are distinct from one-bit bi-tonal black-and-white images, which in the context of computer imaging are images with only the two colours, black, and white (called bi-level or binary images) [1].

II. HAAR CASCADES

Object detection through Haar Cascades is a machine learning based method to achieve object detection with high detection rate. To achieve this, a cascade function is trained with data containing large number of positive and negative images. Positive image refers to images of desired object and negative image refers to object not containing the desired object, which can be found around the object to be detected. Now, the image contains a large number of convolution matrixes or masks. These masks are used to determine features in image. Even a small window can contain thousands of masks. Therefore, this job can be very expensive. To simplify this process, concept of integral image is introduced. In this approach, focus is on three types of features as depicted in image below. Therefore, the operation revolves around these four pixels. Each feature in an image is determined by subtracting sum of pixel values sum in white area of rectangle from black area. When the desired feature matches the feature property of desired object, the object is marked. To understand the approach better, consider an example of car. There are some generic characteristics in case of a car. Firstly, there is some intensity gap between the windshield glass and bonnet of a car. Secondly, there is an intensity gap between headlights and metal part between them. In these two cases, line features will work well. When we consider the case of intensity gap between bonnet and windshield, edge features will fit well. Similarly, there are number of features that can be detected and are common in most of the vehicles.

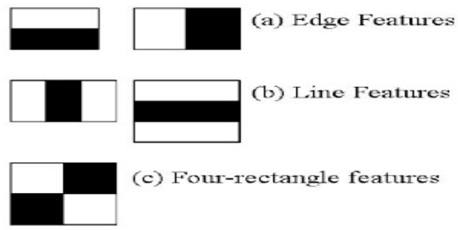

Fig. 1 Image Features

III. FEATURE CALCULATION BY INTEGRAL IMAGE

Integral image is an intermediate result of an image. This is used to determine rectangular features in the image. The integral image is generated from original image to make further process faster. Integral image is used to felicitate quick feature detection. It is the outline of the pixel values in the original image. The integral image at any location (coordinates x,y) is determined by adding the pixels above and to the left of coordinates x and y [2].

$$I(x,y) = \sum_{\substack{x^i<x \\ y^i<y}} O(x^i, y^i)$$

Consider I as integral image and O as original image. Referring Fig 2, value of integral image at point 1, is the total summation of pixels in Area A. At point 2, value is pixel summation of area A and area B. Similarly, at location 4 values is A+B+C+D. The pixel sum within D can be calculated as 4 - (2+3) - 1. So, the features are calculated by subtracting pixel values in different rectangles. To compute edge features (Figure 1), six array references are considered, eight in the case of line features and nine in the case of four-rectangle features. Then, this value is compared with desired criteria threshold to judge whether the window object is desired one or not.

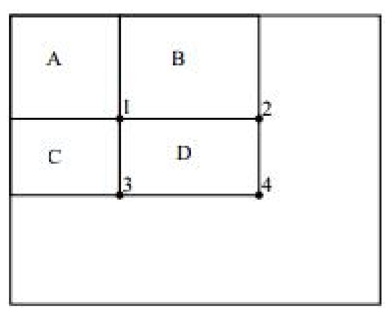

Fig. 2 Array references

IV. ADABOOST

AdaBoost (Adaptive Boosting) is a machine learning meta-algorithm formulated by Freund and Schapire [3] was the first practical boosting algorithm, and remains one of the most widely used and studied, with applications in numerous fields. Adaboost is a technique that helps in combining number of “weak classifiers” into a single “strong classifier”. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favour of those instances misclassified by previous classifiers. There is a standard characteristic of a weak classifier that it should have the detection rate of at least 50 per cent, .i.e. it should be able to detect at least half of the desired objects in an image. So, a bunch of trained classifiers can be used to develop an effectively trained mechanism to detect objects using Adaboost training mechanism. Following analysis is based on [3], [4] and [5].

A. Selection of Training Sets

Each weak classifier is trained on a random sub set of total training sets. AdaBoost assigns a “weight” to each training example, which determines the probability that each example should appear in the training set. Examples with higher weights are more likely to be included in the training set, and vice versa. After training a classifier, AdaBoost increases the weight on the misclassified examples so that these examples will make up a larger part of the next classifiers training set, and the next classifier trained will perform better on them.

