Abstract

Traffic counts, speed and vehicle classification are fundamental data for a variety of transportation projects ranging from transportation planning to modern intelligent transportation systems(ITS). Expressways, highways and roads are getting overcrowded due to increase in number of vehicles. Vehicle detection, tracking, classification and counting is very important, such as highway monitoring, traffic planning, toll collection and traffic flow. Computer Vision based techniques for object detection has been used in this project work and is modified to detect moving vehicles. This is achieved by a series of steps like background subtraction, blob analysis. We have designed and implemented the software using Visual C++ with OpenCV to realize the real-time automatic object detection and vehicle counting.

Chapter 1

Introduction

Traffic counts, speed and vehicle classification are fundamental data for a variety of transportation projects ranging from transportation planning to modern intelligent transportation systems. Still Traffic Monitoring and Information Systems related to classification of vehicles rely on sensors for estimating traffic parameters. Currently, magnetic loop detectors are often used to count vehicles passing over them. In recent year, as the result of the increase in vehicle traffic, many problems have appeared. For example, traffic accidents, traffic congestion and so on. Traffic congestion has been a significantly challenging problem. It has been widely realized that increase of preliminary transportation infrastructure e.g., more pavements, and widened road, have not been able to relieve city congestion. As a result, attention is drawn towards intelligent transportation system (ITS), like predicting the traffic flow based on monitoring the activities at traffic intersections for detecting congestion. Automatic detecting vehicles in video surveillance data is a very challenging problem in computer vision with important practical applications, such as traffic analysis and security. Vehicle detection and counting is important in computing traffic congestion on highways. A system like the one proposed here can provide important data for a design. The main objective of our study is to develop methodology for automatic vehicle detection and its counting on highways.

A system has been developed to detect and count dynamic objects efficiently. Intelligent visual surveillance for road vehicles is a key component for developing autonomous intelligent transportation systems. We present a system for

detecting and tracking vehicles in surveillance video which uses segmentation with initial background elimination using image subtraction and use this subtracted frame to get the blob and track these blobs and check if it crosses a referenced line.

1.1 Motivation

We plan to develop a system which can have important practical applications such as to monitor activities at traffic intersections for detecting congestion, and then predict the traffic flow which assists in regulating traffic. Manually reviewing the large amount of data generated is often impractical. Computer Vision based techniques are more suitable because these systems do not disturb traffic during installation and they are easy to modify.

1.2 Purpose

We intend on developing a system such that given a Real-time traffic video from single camera detect and calculate the number of vehicles passing through a reference line. Getting this data on a regular interval can be used to predict traffic in the near future for traffic monitoring. System implemented here is designed and implemented with Visual C++ software with OpenCV to realize the real-time automatic vehicle detection and vehicle counting.

With the help of OpenCV libraries we systematically follow a chain of processes to count the vehicles from an input video taken from a single camera where we draw a reference line and it counts the number of vehicles passing through the line. This can be done for both ways i.e. vehicles moving upwards and vehicles moving downwards from the reference point. In the process of achieving the goal the heart of the system is the object detection scheme where we track the

moving vehicles. This object detection can also be used in many aspects for different purposes like tracking random movements of people in a mall, a projectile etc. Automatic vehicle counting can also be used to allot the empty slots in parking systems by counting the number of vehicles entering and leaving the parking area or in bridge monitoring systems. The major problems in video monitoring systems are changing light intensities especially at late evenings and at night, weather changes like foggy atmospheres, rain, smoke etc.

Chapter 2

Related Works

It would be fair to acknowledge that this is not the first time a project like this has been undertaken. There have been very exciting papers and works which we had gone through during the preparation of the project. We would like to direct on some of the research, ongoing in this field of technology.

Over the year's researchers in computer vision have proposed various solutions to the automated tracking problem. These approaches can be classified as follows: Blob Tracking, Active Contour Tracking, 3D-Model Based Tracking, Markov Random Field Tracking, Feature Tracking, Color and Pattern-Based Tracking. These are some common methods for automated tracking.

Tracking moving vehicles in video streams has been an active area of research in computer vision. In a real time, system which is used for measuring traffic parameters uses a feature-based method along with occlusion reasoning for tracking vehicles in congested traffic scenes. To handle occlusions, instead of tracking entire vehicles, sub features are tracked. This approach however is computationally very expensive. Alternatively, in a moving object recognition method which uses an adaptive background subtraction technique to separate vehicles from the background, the background is modeled as a slow time-varying image sequence, which allows it to adapt to changes in lighting and weather conditions. In a related work pedestrians are tracked and counted using a single camera. The images from the input image sequence are segmented using background subtraction.

Two of the most cited papers which were of great help in choosing our approach were:

- S. Gupte, O. Masoud, R. F. K. Martin, and N. P. Papanikolopoulos, Detection and classification of vehicles, in Proc. IEEE Transactions on Intelligent Transportation Systems, vol. 3, no. 1, March 2002.

This paper as the title suggests, presents algorithms for vision-based detection and classification of vehicles in monocular image sequences of traffic scenes recorded by a stationary camera. The most appealing part of the paper is the approach for detecting vehicles which has inspired us to use the Blob tracking approach in our project. Moreover, this paper also talks about classification of vehicles based on the contour size. The algorithms suggested for areas like region tracking, segmentation is in-depth and unique.

- N. K. Kanhere, S. T. Birchfield, W. A. Sarasua, and T. C. Whitney, Real time detection and tracking of vehicle base fronts for measuring traffic

counts and speeds on highways, Transportation Research Record, No. 1993, 2007.

The paper presents a real-time system for automatically monitoring a highway when the camera is relatively low to the ground and on the side of the road. It introduces the necessity of Intelligent Transport System (ITS) and about different object detection techniques which were in practice. The approach followed in the paper (vehicle base front [VBF]) is different from ours but throws light on topics like Background Subtraction and Shadow detection. Like the previous paper this also throws light on classification of vehicles and how it can be useful for developing an ITS. The paper presents very insightful details about Background Subtraction. An extract from the paper is presented below which talks about the topic.

“The basic principle of our method is to modify the background image that is subtracted from the current image (called the current background) so that it looks similar to the background in the current video frame. We update the background by taking a weighted average of the current background and the current frame of the video sequence. However, the current image also contains foreground objects. Therefore, before we do the update we need to classify the pixels as foreground and background and then use only the background pixels from the current image to modify the current background. Otherwise, the background image would be polluted with the foreground objects. The binary object mask is used to distinguish the foreground pixels from the background pixels. The object mask is used as a gating function that decides which image to sample for updating the background. At those locations where the mask is 0 (corresponding to the background pixels), the current image is sampled. At those locations where the mask is 1 corresponding to foreground pixels, the current background is sampled. The result of this is what we call the instantaneous background.”

This operation described diagrammatic form as in Fig. 2.1.

