A Real Time Prediction Of Traffic Speed

**V SEMSTER MINI PROJECT – B.TECH (IT) – IIIT ALLAHABAD**



Guidance of-

Dr.Pavan Chakraborty Submitted by-

Piyush Tiwari(RIT2012003)

Baidyanath Prasad(RIT2012010)

Angel Tiwari(RIT2012038)

Ashish Lakra(RIT2012080)

Anusha Chimmili(RIT2012081)

DECLRATION

We hereby declare that the project work entitled “**A Real Time Speed Estimation of Traffic** ” submitted to the IIIT Allahabad, is a record of an original work done by us under the guidance of

Dr. Pavan Chakraborty and has not been submitted to any other University or Institute

or published earlier.

Place: Amethi **Piyush Tiwari (RIT2012003)**

**Baidyanath (RIT2012010)**

**Angel Tiwari (RIT2012038)**

**Ashish Lakra (RIT2012080)**

**Anusha Chimmili (RIT2012081)**

CERTIFICATE FROM THE MENTOR

It is certified that this project report “**A Real Time Speed Estimation of Traffic** ” of mini project taken in V semester is the bona fide work of “Piyush Tiwari (RIT2012003) , Baidyanath (RIT2012010),Angel Tiwari (RIT2012038) ,Ashish Lakra (RIT2012080), & Anusha Chimmili(RIT2012081).**”** who carried out the project work under my supervision.

Dr. Pavan Chakraborty,

Faculty In-charge (Ph.D. Cell),

Faculty Coordinator (IT),

IIIT – Allahabad, Allahabad

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to all the people who helped and supported me. This project would not be complete without the thoughtful guidance of my project mentor, Dr. Pavan Chakraborty. We would also take this opportunity to sincerely thank Mr. Avinash Kumar Singh, who guided us at every stage of the project. A special thanks has to be given to our batch for data collection.

Place: Allahabad **Piyush Tiwari (RIT2012003)**

**Baidyanath (RIT2012010)**

**Angel Tiwari (RIT2012038)**

**Ashish Lakra (RIT2012080)**

**Anusha Chimmili (RIT2012081)**

**CONTENT**

1. Introduction
2. Problem Statement
3. Literature Survey
4. Proposed Approach
5. Activity Chart
6. Implementation
7. Description of Hardware and Software Requirements
8. Limitations
9. References

1.Introduction

THE rapid urban sprawl in most parts of the world has resulted in the increment of traffic congestion where it demands a high reliable traffic management system for monitoring the traffic flow . Traffic management system has been part of Intelligent Transportation System (ITS) applications that provides crucial input parameters such as lane occupation and average velocity for traffic flow prediction . However, more information about the variation of the velocity distribution of the vehicles is also needed in order to estimate the occurrence of traffic congestion.**Traffic Management System(TMS)** are advanced applications which, without embodying intelligence as such, aim to provide innovative services relating to different modes of transport and traffic management and enable various users to be better informed and make safer, more coordinated, and 'smarter' use of transport networks.

Most of the city traffic is controlled by sensors and cameras shall be installed in big highways and streets. But existence of a system for detecting the size of traffic automatically will be felt. Such systems can allows to extract information from the bigger traffic issue and helps us decide to improve the traffic policy. While analysing traffic on a road traffic network various parameters can be monitored. Such parameters include traffic flow speed profile, distance between vehicles, velocity of vehicles, vehicle classification, etc. They can be measured using various sensors like inductive loops, radars, video sensors, etc. Analysis results can be applied for planning and management of road networks in urban and rural areas including highways.

In transportation,vehicle detection may be defined as a system which is capable of detecting the vehicles and measure traffic parameters such as count,speed etc.Vehicle detection by cameras is one of the most promising non-intrusive technologies for large scale data collection implementation of the advanced traffic control and management schemes .Vehicle detection is also the basis for traffic tracking.

2.Problem definition

Our main objective is to find the average velocity of the route using image processing through Opencv in android .This problem consists of two main modules.

1.Vehicle detection and tracking.

In order to detect the vehicle in an image frame we will have to implement the following algorithms:

a)background subtraction

b)blob tracking algorithmn

2.Velocity Calculation.

Calculate the centroid using image processing

3.Literature Survey:

Several techniques of vehicle tracking system have been investigated for traffic monitoring . Coifman et al. reported the use of network of DSPs combined with Pentium processor to calculate the vehicle speeds and compared the results to 5 minutes average traffic speed. Another approach to measure the speed of the vehicle is to use a complex scalable multiprocessor architecture . Although this work performs good results, it is not feasible because of a very high computational cost.

