EE 417 TERM PROJECT REPORT

Detecting Heavy Vehicles and Their Speeds

19 JANUARY 2023

OSMAN MELİH KURT
OKAN ARIF GUVENKAYA

osmankurt@sabanciuniv.edu
okanarif@sabanciuniv.edu

26497
26780

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

Table of Contents

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

INTRODUCTION

1. Importance of the Problem

Traffic control and the enforcement of traffic rules are crucial for maintaining the safety and efficiency of our roads. With the increasing number of vehicles on the road, traffic congestion and accidents have become major concerns. To address these issues, traffic control systems have been implemented to manage the flow of vehicles and ensure that drivers comply with traffic rules. These systems include traffic lights, signs, and cameras that monitor traffic and enforce rules such as speed limits and traffic signals. Additionally, the detection of heavy vehicles, such as trucks and buses, is becoming increasingly important as they can pose a greater risk to other vehicles and road users. The use of computer vision technology to detect and track heavy vehicles can help to improve safety and reduce the risk of accidents.

Computer vision is a field of artificial intelligence that deals with the interpretation of visual data, such as images and videos. By using cameras and image processing algorithms, computer vision systems can detect and track vehicles on the road in real-time. This technology can be used to detect heavy vehicles, such as trucks and buses, and monitor their movements. This can help to improve traffic flow and reduce the risk of accidents caused by these larger vehicles.

One of the important applications of computer vision technology in traffic control is the enforcement of traffic rules. By using cameras and image processing algorithms, computer vision systems can detect violations such as running red lights, speeding, and illegal lane using. This information can then be used to enforce traffic rules and issue fines or penalties to

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

violators. This can help to improve compliance with traffic rules and reduce the risk of accident.

In conclusion, enforcement of traffic rules is essential for maintaining the safety and efficiency of our roads. The use of computer vision technology to detect and track heavy vehicles can help to improve safety and reduce the risk of accidents. With the increasing number of vehicles on the road, the use of computer vision technology in traffic control is becoming increasingly important. (Buch et al., 2011)

2. Problem Definition

Computer vision and image processing are important techniques commonly used in traffic violation detection systems as explained in the previous section. Correspondingly, the project aim is the detection of traffic violations for a specific type of vehicle using computer vision. More specifically, there are two different aims of this project. One of them is the detection of heavy vehicles that exceed the speed limit and the other one is the detection of heavy vehicles that use the wrong lane. For this purpose, heavy vehicles that don't obey the rules will be marked to make roads safer.

METHODS

1- Problem Formulation

This project was created step by step. The implementation of these steps will be given in the next sections. The following steps were followed:

a. Finding Video and Converting to Grayscale

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

A proper video is required to be found. Less vibration in camera means more accurate detection results. To increase computational time a specific fraction of video will be used. First 109 frames were selected for this purpose.

b. Line Extraction Using Hough Lines

To understand the current lanes of the vehicles and detection of using wrong lanes, it is required to do Hough Lines

c. Using Foreground Detector to Detect the Moving Objects

The foreground needs to be detected and separated from the background. This is kind of first step in the detection of moving objects.

d. Noise Elimination for Foreground Detector

There are lots of noises and external disturbances as the shadow. To detect moving heavy vehicles clearly by using Blob Analysis, it is required to apply a good noise-elimination filter.

e. Blob Analysis

To detection of the moving object and take them into a rectangle Blob analysis is required. With help of the blop analysis moving regions are connected and the detection becomes better.

f. Determining the Speed of Objects

By using some of the speed calculation techniques, the speed of moving objects can be determined.

g. Detection of Traffic Violations

By checking the results of speed and lane determination, traffic violations can be detected, and a warning will be given.

2- Solution Method and Implementation

1. Find Video and Converting Grayscale

In the beginning, a proper video is required to be found. After some research on the internet, a video about highway traffic was found on YouTube. The video name is "4K Highway traffic seen through drone (Copyright Free)" and it was recorded by the drone. The link for the video is:

The link: https://www.youtube.com/watch?v=zaRJVA8959A

Important Note: Since the video take too much space, Sucourse did not let to submit it. The grayscale converted version was uploaded into google drive. Download the video an put it in the same folder with MATLAB codes to run the code.