B. Classifier Output Weights

After training of classifiers, the weight of each classifier is computed based on its accuracy. Classifiers that are more accurate are assigned more weight. Weighted classifiers are assigned more priority in determining results.

C. Definition and Equations

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

The final classifier consists of T weak classifiers. $h_t(x)$ is the output of weak classifier 't'. In this paper, outputs are limited to +1 or -1. α_t is the weight assigned to classifier t as determined by AdaBoost. The final output H is the linear summation of all the weak classifiers.

The classifiers are trained one at a time. After training of each classifier, probabilities of each of the training examples are updated for the next classifier. The first classifier (t=1) is trained with examples which all have equal probability. For output weight calculation of each classifier, following formula is used.

$$\alpha_t = \frac{1}{2}\ln\frac{1-e_t}{e_t}$$

Formula for output weight is based on classifier's error rate. $\boldsymbol{e_t}$ is the number of misclassification over the training set divided by the total size of training set.[4]

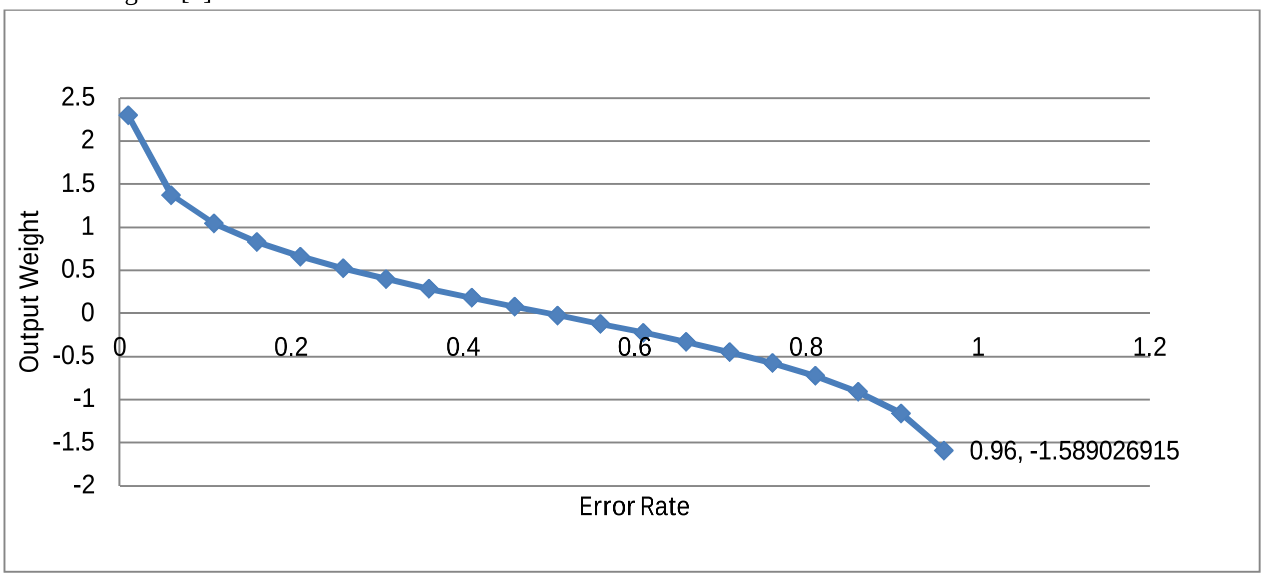

Fig. 3 Output weight vs Error rate

Following points should be noted from the above graph.

1) The weight of classifier grows exponentially when erroe rate tends to 0.
2) The weight of classifier is 0 when error rate is 0.5 .
3) The weight of classifier grows exponentially negative when error rate approaches one. Those classifiers with error rate greater that 0.5 are no better than random guessing.