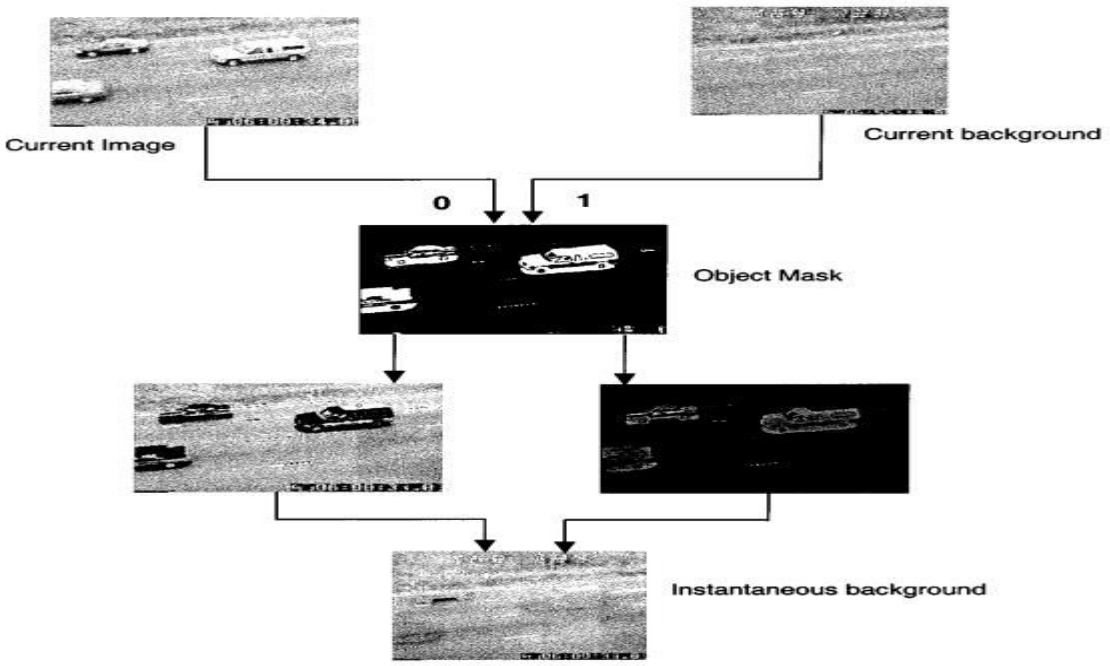

Fig. 2.1 Computation of instantaneous background

Chapter 3

Proposed Methodology

3.1 Overview

The proposed algorithm consists of five major steps in tracking and counting vehicles. Brief description of the steps in the process is given below:

- **Background Subtraction**

Background subtraction is a simple and effective technique for extracting foreground objects from a scene. The process of background subtraction involves initializing and maintaining a background model of the scene.

- **Blob Detection**

  On the subtracted image threshold operation is applied to get the blob. The threshold image is then filtered using dilation and erosion. The contour of the blobs in filtered threshold image is found and drawn. The convex hull function is finally applied which makes the blob well defined.

- **Blob Analysis**

  The detected blobs are then filtered out to be sure that the objects detected are vehicles. Constraints like bounding area, aspect ratio are applied to remove erroneous results.

- **Blob Tracking**

  For each blob, a vector of points (pair of coordinates) of center position is stored in order to predict the next position which is useful for tracking. Use of this in the applied algorithm will be discussed later with proper examples.

- **Vehicle Tracking**

  A reference line is drawn on the video frame depending on the view of camera (orientation of camera). If any blob crosses that line then it gets highlighted and vehicle count is increased by one. Direction of the car is calculated based on the previous frame center position, current frame center position and the referenced line.

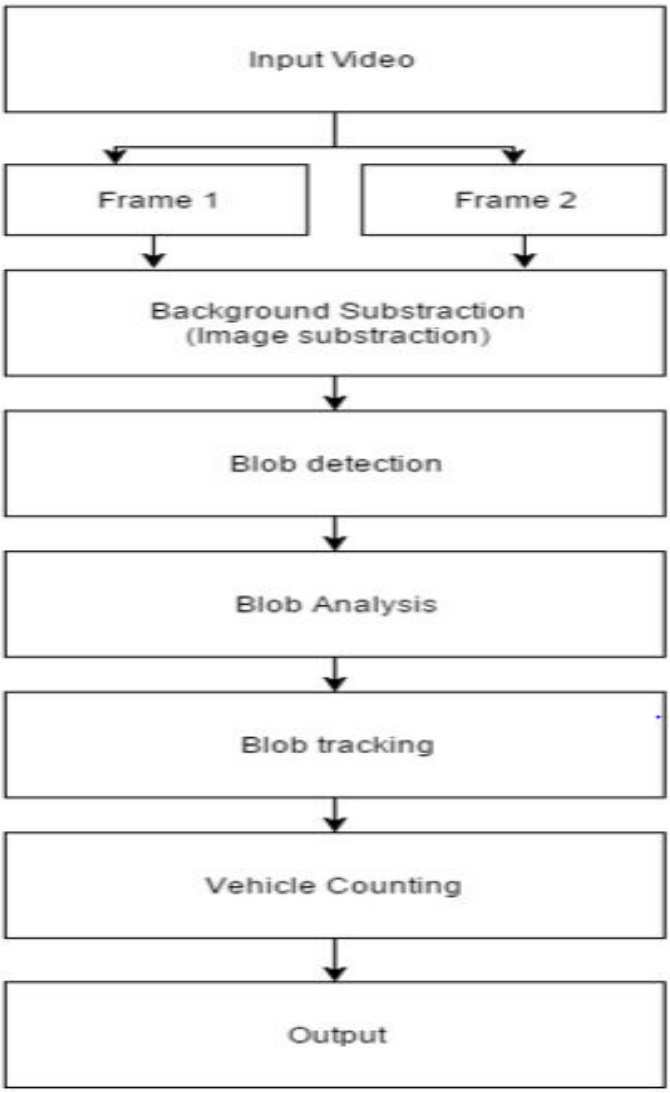

Fig. 3.1 Flow diagram representation of the steps involved in the process

3.2 Background Subtraction

Background subtraction is a simple and effective technique for extracting foreground objects from a scene.