The drive link: https://drive.google.com/file/d/1gFpfQof3Nz3yHJ4P8OSriV-1suKw9IVt/view?usp=sharing

An image consists of pixels. The image itself can be represented in RGB (Red Green Blue) or grayscale. The RGB images are stored as a matrix with 3 channels whereas grayscale images are stored as a one-dimensional matrix. In image processing and computer vision, the images proceed in grayscale because grayscale images take lower space. For one pixel of RGB images, 8 bits for each color at a total of 24 bits area is required while in grayscale 8 bits is enough. So, the processing is easier and faster with grayscale. This is why grayscale versions are used (Klette, R., 2014).

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

The videos consist of frames which are the images. So the video means a sort of different sequential images. For image processing video taken from the internet is converted into grayscale. This operation was done by the “ConvertVideoRGB2Gray.m” file. It takes the file name of the original video and by using VideoReader() built-in function, the video is uploaded. Then, to write grayscale video the VideoWriter() built-in function is used. Before starting the video writing, it is required to use open() and after all operations, it is required to use close() built-in functions for successful writing operations. By using a while loop, video frames are converted into grayscale by checking their channel sizes. If they are RGB images, they have been converted into grayscale by the rgb2gray() built-in function. The function itself is in the appendix part under the title “ConvertVideoRGB2Gray.m”.

Figure 1: Frame 1 of the frame with RGB and Grayscale

In the main code, the grayscale video is read via the built-in function “VideoReader()”. It creates a VideoReader object consisting of the following properties:

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

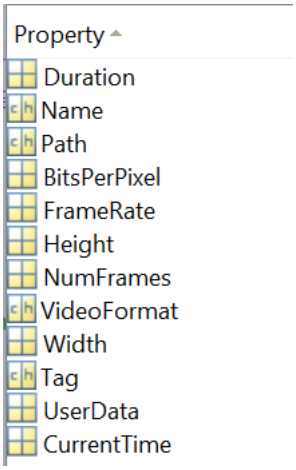

Figure 2: VideoReader Object Properties

When the time comes to speed calculation, the between 2 frames become important. It is calculated by:

$$Time\ duration\ between\ 2\ frames = frameDuration = \frac{Total\ Video\ Duration}{Total\ Frame\ Number}$$

2. Traffic Line Extraction via Hough Lines

Hough transform is a method for feature extraction from images such as extraction of lines or circles. For line detection, the lines itself converted into points in Hough Transform. But, since we have finite pixels in the image and the lines include an infinite number of points on themselves, the line parameters are kept as a distance “rho” and angle “theta”. Where the rho is the perpendicular distance of the line to the origin and the theta is the angle between a perpendicular line and the x-axis. As a result, by Hough Transform, the line becomes a point and the point becomes the wave. And the detection of lines is determined by several waves that intercept at one point. The conversion happens as following figure (Unel, 2022).