4.Proposed Approach:

Basically our system is concerned with the velocity estimation that includes vehicle detection and tracking. The proposed system first extracts the moving vehicle using frame differencing techniques. The moving object is tracked and the speed is computed based on the displacement of the object’s centroid using kinematics equations. The calibration between pixel distance and real world distance travelled has been done to overcome perspective distortion. With this method, average velocity can be estimated independent of the position of the vehicles relative to the camera.

Read image sequence in opencv

Implement background subtraction algo

Capture images

Implement optical flow

Describe best path for vehicle

Velocity calculation through image processing

Phase I

First images are captured by camera. Then the sequence of images are taken as input. The processImage function is called and the images are read by using [imread](http://docs.opencv.org/modules/imgcodecs/doc/reading_and_writing_images.html?highlight=imread" \l "imread), after individuating the correct path for the next frame to read.

Syntax

frame = imread(fistFrameFilename);

Phase II

Now we will use the **Background Subtraction algorithm**. Background subtraction (BS) is a common and widely used technique for generating a foreground mask (namely, a binary image containing the pixels belonging to moving objects in the scene) by using static cameras.

As the name suggests, BS calculates the foreground mask performing a subtraction between the current frame and a background model, containing the static part of

scene or, more in general, everything that can be considered as background given the characteristics of the observed scene.

1. Read image sequences by using [imread](http://docs.opencv.org/modules/imgcodecs/doc/reading_and_writing_images.html?highlight=imread" \l "imread);
2. Create and update the background model by using [BackgroundSubtractor](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractor" \l "backgroundsubtractor) class;
3. Get and show the foreground mask by using [imshow](http://docs.opencv.org/modules/highgui/doc/user_interface.html?highlight=imshow" \l "imshow);
4. Save the output by using [imwrite](http://docs.opencv.org/modules/imgcodecs/doc/reading_and_writing_images.html?highlight=imwrite" \l "imwrite) to quantitatively evaluate the results.

Background modeling consists of two main steps:

* 1. Background Initialization;
  2. Background Update.

In the first step, an initial model of the background is computed, while in the second step that model is updated in order to adapt to possible changes in the scene.

* Two different methods are used to generate two foreground masks:
  1. [MOG](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractorMOG#backgroundsubtractormog)
  2. [MOG2](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractorMOG2#backgroundsubtractormog2)

EXPLANATION:

1.First, three Mat objects are allocated to store the current frame and two foreground masks, obtained by using two different BS algorithms.

Mat frame; *//current frame*

Mat fgMaskMOG; *//fg mask generated by MOG method*

Mat fgMaskMOG2; *//fg mask fg mask generated by MOG2 method*

2.Two [BackgroundSubtractor](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractor" \l "backgroundsubtractor) objects will be used to generate the foreground masks.

Ptr<BackgroundSubtractor> pMOG; *//MOG Background subtractor*

Ptr<BackgroundSubtractor> pMOG2; *//MOG2 Background subtractor*

...

*//create Background Subtractor objects*

pMOG = createBackgroundSubtractorMOG(); *//MOG approach*

pMOG2 = createBackgroundSubtractorMOG2(); *//MOG2 approach*

3.The command line arguments are analysed. Image sequences (by choosing the option -img).

**if**(strcmp(argv[1], "-img") == 0) {

*//input data coming from a sequence of images*

processImages(argv[2]);

}

4.Every frame is used both for calculating the foreground mask and for updating the background. If you want to change the learning rate used for updating the background model, it is possible to set a specific learning rate by passing a third parameter to the ‘apply’ method.

*//update the background model*

pMOG->apply(frame, fgMaskMOG);

pMOG2->apply(frame, fgMaskMOG2);

*//read the first file of the sequence*

frame = imread(fistFrameFilename);

**if**(!frame.data){

*//error in opening the first image*

cerr << "Unable to open first image frame: " << fistFrameFilename << endl;

exit(EXIT\_FAILURE);

}

...

*//search for the next image in the sequence*

ostringstream oss;

oss << (frameNumber + 1);

string nextFrameNumberString = oss.str();

string nextFrameFilename = prefix + nextFrameNumberString + suffix;

*//read the next frame*

frame = imread(nextFrameFilename);

**if**(!frame.data){

*//error in opening the next image in the sequence*

cerr << "Unable to open image frame: " << nextFrameFilename << endl;

exit(EXIT\_FAILURE);

}

*//update the path of the current frame*

fn.assign(nextFrameFilename);

To quantitatively evaluate the results obtained, we need to:

* Save the output images;
* Have the ground truth images for the chosen sequence.