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

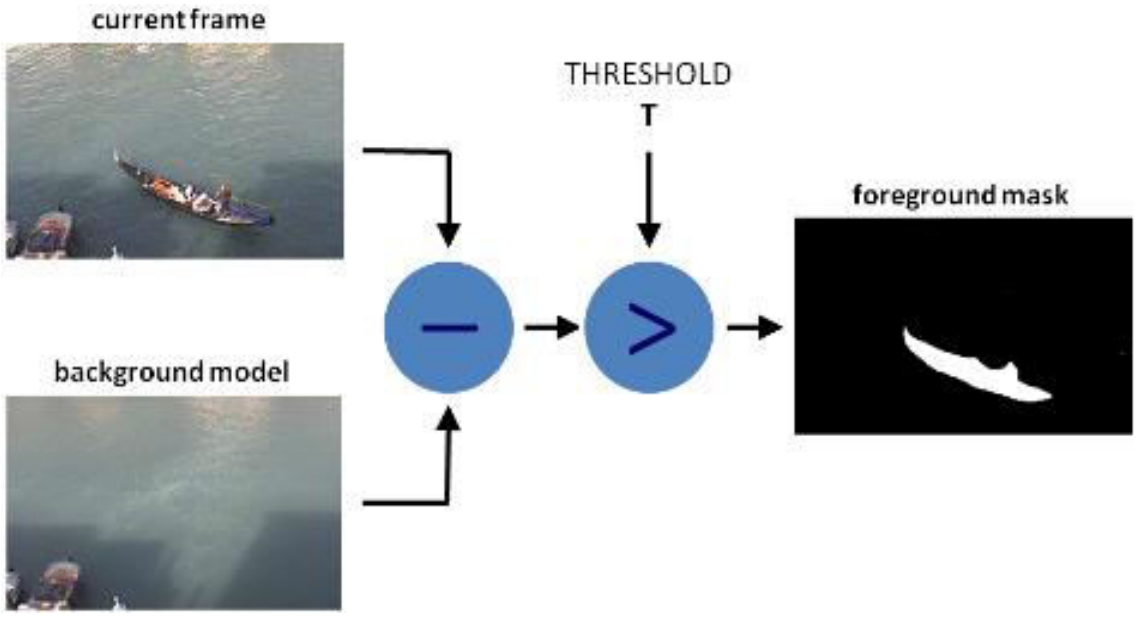

Fig. 3.2 Steps involved in Background Subtraction

The above Fig 3.2 (source: http://docs.opencv.org/3.1.0/d1/dc5/tutorialbackgroundsubtraction.html) explains the process of background subtraction to get a foreground mask. The new term introduced here is Threshold. So, what is Threshold?

In simple terms Threshold can be explained with an example: If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is **cv2.threshold**. First argument is the source image, which should be a grayscale image. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value.

We follow a similar approach for background subtraction. The step by step process of our algorithm is given as:

- Two consecutive frames are taken.
- The images are then converted to gray-scale format.
- Gaussian blur (It is a function in OpenCV used for smoothing of images mainly for reducing noise) is then applied to both the gray-scale images.
- Finally, absolute difference between the two images are taken.

Background Subtraction plays a crucial role in our project because it allows us to extract the main characters in the video for further computation. Fig. 3.3 is a snapshot from our output which displays result when the video is converted to greyscale and the subtracted image is obtained.

Fig. 3.3 Subtracted Image of two Consecutive Frames

A code snippet used in our project for this purpose is given below:

```
std::vector<Blob> currentFrameBlobs;

cv::Mat imgFrame1Copy = imgFrame1.clone(); // clone frame for taking absolute difference
cv::Mat imgFrame2Copy = imgFrame2.clone();

cv::Mat imgDifference;
cv::Mat imgThresh;

cv::cvtColor(imgFrame1Copy, imgFrame1Copy, CV_BGR2GRAY); // convert image to greyscale
cv::cvtColor(imgFrame2Copy, imgFrame2Copy, CV_BGR2GRAY);

cv::GaussianBlur(imgFrame1Copy, imgFrame1Copy, cv::Size(5, 5), 0);
cv::GaussianBlur(imgFrame2Copy, imgFrame2Copy, cv::Size(5, 5), 0);

cv::absdiff(imgFrame1Copy, imgFrame2Copy, imgDifference); // absolute difference of the two frames
cv::imshow("img_substraction", imgDifference); //display

cv::threshold(imgDifference, imgThresh, 30, 255.0, CV_THRESH_BINARY);
```

3.3 Blob Detection

BLOB stands for Binary Large OBject and refers to a group of connected pixels in a binary image. The term "Large" indicates that only objects of a certain size are of interest and that the other "small" binary objects are usually noise. A Blob is a group of connected pixels in an image that share some common property (e.g. grayscale value). To detect the blob in the subtracted images following steps are taken:

3.3.1 Image Threshold

We proceed by separating out regions of an image corresponding to objects which we want to analyze. This separation is based on the variation of intensity between the object pixels and the background pixels. To differentiate the pixels, we are interested in from the rest (which will eventually be rejected), we perform a comparison of each pixel intensity value with respect to a threshold (determined according to the problem to solve). Once we have separated properly the important pixels, we can set them with a determined value to identify them.
On the subtracted image threshold operation is applied to get the blob, the result can be seen in Fig 3.4. In this case, the region of interest is the vehicle or blob.

3.4 Threshold image of the Subtracted Image

3.3.2 Image Filtering

The threshold image is then filtered using morphological operations. Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation. Two basic morphological operators are Erosion and Dilation. Fig 3.5 is given as an input on which we will apply erosion and dilation.

Erosion The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object. So, what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). All the pixels near boundary will be discarded depending upon

the size of kernel. So, the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises, detach two connected objects etc.

Dilation It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So, it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So, we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Erosion and Dilation are done to view the object clearly. By erosion and dilation, the boundary region of vehicles can be clearly seen. Then mask of the image can be taken for vehicle detection and classification based on the shape features. The Dilation process is performed by laying the **structuring element B** on the **image A** and sliding it across the image in a manner similar to convolution. Dilation adds pixels to the boundaries of objects in an image by finding the local maxima and creates the output matrix from these maximum values as shown in equation (1):

$$A \oplus B = \left\{ Z \mid (\hat{B})_z \cap A \neq \varphi \right\}$$

The erosion process is similar to dilation, but here pixels are tuned to white, not black. Erosion removes pixels on object boundaries in an image by finding the local minima and creates the output matrix from these minimum values as shown in equation (2).

$$A \Theta B = \left\{ Z \mid (B)_z \subseteq A \right\}$$

Fig. 3.5 Actual Image(Input) Fig. 3.6 Image after Erosion Fig.3.7 Image after Dilation

3.3.3 Contours

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. Contour tracing is one of many preprocessing techniques performed on digital images in order to extract information about their general shape. Once the contour of a given pattern is extracted, it's different characteristics will be examined and used as features which will later on be used in pattern classification. Therefore, correct extraction of the contour will produce more accurate features which will increase the chances of correctly classifying a given pattern. In our procedure, before finding contours, threshold is applied. In OpenCV, finding contours is like finding white object from black background. Using contours, we can create a list of points for each "patch" or "blob" of white in the image. Then we can do whatever you want with these points like figure out the center of the patch, or calculate its approximate size etc. Fig 3.8 is representation of contour we get during our intermediate step in tracking the vehicles. Mask is used in this concept.