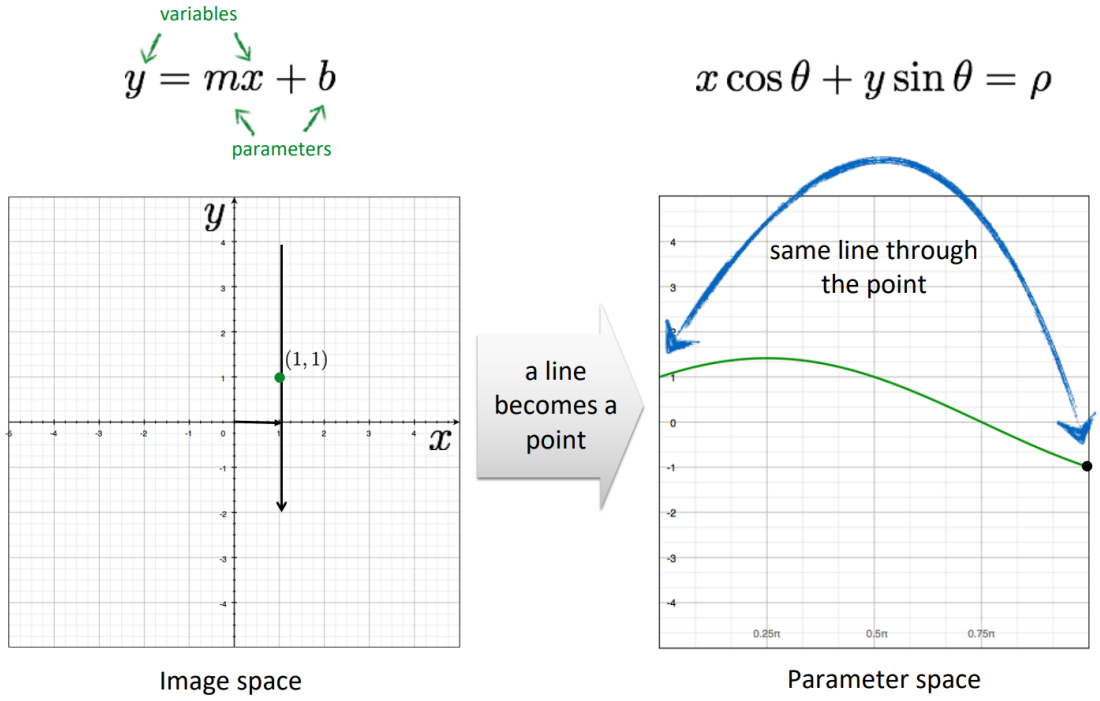

Figure 3: Hough Transfer of Line in Image Space

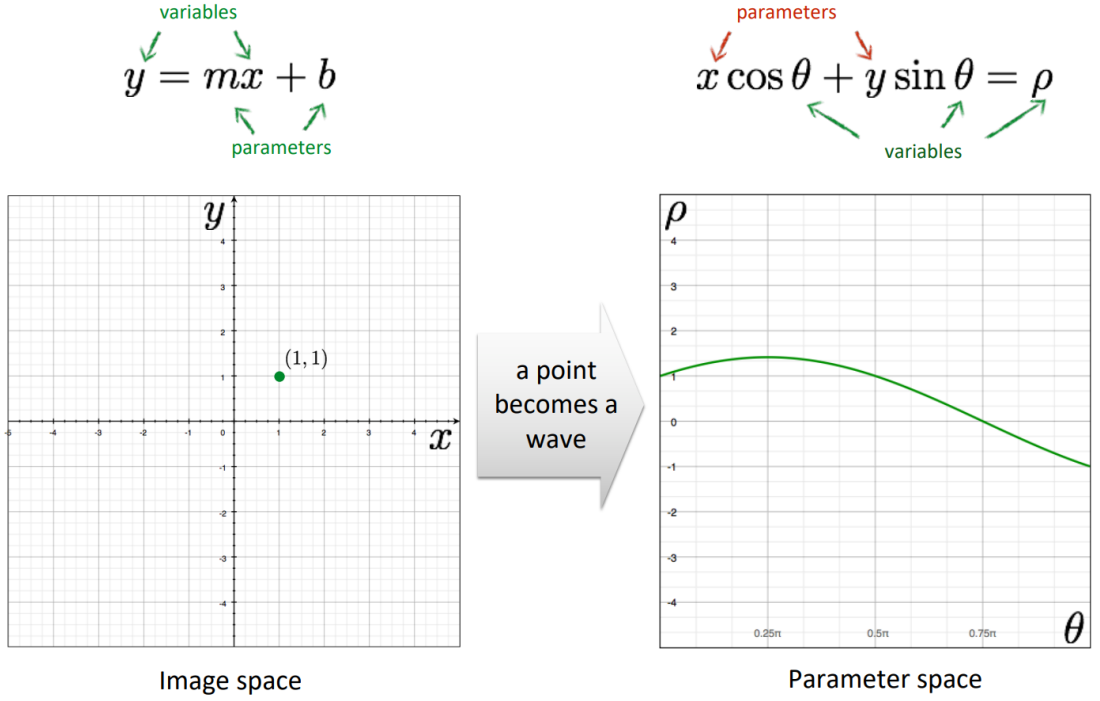

Figure 4: Hough Transfer of Point in Image Space

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

The function is written to take input as an image I, edge detector type, threshold ratio, and max peak numbers then it detects edges and plots the Hough Transform of it. Additionally, it detects max peak points in Hough Transform and detects the lines on the image, and shows them. The lines are drawn as green, and the start and end points of the lines are pointed as a yellow and red cross.

At first, the size of the image is controlled and if the image is in RGB mode, then it is converted into grayscale. It is understood by checking the channel number of the array. RGB images have 3 channels whereas grayscale images have one. The data type of the image is in uint8 in MATLAB. To perform matrix operations, it is required to convert them into double array format.