In order to save the output images, we can use [imwrite](http://docs.opencv.org/modules/imgcodecs/doc/reading_and_writing_images.html?highlight=imwrite" \l "imwrite)

string imageToSave = "output\_MOG\_" + frameNumberString + ".png";

bool saved = imwrite(imageToSave, fgMaskMOG);

**if**(!saved) {

cerr << "Unable to save " << imageToSave << endl;

}

**MOG:**

BackgroundSubtractorMOG::**BackgroundSubtractorMOG**(int **history**, int **nmixtures**, double **backgroundRatio**, double **noiseSigma**=0)

|  |  |
| --- | --- |
| **Parameters:** | * **history** – Length of the history. * **nmixtures** – Number of Gaussian mixtures. * **backgroundRatio** – Background ratio. * **noiseSigma** – Noise strength. |

## BackgroundSubtractorMOG::operator()

Updates the background model and returns the foreground mask

void BackgroundSubtractorMOG::**operator()**(InputArray **image**, OutputArray **fgmask**, double **learningRate**=0)

Parameters are the same as in **[BackgroundSubtractor::operator()](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractorMOG2" \l "void BackgroundSubtractor::operator()(InputArray image, OutputArray fgmask, double learningRate)" \o "void BackgroundSubtractor::operator()(InputArray image, OutputArray fgmask, double learningRate))**

## BackgroundSubtractorMOG2::operator()

Updates the background model and computes the foreground mask

void BackgroundSubtractorMOG2::**operator()**(InputArray **image**, OutputArray **fgmask**, double **learningRate**=-1)

## BackgroundSubtractorMOG2::getBackgroundImage

Returns background image

void BackgroundSubtractorMOG2::**getBackgroundImage**(OutputArray **backgroundImage**)

See **[BackgroundSubtractor::getBackgroundImage()](http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractorMOG" \l "void BackgroundSubtractor::getBackgroundImage(OutputArray backgroundImage) const" \o "void BackgroundSubtractor::getBackgroundImage(OutputArray backgroundImage) const)**.

Background Subtraction

**Background subtraction**, also known as Foreground Detection, is a technique in the fields of [image processing](http://en.wikipedia.org/wiki/Image_processing) and [computer vision](http://en.wikipedia.org/wiki/Computer_vision) wherein an image's foreground is extracted for further processing (object recognition etc.). Generally an image's regions of interest are objects (humans, cars, text etc.) in its foreground. After the stage of image preprocessing (which may include [image denoising](http://en.wikipedia.org/wiki/Image_denoising), post processing like morphology etc.) object localisation is required which may make use of this technique. Background subtraction is a widely used approach for detecting moving objects in videos from static cameras. The rationale in the approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called “background image”, or “background model”. Background subtraction is mostly done if the image in question is a part of a video stream. Background subtraction provides important cues for numerous applications in computer vision, for example surveillance tracking or human poses estimation. However, background subtraction is generally based on a static background hypothesis which is often not applicable in real environments. With indoor scenes, reflections or animated images on screens lead to background changes. In a same way, due to wind, rain or illumination changes brought by weather, static backgrounds methods have difficulties with outdoor scenes.

### Background mixture models

In this technique, it is assumed that every pixel's intensity values in the video can be modeled using a [Gaussian mixture model](http://en.wikipedia.org/wiki/Gaussian_mixture_model). A simple heuristic determines which intensities are most probably of the background. Then the pixels which do not match to these are called the foreground pixels. Foreground pixels are grouped using 2D [connected component](http://en.wikipedia.org/wiki/Connected-component_labeling) analysis

At any time t, a particular pixel ()'s history is.



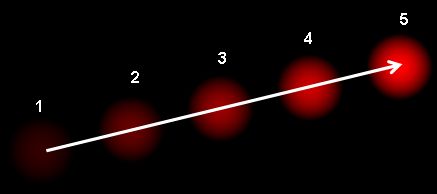
This history is modeled by a mixture of *K* Gaussian distributions:

  
where 

An on-line [K-means](http://en.wikipedia.org/wiki/K-means) approximation is used to update the Gaussians.

Optical Flow –Lucas Kanade method.

* We will use functions like **cv2.calcOpticalFlowPyrLK()** to track feature points in a video.
* Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movemement of object or camera. It is 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second.



It shows a ball moving in 5 consecutive frames. The arrow shows its displacement vector. Optical flow has many applications in areas like :

* Structure from Motion
* Video Compression
* Video Stabilization ...

Optical flow works on several assumptions:

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighbouring pixels have similar motion.

We have seen an assumption before, that all the neighbouring pixels will have similar motion. Lucas-Kanade method takes a 3x3 patch around the point. So all the 9 points have the same motion. We can find  for these 9 points. So now our problem becomes solving 9 equations with two unknown variables which is over-determined. A better solution is obtained with least square fit method. Below is the final solution which is two equation-two unknown problem and solve to get the solution.



So from user point of view, idea is simple, we give some points to track, we receive the optical flow vectors of those points. But again there are some problems. Until now, we were dealing with small motions. So it fails when there is large motion. So again we go for pyramids. When we go up in the pyramid, small motions are removed and large motions becomes small motions. So applying Lucas-Kanade there, we get optical flow along with the scale.