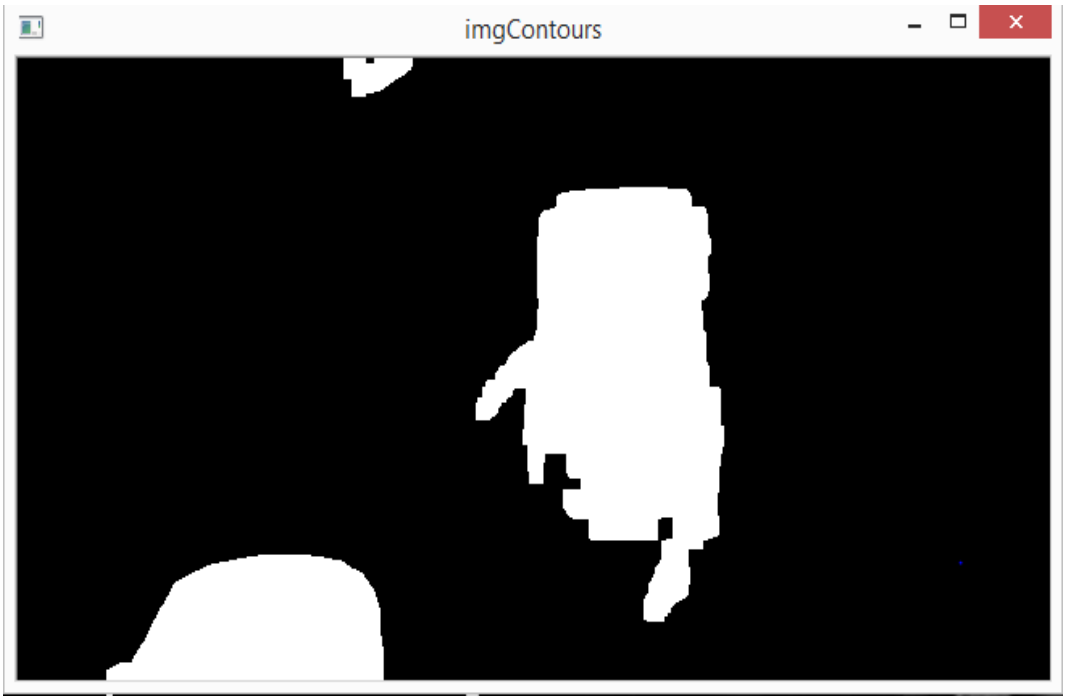

Fig. 3.8 Image showing the contour of the filtered image

3.3.4 Convex Hull

In Computer Vision and Math jargon, the boundary of a collection of points or shape is called a “hull”. A boundary that does not have any concavities is called a **Convex Hull**. The convex hull or convex envelope of a set X of points in the Euclidean plane or in a Euclidean space is the smallest convex set that contains X. For instance, when X is a bounded subset of the plane, the convex hull may be visualized as the shape enclosed by a rubber band stretched around X. Formally, the convex hull may be defined as the intersection of all convex sets containing X or as the set of all convex combinations of points in X. In Fig. 3.9 we use an example for the elastic band analogy of convex hull for a set of finite points.

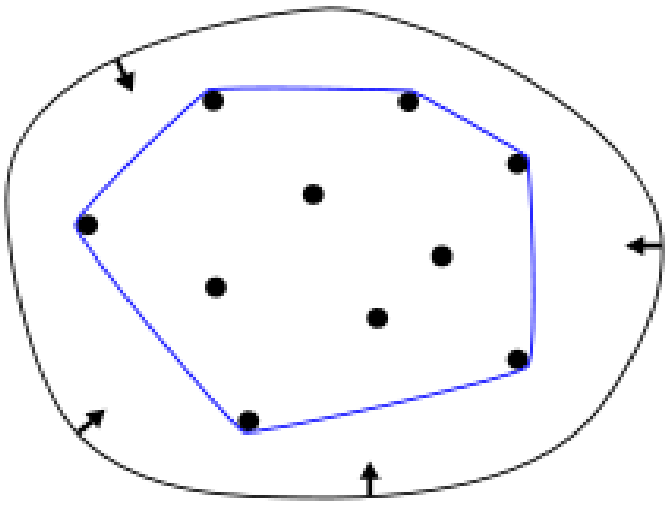

Fig. 3.9 Convex Hull of a finite set

The convex hull function is applied on the blobs which makes the blobs more defined. Using the OpenCV functions available we can calculate the bounding areas of the contours tracked and further apply them for efficient car counting. The constraints can be further applied while tracking vehicles in order to eliminate multiple counts and lower the effect of noise while calculating.

Applying convex hull function on our blob we get rectangular contours which helps our further calculations. A snapshot is provided in Fig. 3.10, 3.11 shows the real-time output in our project.