Then by edge built-in function, the edge detection is done. There are 2 inputs of that function, one is the image itself the other one is the edge detection method. After that, by the hough function, the Hough transform is done and the hough diagram is formed by using the edge detected image by the edge() built-in function. The hough transformation is also done by the hough() built-in function. To get the max-specific peak points of the hough function, the houghpeaks() built-in function is used. It is also a built-in function, before using it, the threshold value is determined. So, the houghpeaks() function gets the peak values above the threshold with the entered number time. Then the image itself converted back to the uint8 type. The function itself is in the appendix part under the title “HoughLineDetection.m”.

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

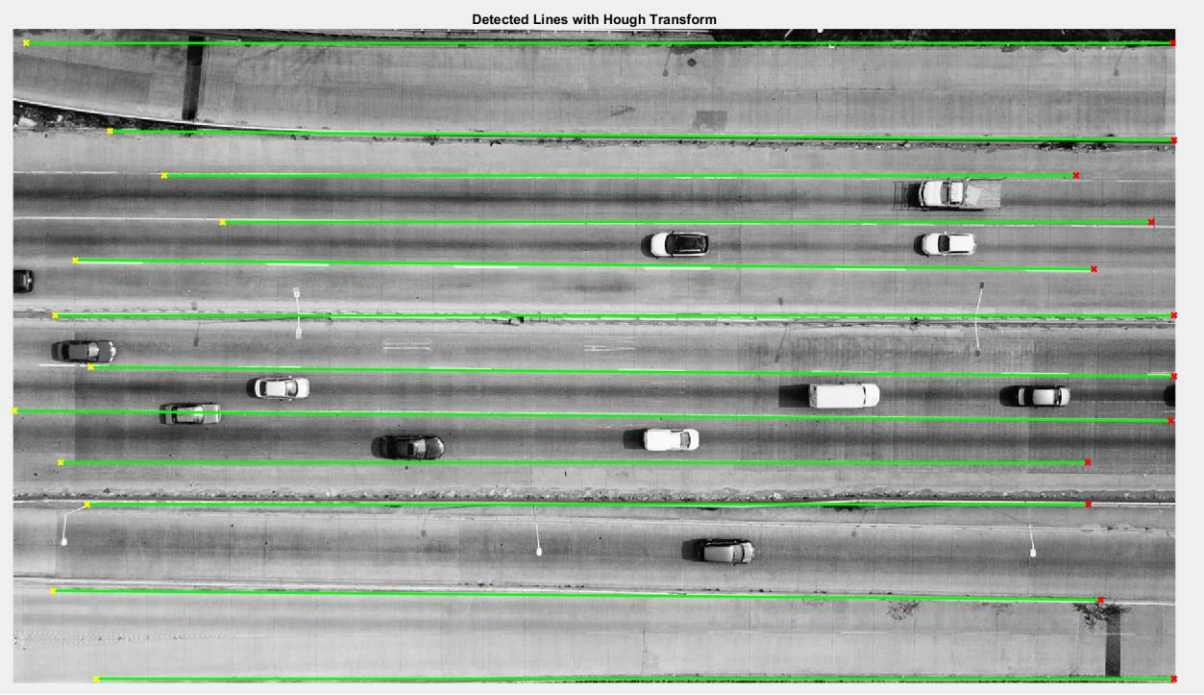

Figure 5: Detected Hough Lines Sample

The function is used for the detection of the correct traffic line and its orientations. The function itself finds all the hough lines in the given image. The hough lines corresponding to traffic lines will be selected manually in the DrawLines() function. This function gets the detected hough lines as an input, then it draws the traffic lines, that will be examined, and pass through from leftmost the rightmost on detected hough line direction.

In the beginning, the necessary hough lines are selected manually. Then to draw lines the start and end points of the lines were calculated. In the beginning, the x points are initialized as go from leftmost to the rightmost. Then y values are calculated by the following formula:

$$y = \frac{\rho - x \times \cos(\theta)}{\sin(\theta)}$$

Then by taking the first and last points of the lines, the desired lines are drawn with magenta color. The function itself is in the appendix part under the title “DrawLines.m”.