After computing at, training example weights are updated using following formula:

$$D_{t+1}(i) = \frac{D_t(i)e^{(-a_t y_i h_t(x_i))}}{Z_t}$$

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VIII, August 2017- Available at www.ijraset.com

The variable D_t is a vector of weights, with one weight for each training example in the training set. 'i' refers to training example number. D_t refers to distribution. Each weight $D(i)$ represents the probability that training example 'i' to be selected as part of the training set. Weights are normalized by dividing each of them by sum of all the weights (Z_t). This vector is updated for each new weak classifier that is trained. D_t refers to the weight vector used when training classifier 't'. This equation needs to be evaluated for each of the training samples 'i'(x_i, y_i). Each weight from the previous training round is going to be scaled up or down by this exponential term. The function e^xwill return a fraction for negative values of x, and a value greater than one for positive values of x. So the weight for training sample 'i' will be either increased or decreased depending on the final sign of the term "$-\alpha * y * h(x)$". For binary classifiers whose output is constrained to either -1 or +1, the terms y and $h(x)$ only contribute to the sign and not the magnitude.

y_i is the correct output for training example 'i', and $h_t(x_i)$ is the predicted output by classifier t on this training example. If the predicted and actual outputs agree, $y * h(x)$ will always be +1 (either 1 * 1 or -1 * -1). Else, $y * h(x)$ will be a negative term. Ultimately, misclassification by a classifier with a positive a_t (output weight) will result in giving this training example a larger weight.

Note: By introducing a_t (output weight) in above formula, classifier's effectiveness is also taken into consideration when updating weights.

A much more in-depth explanation of Adaboost algorithm can be found in a book by Schapire and Freund[6].

V. APPLICATION IN DYNAMIC TRAFFIC MANAGEMENT USING OPENCV

This section is devoted to the description of proposed vehicle detection system and management traffic system to make everyday life easier. This application uses Adaboost algorithm (described in section IV) to train classifiers for vehicle detection.

OpenCV (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. In this paper, we will use OpenCV to implement cacades and detect objects. The system is composed of three parts: learning section, detection section and traffic light management section.

A. Learning Section

This sections consists of two phases: Extraction phase and Training phase

1) *Extraction Phase*: In this phase, descriptor based on haar wavelet [7] extracted for each image in training set, a feature vector. This database contains set of positive samples (containing vehicle) and negative samples (containing no vehicle). Negative images should contain pictures, which can be found near vehicles on roads like grass, sign boards, road sign and hoarding boards. The negative image database helps training classifiers to ignore such windows containing such images and save time and hence making operation effective and less expensive.

Positive samples are created by using opencv_createsamples application. Bunch of positive samples can be created using this utility of OpenCV. It takes one object image in input and creates a large set of positive samples from the given object image by randomly rotating the object, changing the image intensity as well as placing the image on arbitrary backgrounds. However, considering only one image of vehicle is not a good idea. To create a robust system, large number of vehicles images of various categories is required. Following is the command to run create_sample utility:

Opencv_createsamples.exe –info <*pathOfFileInfo.txt* > -vec <*vecFileName*>

Output of above command is in form of vector file (.vec). Info.txt contains location positive sample images following the coordinates of bounding rectangle as mentioned below.

rawdata/positiveImages/pic1.bmp 1 80 73 86 176
rawdata/positiveImages/pic2.bmp 1 114 116 84 218

2) *Training Phase:* The next step is the actual training of the boosted cascade of weak classifiers, based on the vector file that was prepared in extraction phase. This where AdaBoosting (section IV) comes into picture. Internally, OpenCV uses boosting to train classifiers. The following command is used:

opencv_traincascade.exe -data classifier -vec <*vecFileName*> -bg <*PathOfNegative.txt*> -numStages 7 -miniHitRate 0.98 -maxFalseAlarmRate 0.5

numStages refers to number of cascade stages to be trained. More details of parameters are present in [8]. Negative.txt contains location of negative image samples as shown below.

C:\majorProject\opencv\images\negativeImages\pic60.bmp
C:\majorProject\opencv\images\negativeImages\pic161.bmp

After the opencv_traincascade application has finished its work, the trained cascade will be saved in cascade.xml file in the -data folder. The commands mentioned above with additional parameters are present in detail in official website of OpenCV [8].