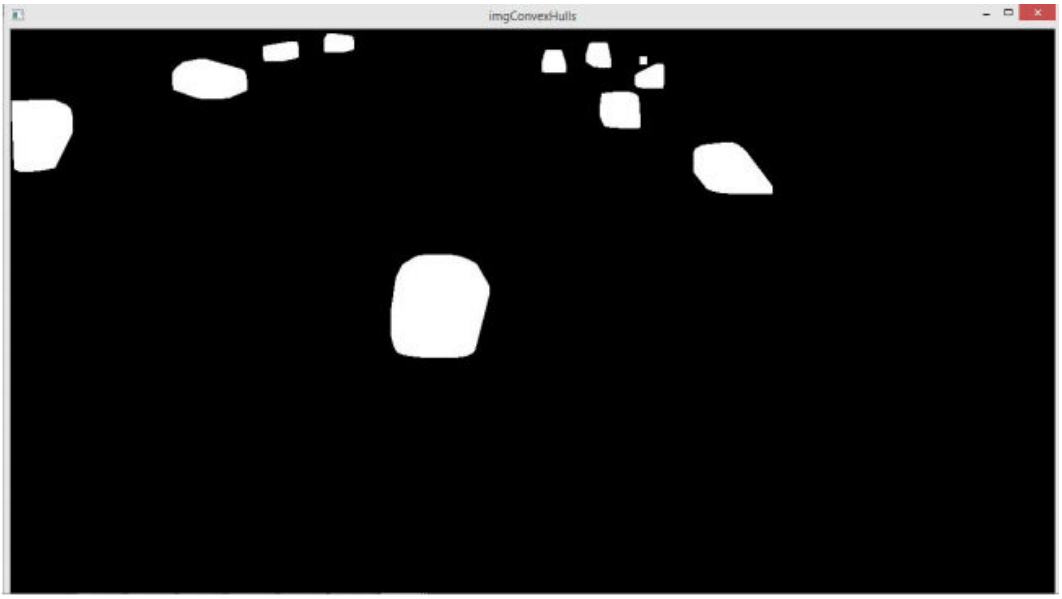

Fig. 3.10 Image showing blobs after convex hull function is applied

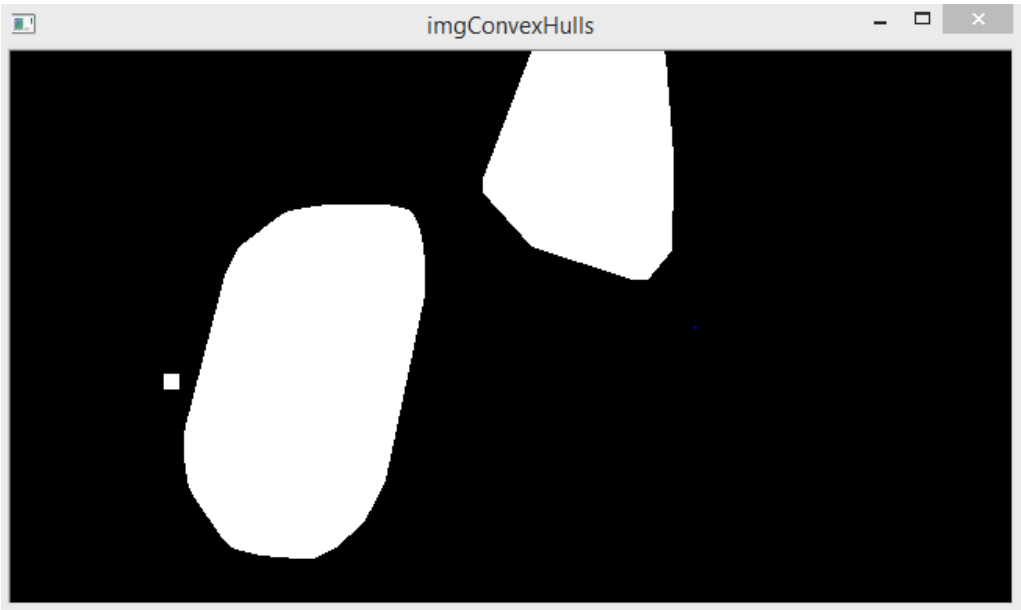

Fig. 3.11 Image showing blobs after convex hull function is applied

Snippet of code is given below which displays implementation of image filtering, drawing contours and extracting convex hulls.

```
cv::GaussianBlur(imgFrame1Copy, imgFrame1Copy, cv::Size(5, 5), 0);
cv::GaussianBlur(imgFrame2Copy, imgFrame2Copy, cv::Size(5, 5), 0);

cv::absdiff(imgFrame1Copy, imgFrame2Copy, imgDifference); // absolute difference of the two frames
cv::imshow("img_substraction", imgDifference); //display

cv::threshold(imgDifference, imgThresh, 30, 255.0, CV_THRESH_BINARY);

cv::imshow("imgThresh", imgThresh);

cv::Mat structuringElement3x3 = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
cv::Mat structuringElement5x5 = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(5, 5));
cv::Mat structuringElement7x7 = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(7, 7));
cv::Mat structuringElement15x15 = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(15, 15));

for (unsigned int i = 0; i < 2; i++) {
    cv::dilate(imgThresh, imgThresh, structuringElement5x5);
    cv::dilate(imgThresh, imgThresh, structuringElement5x5);
    cv::erode(imgThresh, imgThresh, structuringElement5x5);
}

cv::Mat imgThreshCopy = imgThresh.clone();

std::vector<std::vector<cv::Point> > contours;

cv::findContours(imgThreshCopy, contours, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);

drawAndShowContours(imgThresh.size(), contours, "imgContours");

std::vector<std::vector<cv::Point> > convexHulls(contours.size());

for (unsigned int i = 0; i < contours.size(); i++) {
    cv::convexHull(contours[i], convexHulls[i]);
}

drawAndShowContours(imgThresh.size(), convexHulls, "imgConvexHulls");
```

```
// drawing contours to output

void drawAndShowContours(cv::Size imageSize, std::vector<std::vector<cv::Point> > contours, std::string strImageName) {
    cv::Mat image(imageSize, CV_8UC3, SCALAR_BLACK);

    cv::drawContours(image, contours, -1, SCALAR_WHITE, -1);

    cv::imshow(strImageName, image);
}


void drawAndShowContours(cv::Size imageSize, std::vector<Blob> blobs, std::string strImageName) {

    cv::Mat image(imageSize, CV_8UC3, SCALAR_BLACK);

    std::vector<std::vector<cv::Point> > contours;

    for (auto &blob : blobs) {
        if (blob.blnStillBeingTracked == true) {
            contours.push_back(blob.currentContour);
        }
    }

    cv::drawContours(image, contours, -1, SCALAR_WHITE, -1);

    cv::imshow(strImageName, image);
}
```

3.4 Blob Analysis

In this module, our main aim is to analyze the contour and calculate the bounding areas for the rectangles. The constraints are calculated for detection of the vehicles and to distinguish between vehicles and noises which may be present because of camera errors. Different limits are set for different features of the blob.

- Bounding rectangle area it is set to more than 800.
- Width and height it is set to more than 50.
- Diagonal length it is set to more than 80.
- Aspect Ratio(width/height) is set to less than 4 and greater than 0.2

The blobs satisfying the above constraints are then added to the list of current frames. These detected blobs are considered for tracking and if a blob passes the reference line the count for the number of vehicles is increased by one. The count of vehicle is shown on top-right corner of the screen.

Snippet of code is given below which shows how the constraints are being implemented in the real project.

```
drawAndShowContours(imgThresh.size(), convexHulls, "imgConvexHulls");
// constraints to detect vehicles

for (auto &convexHull : convexHulls) {
    Blob possibleBlob(convexHull);

    if (possibleBlob.currentBoundingRect.area() > 800 &&
        possibleBlob.dblCurrentAspectRatio > 0.2 &&
        possibleBlob.dblCurrentAspectRatio < 4.0 &&
        possibleBlob.currentBoundingRect.width > 30 &&
        possibleBlob.currentBoundingRect.height > 30 &&
        possibleBlob.dblCurrentDiagonalSize > 60.0 &&
        (cv::contourArea(possibleBlob.currentContour) / (double)possibleBlob.currentBoundingRect.area()) > 0.50) {
        currentFrameBlobs.push_back(possibleBlob);
    }
}

//drawAndShowContours(imgThresh.size(), currentFrameBlobs, "imgCurrentFrameBlobs");

if (blnFirstFrame == true) {
    for (auto &currentFrameBlob : currentFrameBlobs) {
        blobs.push_back(currentFrameBlob);
    }
}
else {
    matchCurrentFrameBlobsToExistingBlobs(blobs, currentFrameBlobs);
}

//drawAndShowContours(imgThresh.size(), blobs, "imgBlobs");
```

3.5 Blob Tracking

For each blob a vector of points (pair of coordinates) of center position is stored. The coordinates are stored after we filter out the required blobs from the erroneous ones. Algorithm used for tracking the blob given the coordinates of the blob in previous frame:

• Next position of the center point is predicted based on the center points of the previous frames.

• If the number of previous frame is one then the predicted position of the center point is same as the previous position.

• If the number of previous frame is equal to two then the predicted coordinates are the previous frame coordinates in addition with the difference of the last two frames.

• If the number of previous frame is greater than or equal to 3 then a weight of two is given to the difference of the last two frames and a weight of one is given to the difference to the 2nd last and 3rd last frame. It is then summed and average over. This is then added to the coordinates of the last frame to predict the next center point coordinates.

• For each blob in the current frame its predicted center position is then compared to the center position of blob of previous frames.

• It takes the one with the lowest difference.

• If the difference is lower than half the diagonal size then its center position is added to the list of center points of that blob.

• If not then it is considered as a new blob.