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

To get these lines, the parameters were optimized. To get traffic lines correctly the best frame was selected that there is no car on the traffic line. This frame-read operation is done via a built-in function called read(). It takes the videoReader object and gets which frame you want. The first frame was proper for this purpose so it is used. The final version of the optimized parameters and gotten frame assigns were as follows:

```
% Hough Line Detection Parameters
edgeDetector = "LoG";     % edge detector type
thresholdRatio = 0.2;       % threshold
peakNum = 12;               % peak number of the hough lines
fillGap = 1000;             % fill gap amount between 2 lines
minLength = 10;             % minimum lenght to detect line

vidFrame = read(video,1);   % get the specific video frame as an image
```

Figure 6: Optimized Final Parameters For Hough Line Detection

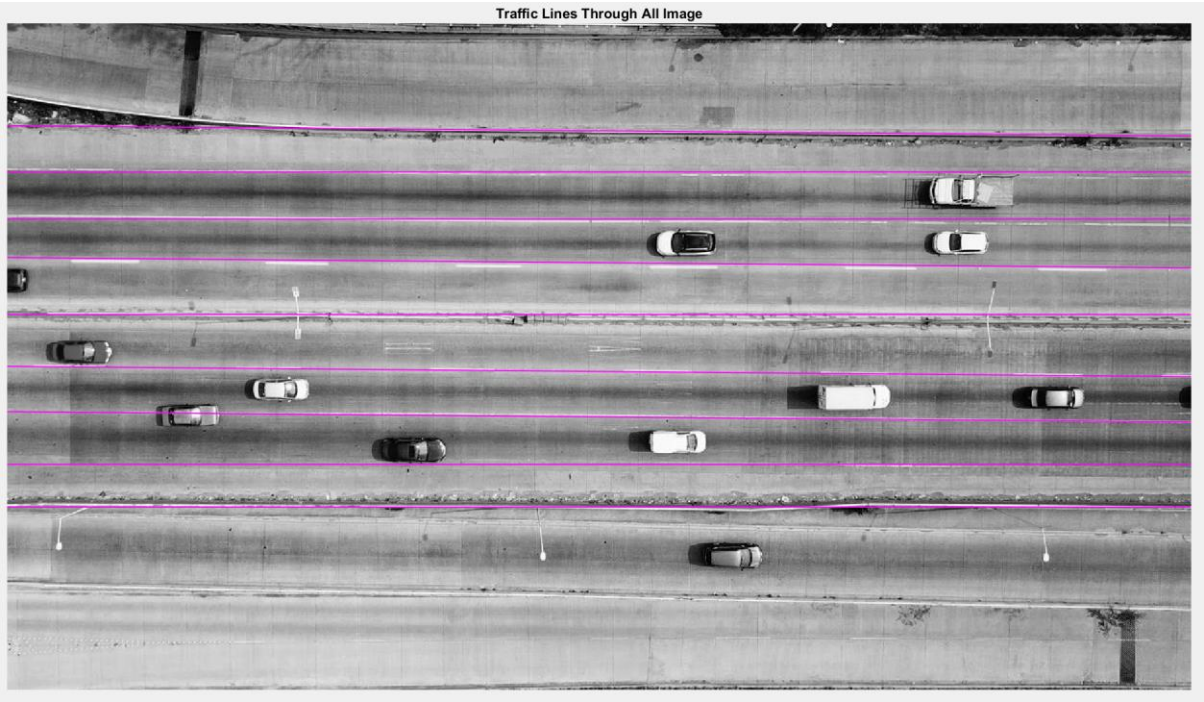

Figure 7: Drawn Traffic Lines with Detected Hough Lines

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

3. Foreground Detection

In the video to detect a moving object, the foreground needs to be detected and separated from the background. For this purpose, MATLAB has a special tool called "ForegroundDetector". By training that tool with different frames, it detects the background and foreground of the road. Then it applies a mask for segmentation. As a result, by using background subtraction it detects foreground objects and allows the user to see foreground objects from a stationary camera orientation.

The tool uses the Gaussian Mixture model for detection. It is a parametric probability density function represented as a weighted sum of Gaussian component densities. A Gaussian mixture model formula:

$$p(\mathbf{x}|\lambda) = \sum_{i=1}^{M} w_i \; g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i),$$

Where:

x = D-dimensional continuous-valued data vector

M = Component number

w = mixture weights

function g = component Gaussian densities

Formula of the function g:

$$g(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)' \; \Sigma_i^{-1} \; (\mathbf{x}-\boldsymbol{\mu}_i)\right\}$$

Where:

μ = mean vector

Σ = covariance matrix

There is a one constraint. Mixture weights should satisfy following formula:

$$\sum_{i=1}^{M} w_i = 1$$

For the ForegroundDetector, there are 2 input parameters were given, these were

- NumGaussians: Number of Gaussian modes in the mixture model
- NumTrainingFrames: Number of initial video frames for training background model

These two values were optimized, and the final given parameter values were 5 for NumGaussians and 726(total number of frames in video) for NumTrainingFrames ("Foreground detection using gaussian mixture models - MATLAB,").