3) *Object Detection Section:* The trained classifier generated in training phase (cascade.xml) is used to detect desired object. The cascade.xml is loaded into detection program, image is passed as input, converted into gray scale image and finally program returns the position of detected object as Rectangle (x,y,w,h). Further, the count of objects detected is used in next stage.

Fig. 4 Original Frame

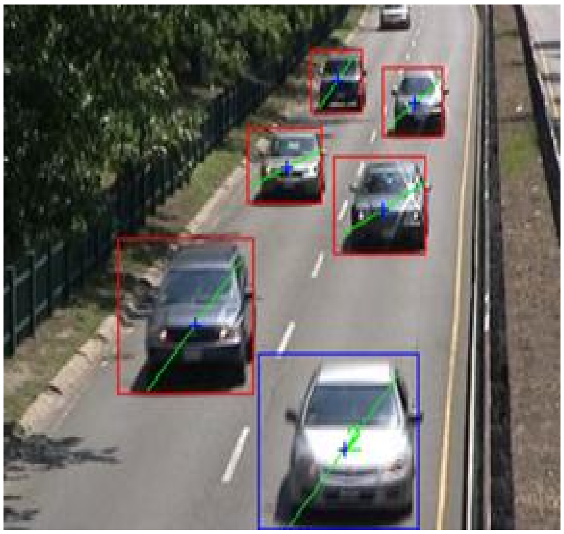

Fig. 5 Vehicle tracking

4) *Traffic Management:* The vehicle count obtained in object detection section is the key to determine time allocation for each of the lanes. To determine vehicle count, images need to passed as input to the system. To make this job easier, OpenCv provides utilities to access camera via program. More details on this can referred from [9]. The lane in which has high count of vehicles is allotted more time in comparison to lanes having less vehicles. The traffic lights allow the passage of lane traffic accordingly. In current scenario of growing population, such an approach will help in saving time and making everyday life easier. Appropriate time required for vehicle passage to each lane is allotted. Microcontroller layer receives the output duration and LEDs display the relevant information. As a result, vehicles in other lanes will not have to wait for pre-determined duration of traffic lights. Refer fig.6 for detailed flow diagram. There should be some threshold time for each of the lanes even if zero vehicle detection considering system limitations.

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VIII, August 2017- Available at www.ijraset.com