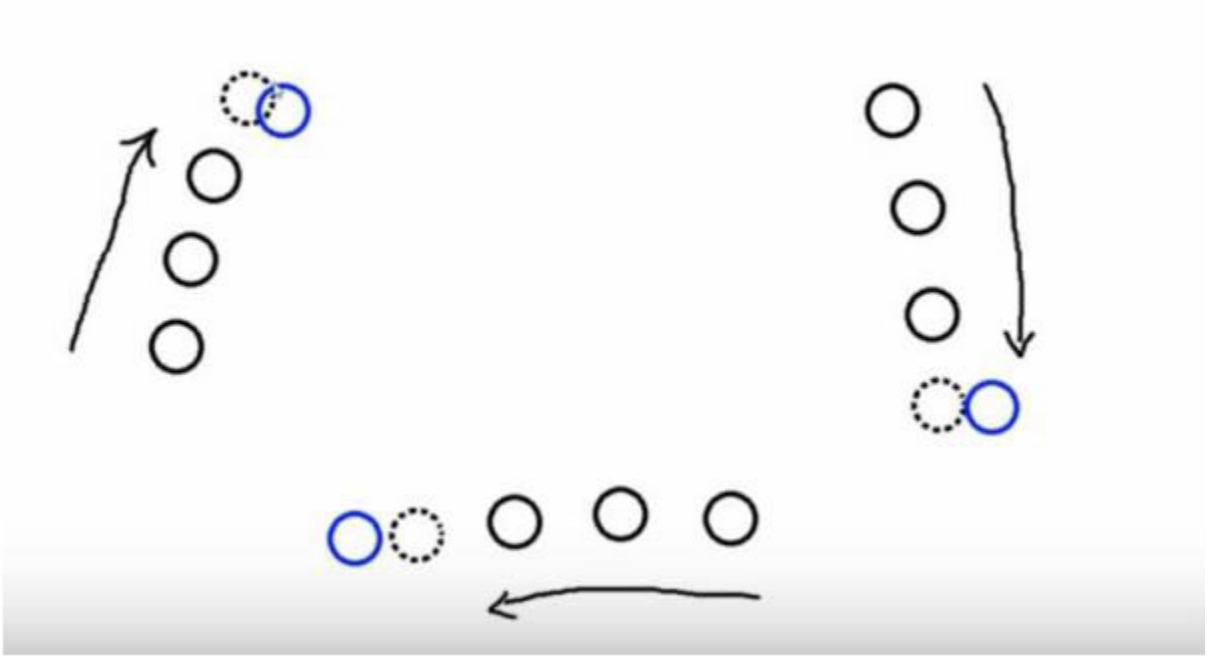

Fig. 3.12 Image showing prediction algorithm

The algorithm stated above can be explained with Fig. 3.12. It shows three object moving simultaneously. The dotted circle is the predicted next position of the circle respectively which is calculated as per the algorithm stated and the blue ones are the next positions of circles respectively. As algorithm states, each of the predicted next position is compared with the blue ones and the one which is closet is found and if its distance is less than half the diagonal size then the blue one is added to the one with the closest predicted next position. If the existing circle is not matched with any circle or is not a new circle for more than 5 frames then it is considered as nonexistent.

Snippet of code is given below which explains how the concept is implemented in our project.

```
void matchCurrentFrameBlobsToExistingBlobs(std::vector<Blob> &existingBlobs, std::vector<Blob> &currentFrameBlobs) {
    for (auto &existingBlob : existingBlobs) {

        existingBlob.blnCurrentMatchFoundOrNewBlob = false;

        existingBlob.predictNextPosition();
    }
    for (auto &currentFrameBlob : currentFrameBlobs) {

        int intIndexOfLeastDistance = 0;
        double dblLeastDistance = 100000.0;

        for (unsigned int i = 0; i < existingBlobs.size(); i++) {

            if (existingBlobs[i].blnStillBeingTracked == true) {

                double dblDistance = distanceBetweenPoints(currentFrameBlob.centerPositions.back(), existingBlobs[i].predictedNextPosition);

                if (dblDistance < dblLeastDistance) {
                    dblLeastDistance = dblDistance;
                    intIndexOfLeastDistance = i;
                }
            }
        }

        if (dblLeastDistance < currentFrameBlob.dblCurrentDiagonalSize * 0.5) {
            addBlobToExistingBlobs(currentFrameBlob, existingBlobs, intIndexOfLeastDistance);
        }
        else {
            addNewBlob(currentFrameBlob, existingBlobs);
        }
    }

    for (auto &existingBlob : existingBlobs) {

        if (existingBlob.blnCurrentMatchFoundOrNewBlob == false) {
            existingBlob.intNumOfConsecutiveFramesWithoutAMatch++;
        }

        if (existingBlob.intNumOfConsecutiveFramesWithoutAMatch >= 5) {
            existingBlob.blnStillBeingTracked = false;
        }

    }

}
```

The algorithm stated above can be explained with the figure shown above. It shows three object moving simultaneously. The dotted circle is the predicted next position of the circle respectively which is calculated as per the algorithm stated and the blue ones are the next positions of circles respectively. As algorithm states, each of the predicted next position is compared with the blue ones and the one which is closet is found and if its distance is less than half the diagonal size then the blue one is added to the one with the closest predicted next position. If the existing circle is not matched with any circle or is not a new circle for more than 5 frames then it is considered as nonexistent.

Snippet of code is given below which shows how rectangular blocks are drawn around the vehicles being tracked.

```
void drawBlobInfoOnImage(std::vector<Blob> &blobs, cv::Mat &imgFrame2Copy) {

    for (unsigned int i = 0; i < blobs.size(); i++) {

        if (blobs[i].blnStillBeingTracked == true) {
            cv::rectangle(imgFrame2Copy, blobs[i].currentBoundingRect, SCALAR_RED, 2);

            int intFontFace = CV_FONT_HERSHEY_SIMPLEX;
            double dblFontScale = blobs[i].dblCurrentDiagonalSize / 60.0;
            int intFontThickness = (int)std::round(dblFontScale * 1.0);

            //cv::putText(imgFrame2Copy, std::to_string(i), blobs[i].centerPositions.back(), intFontFace,
        }
    }
}
```

Fig. 3.13 Vehicle Tracking using the proposed algorithm

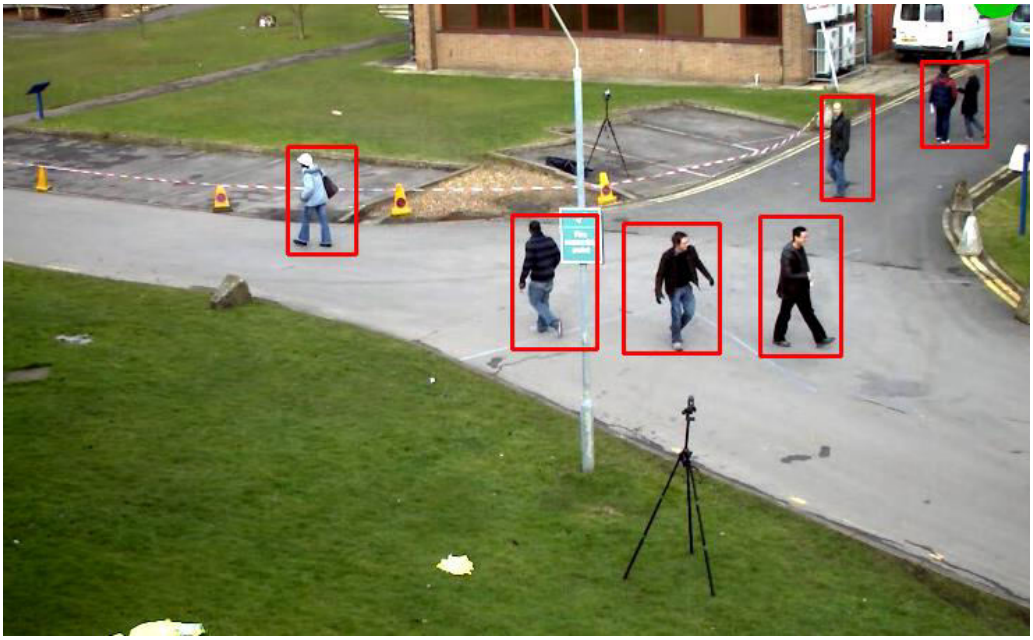

Fig. 3.14 Object tracking algorithm performance for tracking people in random motion

3.6 Vehicle Counting

A green reference line is drawn on the video frame depending on the view of camera (orientation of camera). In this case, the referenced line is drawn horizontally. If any blob(vehicle) crosses that line then it gets highlighted in red and vehicle count is increased by one. The number of vehicles crossing that referenced line is shown on the top right corner of the output video. Direction of the car is calculated based on the previous frame center position, current frame center position and the referenced line. The project supports both movements (towards the camera and away from the camera) of cars, only a slight change of code makes it compatible either ways.