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

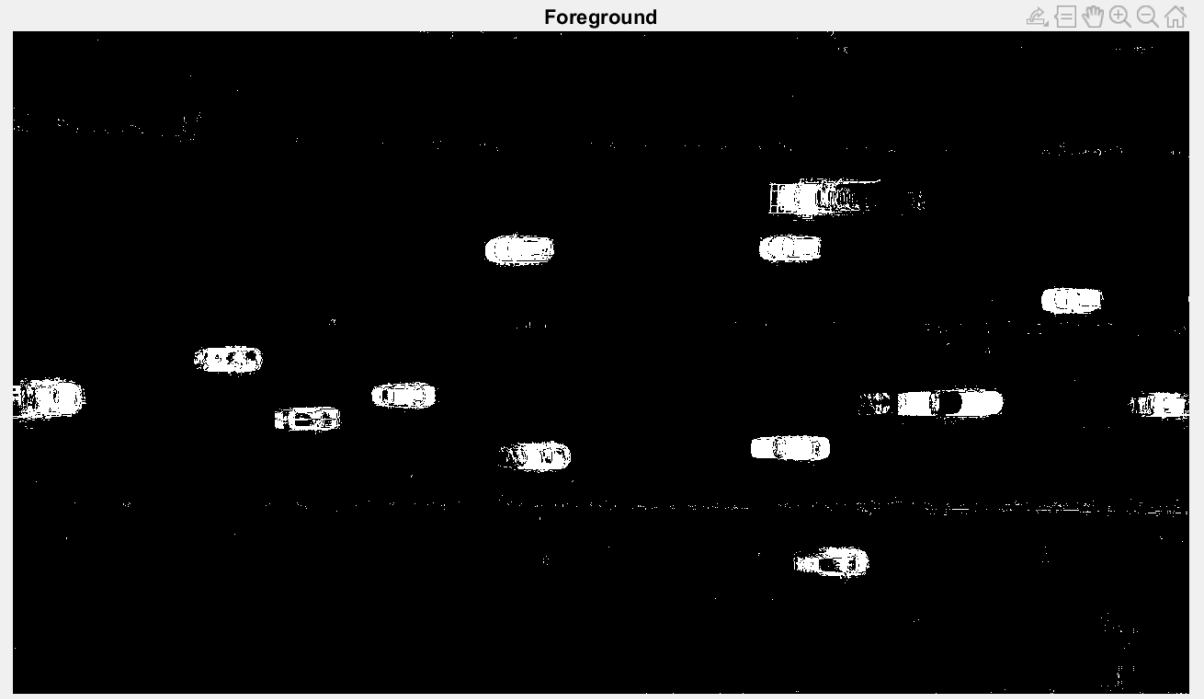

Figure 8: Foreground Segmented Frame

4. Noise Elimination for Foreground Detection

Foreground detection is too important for detection of the moving objects. When only foregroundDetector is used, the results were not good. That is because in the video there are lots of noises and external disturbances as the shadow. To detect moving heavy vehicles clearly, it is required to apply a good noise-elimination filter.

There are several filters for noise cancellation such as Box and Gaussian. In general, the Gaussian is used more than the box filter since the weight of the elements is placed in the middle of the kernel, there is only one lobe in Fourier transforms. And this causes less loss of f information while blurring.

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

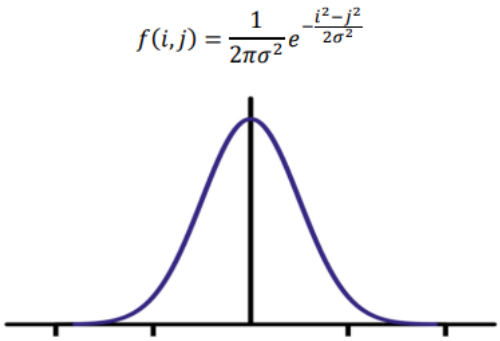

Figure 9: Gaussian Function and Filter Formula

Therefore, in the beginning, the Gaussian filter was applied to the frames. The gaussian filter decreases the noise until some point. However, it was not enough. Other solutions were tried. One of the best solutions is applying morphological structuring elements.