OpenCV provides all these in a single function, **cv2.calcOpticalFlowPyrLK()**. Here, we create a simple application which tracks some points in a video. To decide the points, we use **cv2.goodFeaturesToTrack()**. We take the first frame, detect some Shi-Tomasi corner points in it, then we iteratively track those points using Lucas-Kanade optical flow. For the function **cv2.calcOpticalFlowPyrLK()** we pass the previous frame, previous points and next frame. It returns next points along with some status numbers which has a value of 1 if next point is found, else zero. We iteratively pass these next points as previous points in next step.

Phase III

Through this method we will get the velocity in pixel change per millisecond.Further we will covert this into real world coordinates, that is, in kilometer per hour.(**using camera caliberation**)

5.Activity Chart

Step 1:

Download and install

a) Eclipse(JRE already installed)

b) MinGW

c) OpenCV

Step 2:

Collection of the database ,that is ,the images of the lanes captured through an android camera .

Step 3:

After installing all these we will implement the code of the above described algorithms first of all for detecting and tracking the traffic from images taken as input and then implementing the algorithm to calculate the average velocity of the traffic through image processing and making it efficient.

Time and fixed distance

Velocity Estimation

Background Subtraction (MOG)

image

A Real Time Prediction of Traffic Speed

Foreground Mask

Average velocity

Time taken by vehicle to move from one frame to the other

Frames

Optical Flow

6. Implementation.

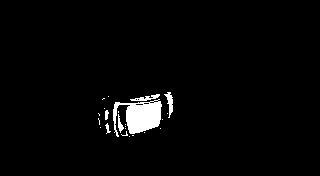
First of all,we implemented Haar classifier for face detection in opencv on visual studio.

It could not give us satisfying results for moving objects as background could not be separated .

Then collecting our own dataset we chose background subtraction algorithm.which as stated above separates background from the foreground.In foreground mask we obtained the moving objects. Video is captured then this video is converted into frames .Implementing Background Subtraction algo we differentiated the moving objects from Background using gaussian mixtures.MOG and MOG2 methods

It is a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. It uses a method to model each background pixel by a mixture of K Gaussian distributions (K = 3 to 5). The weights of the mixture represent the time proportions that those colours stay in the scene. The probable background colours are the ones which stay longer and more static.

While coding, we need to create a background object using the function, **cv2.createBackgroundSubtractorMOG()**. It has some optional parameters like length of history, number of gaussian mixtures, threshold etc. It is all set to some default values. Then inside the video loop, use backgroundsubtractor.apply() method to get the foreground mask.

Once we got the separated moving objects we implemented the optical flow TVL1 method to check the flow of the vehicles which gives the time a vehicle takes to cover the distance in the frames.To compute the distance we implemented Lucas-Kanade Optical flow method which gives the displacement of the points traced in the consecutive frames. The threshold is 60 frames.Total distance covered in that frame is fixed hereby we got velocities after 2 sec then we have taken the average velocity.which is our final output.Frame rate is 30frame per sec.

7.Result:

The output is the average velocity of the traffic on a particular route.Video captured by a static camera.

Accuracy of our project is 64.77.

8.Description of Hardware & Software Requirements:

8.1.Hardware

Camera. 30 frame per second

8.2.Software

OpenCv2.4.9 , Eclipse Juno ,Android platform 4.4.2 kitkat and so on,MinGW,JRE

Language used:

C++,Java(for android)

PLATFORMS USED:

Windows and Android

9.Limitations

Dealing with challenges: The MOG model deals with the movement in the background (MB) due to the multimodality in the representation step. The maintenance step permits to cope up with the gradual illumination changes and the learning rate determines the speed of adaptation to illumination changes but also the speed of the incorporation of background objects moved or inserted and the speed of incorporation of a moving object which stopped. This is one of the disadvantages of the MOG . Another disadvantage is that the pixel-wise aspect prevents to handle some critical situations which can be only detected spatially and temporally. Furthermore, some critical situations need pre-processing or post-processing Another disadvantage is the use of the RGB which can permit to make well shadows detection .

.

10.REFERENCES :

<http://waset.org/publications/701/vehicle-velocity-estimation-for-traffic-surveillance-system>s

<http://docs.opencv.org/trunk/doc/tutorials/video/background_subtraction/background_subtraction.html>

<https://www.behance.net/gallery/Vehicle-Detection-Tracking-and-Counting/4057777>

<http://www.researchgate.net/publication/239525819_Moving_Vehicle_Detection_for_Measuring_Traffic_Count_Using_OpenCV>

<http://docs.opencv.org/master/doc/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html>

<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6524555>