We have collected sample videos from a nearby bridge from a higher angle both ways from the same camera. The results in both cases are satisfactory and very close to perfection. Fig. 3.15 and Fig. 3.16 shows the output for cars moving away and cars moving towards the frame respectively. For either of the cases the reference line is adjusted to fit the frame.

Snippet of code given below explains how the count is displayed on the output video.

```
void drawCarCountOnImage(int &carCount, cv::Mat &imgFrame2Copy) {

    int intFontFace = CV_FONT_HERSHEY_SIMPLEX;
    double dblFontScale = (imgFrame2Copy.rows * imgFrame2Copy.cols) / 200000.0;
    int intFontThickness = (int)std::round(dblFontScale * 3.0);

    cv::Size textSize = cv::getTextSize(std::to_string(carCount), intFontFace, dblFontScale, intFontThickness, 0);

    cv::Point ptTextBottomLeftPosition;

    ptTextBottomLeftPosition.x = imgFrame2Copy.cols - 1 - (int)((double)textSize.width * 1.25);
    ptTextBottomLeftPosition.y = (int)((double)textSize.height * 1.25);

    cv::putText(imgFrame2Copy, std::to_string(carCount), ptTextBottomLeftPosition, intFontFace, dblFontScale, SCALAR_GREEN, intFontThickness);

}
```

Snippet of code given below explains how the car count increases when a car passes through the reference line. Logic for both the cases are given i.e. when car moves upwards and when the car moves downwards.

```
bool checkIfBlobsCrossedTheLine(std::vector<Blob> &blobs, int &intHorizontalLinePosition, int &carCount,int &carCount2) {
    bool blnAtLeastOneBlobCrossedTheLine = false;

    for (auto blob : blobs) {

        if (blob.blnStillBeingTracked == true && blob.centerPositions.size() >= 2) {
            int prevFrameIndex = (int)blob.centerPositions.size() - 2;
            int currFrameIndex = (int)blob.centerPositions.size() - 1;

    // ********************Logic for car moving away from the frame*********************************//
    //************************************UPWARD MOVEMENT*****************************************//
    //      if (blob.centerPositions[prevFrameIndex].y > intHorizontalLinePosition && blob.centerPositions[currFrameIndex].y <= intHorizontalLinePosition) {
    //          carCount++;


     //********************Logic for car moving towards the frame*********************************//
     //************************************DOWNWARD MOVEMENT*****************************************//

             if (blob.centerPositions[prevFrameIndex].y < intHorizontalLinePosition && blob.centerPositions[currFrameIndex].y >= intHorizontalLinePosition) {
                 carCount++;

             std::cout << blob.currentBoundingRect.area() << "\n";
             /*if (blob.currentBoundingRect.area() > 20000) {
                 std::cout << "More than 600\n";
             }
             else if (blob.currentBoundingRect.area() < 20000) {
                 std::cout << "Less than 600\n";
             }*/
             blnAtLeastOneBlobCrossedTheLine = true;
         }
         /*if (blob.centerPositions[prevFrameIndex].y <= intHorizontalLinePosition && blob.centerPositions[currFrameIndex].y > intHorizontalLinePosition) {
             carCount2++;
             blnAtLeastOneBlobCrossedTheLine = true;
             std::cout << carCount2 << "\n";
         }*/
     }

    }

    return blnAtLeastOneBlobCrossedTheLine;
}
```

3.7 Results

The system was implemented using Visual Studio 2015 with OpenCV 3.2.0. We tested our system with sample videos taken from a footbridge (near Laketown, Kolkata). The videos were taken from the same camera and the system was tested for both the cases i.e. for vehicles moving upwards (away from the camera) and downwards (towards the camera).

The result for vehicle moving upwards is satisfactory giving 95% accuracy in average case and for vehicle moving downward the efficiency drops to 84% in average case. Errors in detection were most frequently due to occlusions and/or poor segmentation. Also, when multiple vehicles occlude each other, they are often detected as a single vehicle. However, if the vehicles later move apart, the tracker is robust enough to correctly identify them as separate vehicles. The orientation of the camera affects the results. We intend to further test the system on more complex scenes and under a wider variety of illumination and weather conditions. Output has been shown in Fig. 3.14 and 3.15 respectively.

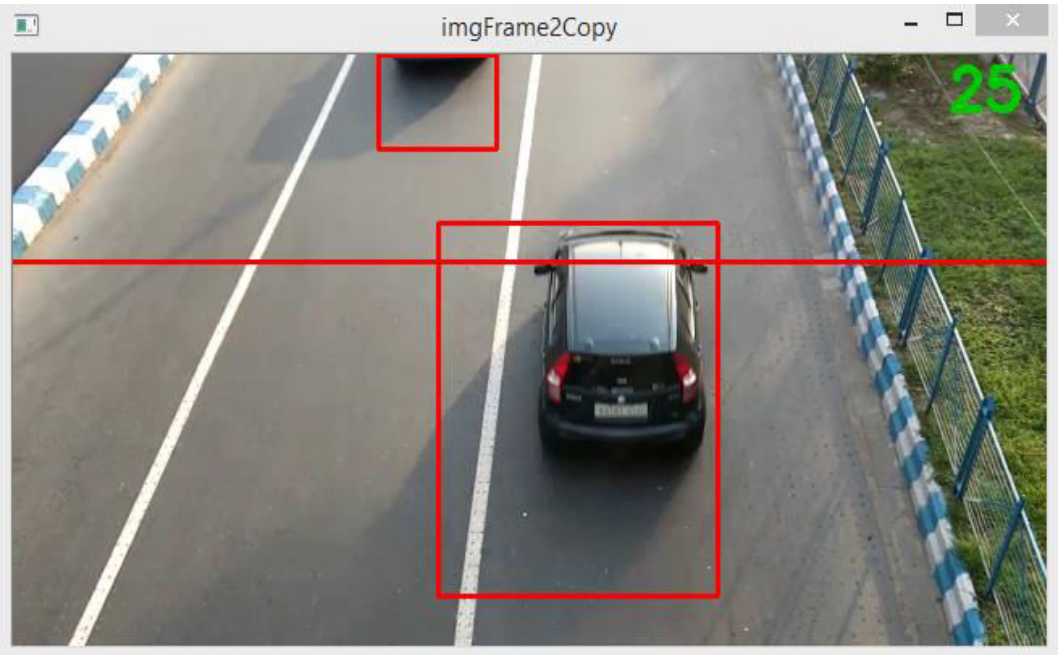

Fig. 3.15: Image showing the snapshot of the output video for vehicles moving upward.