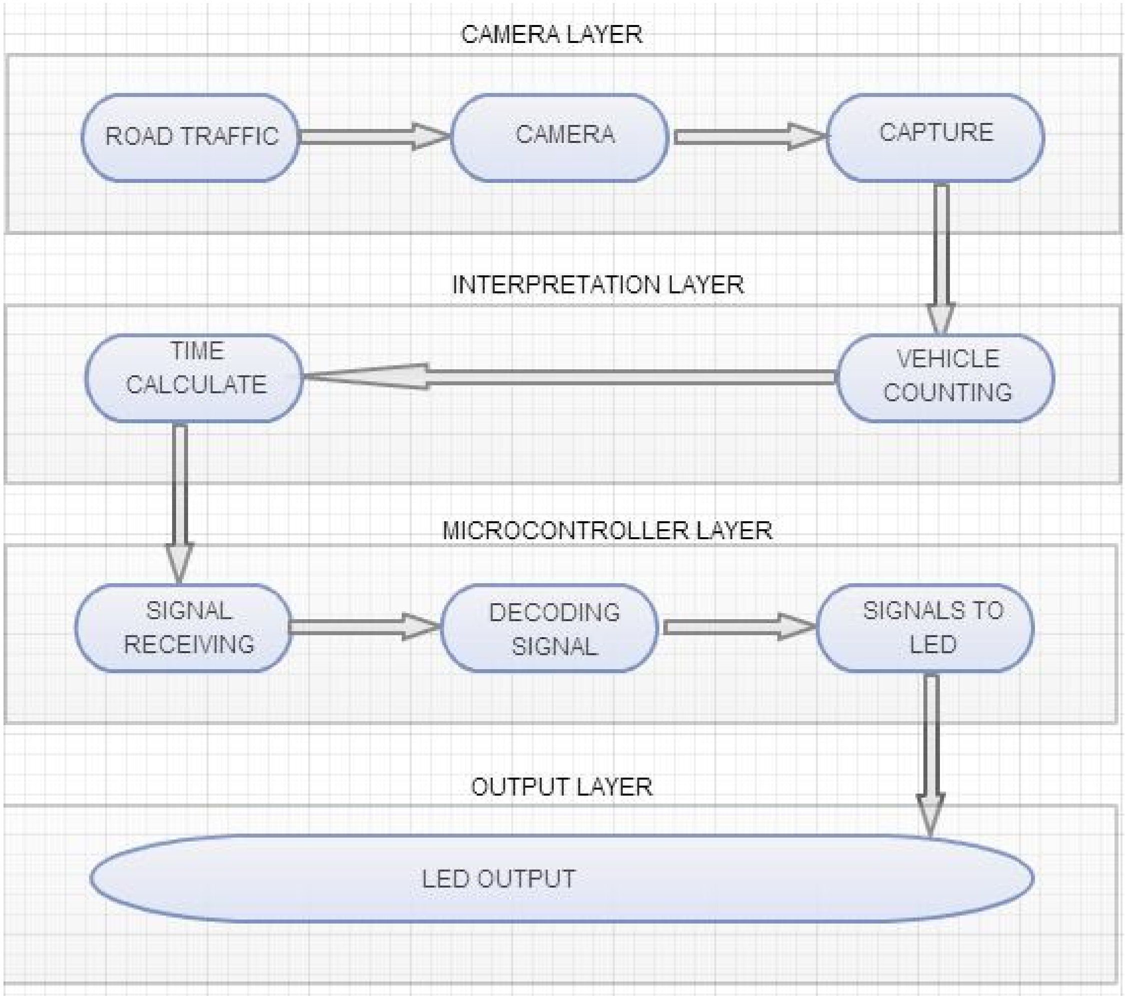

Fig. 6 Flow diagram for traffic scheduling system

VI. CONCLUSION

Traffic congestion is a severe problem and the widespread use of information technology provides an opportunity to enhance the techniques of Traffic Management systems. This paper proposes a solution to achieve this via dynamic traffic scheduling. The image is first captured of a particular traffic lane from a traffic square. Then the adequate time duration is allotted to that lane for vehicle passage according to traffic density. There is always some threshold and maximum duration.Concept of Haar Cascade is used in implementation of classifiers and object detection. Haar Cascades based on Adaboost is used in this paper to train classifiers, which is further used to detect vehicles. AdaBoost, short for "Adaptive Boosting", is a machine learning meta-algorithm that helps in combining multiple "weak classifiers" into a single "strong classifier". The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. OpenCV (Open Source Computer Vision Library) provides utilities for implementation of trained classifiers using positive and negative image samples as dataset.

REFERENCES

[1] G. Jyothi, CH. Sushma, D.S.S. Veeresh."Luminance Based Conversion of Gray Scale Image to RGB Image". International Journal of Computer Science and Information Technology Research. Vol. 3, Issue 3, pp: (279-283), Month: July - September 2015, Available at: www.researchpublish.com

[2] Monali Chaudhari#1 , Shanta sondur*2 , Gauresh Vanjare$3. "A review on Face Detection and study of Viola Jones method". International Journal of Computer Trends and Technology (IJCTT) – volume 25 Number 1 – July 2015

[3] Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences 55(1), 119–139 (1997)

[4] Lecture on "Survey of Boosting from an Optimization Perspective by Manfred K. Warmuth". Available at https://www.youtube.com/watch?v=R3od76PZ08k&list=

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue VIII, August 2017- Available at www.ijraset.com

[5] Robert E. Schapire .” Explaining AdaBoost”. Available at http://rob.schapire.net/papers/explaining-adaboost.pdf .
[6] Schapire, R.E., Freund, Y.: Boosting: Foundations and Algorithms. MIT Press (2012)
[7] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features. In: Computer Vision and Pattern Recognition. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on; Vol. 1, pp. I-511.
[8] http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.htmlandhttp://docs.opencv.org/trunk/db/d28/tutorial_cascade_classifier.html
[9] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html