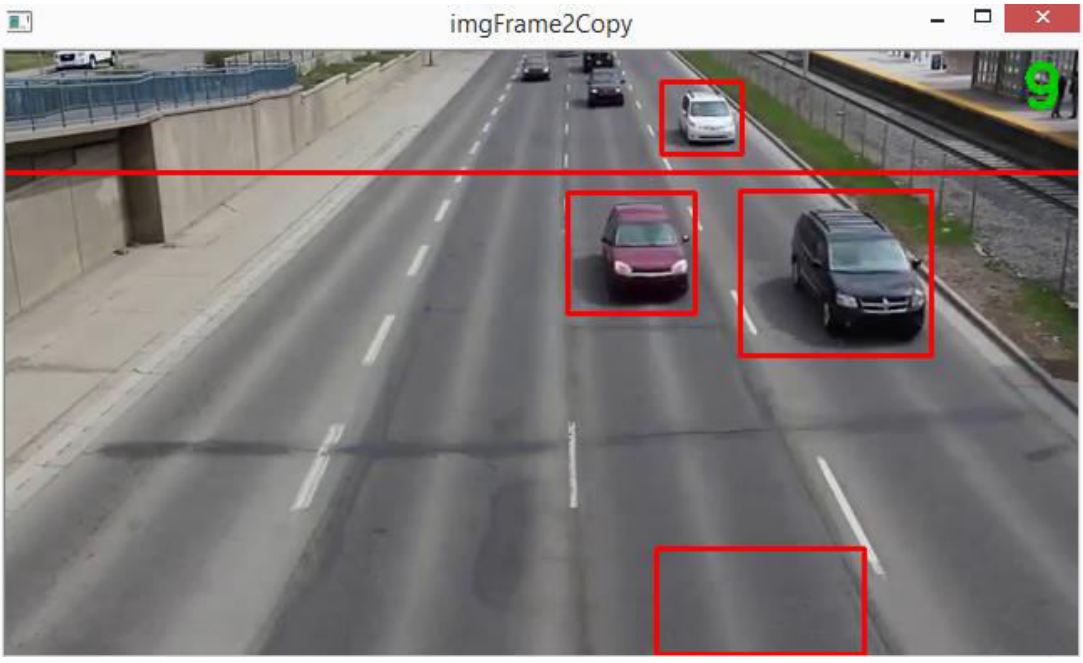

Fig. 3.16: Image showing the snapshot of the output video for vehicles moving downwards.

Chapter 4

Future Works

Vehicle detection and counting is a very interesting topic to work on and we have summed up what our project can do, but there are many scopes of improvements and add on which can make it a sophisticated technology for practical usage. In the course of making this project we have learnt a lot about OpenCV which has given us insight about numerous algorithms for image processing. These can further be used to improve the software.

We have so far focused on only tracking the vehicles and counting from a traffic video. Vehicle detection and further classification can be considered as the next milestone because it can add on a very unique and practical feature for the software. Vehicle classification is an inherently hard problem as for classification in further categories we cannot just depend on the aspect ratio and area of our contours but other parameters have to be kept in mind. Fig. 4.1 shows the brief idea for growing the project on the grounds we have built it on.

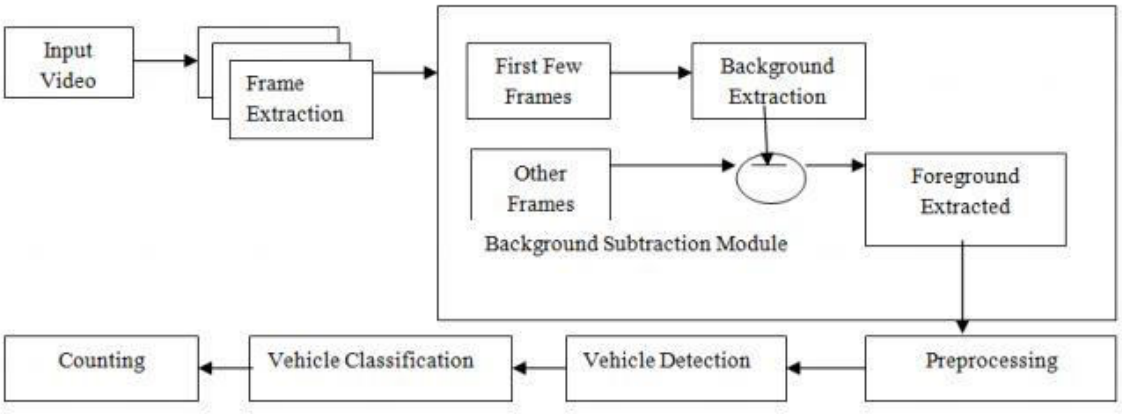

Fig. 4.1 Block Diagram for Vehicle Detection and Classification

As discussed previously the tracking system can also be improved and can be used for relevant areas of study like “Discrete Continuous optimization for muti-target tracking”. Also, we can get live data from a running camera and get the data at regular intervals. These can be used to draw a graph in which node represents the traffic of that road and edge representing the dependencies of the road with different roads. We can also upgrade our system which not only classifies the vehicles but also tracks the speed. This can be applied to detect over speeding vehicles on highways and a License plate recognition system can be worked out which basically involves tracking. Other proposals can also be derived based on the data we fetch and how to use it for developing an ITS.

Conclusion

In this report, a system for Vehicle detection and counting has been presented. Vehicle Detection and Counting is necessary to establish an enriched information platform and improve the quality of intelligent transportation systems. We have presented a system which is implemented in Visual Studio 2015 with the help of OpenCV 3.2.0. The system is general enough to be capable of detecting, tracking vehicles while requiring only minimal scene-specific knowledge. Image from video sequence are taken to detect moving vehicles, so that background is extracted from the images. The extracted background is used in subsequent analysis to detect moving vehicles. The system is implemented using OpenCV image development kits and experimental results are demonstrated from real-time video taken from single camera. Further improvements have been proposed which we were unable to fulfil due to lack of time and lack practical knowhow to develop a more sophisticated technology which can be of a greater practical usage like Vehicle classification, collect regular data for predicting traffic congestion, speed tracking of vehicles, license plate recognition. About the system, the user should provide with only the traffic video and the system calculates the count and also outputs the blob positions detected on the terminal. Finally we would like to acknowledge that in the journey of making this project we have learnt a lot about the Computer Vision library and it has been a great experience learning about object tracking and detection.

References

[1] N. K. Kanhere, S. T. Birchfield, W. A. Sarasua, and T. C. Whitney, Real time detection and tracking of vehicle base fronts for measuring traffic counts and speeds on highways, Transportation Research Record, No. 1993, 2007.

[2] S. Gupte, O. Masoud, R. F. K. Martin, and N. P. Papanikolopoulos, Detection and classification of vehicles, in Proc. IEEE Transactions on Intelligent Transportation Systems, vol. 3, no. 1, March 2002.

[3] Andrew, A. M. (1979),” Another efficient algorithm for convex hulls in two dimensions”, Information Processing Letters , 9 (5): 216219, doi:10.1016/00200190(79)90072-3.

[4] A. Andriyenko, K. Schindle, S. Roth (2012) “discrete continous optimization for multi target tracking”. 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[5] OpenCV Documentation and tutorial.
http://docs.opencv.org/2.4.13/

[6] Blob detection concept from online portal LearnOpenCV
https://www.learnopencv.com/blob-detection-using-opencv-python-c/