A strel object represents a flat morphological structuring element, which is an essential part of morphological dilation and erosion operations.

A flat structuring element is a binary-valued neighborhood, either 2-D or multidimensional, in which the true pixels are included in the morphological computation, and the false pixels are not. The center pixel of the structuring element, called the origin, identifies the pixel in the image being processed. Use the strel() function (described below) to create a flat structuring element. You can use flat structuring elements with both binary and grayscale images. The following figure illustrates a flat structuring element.

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

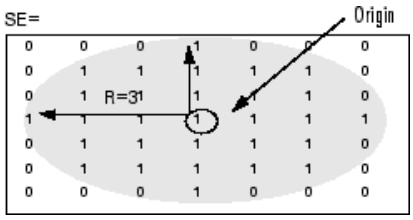

Figure 10: Structuring Element Explanation

With this strel() method, the noise cancellation process is done better. The strel object parameters are optimized as creating a square structuring element whose width is 1 pixel ("Detect cars using gaussian mixture models - MATLAB").

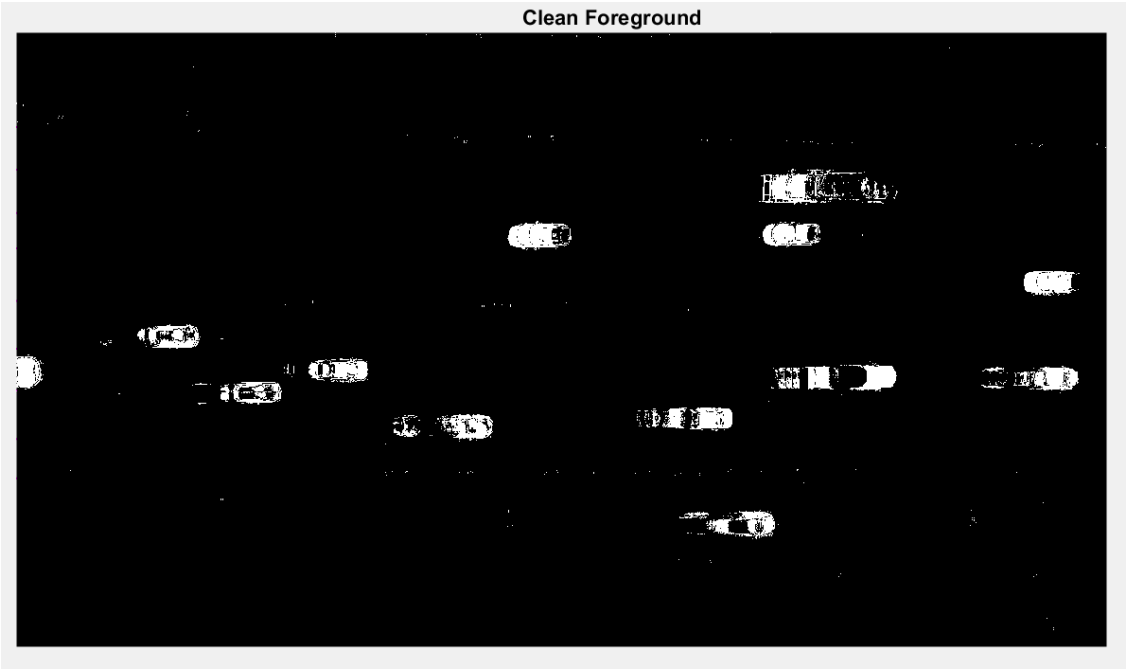

Figure 11: Foreground Segmented Frame after Structural Noise Cancellation Filtering

5. Blob Analysis

To compute statistics of connected regions, the blob analysis is done. The blobAnalysis object is created in the code as “vision.BlobAnalysis()”. It takes some parameters. These are:

- BoundingBoxOutputPort: Return coordinates of bounding boxes = true
- AreaOutputPort: Return blob area = false
- CentroidOutputPort: Return coordinates of blob centroids = false
- MinimumBlobArea: Minimum blob area in pixels = 3500

The parameter MinimumBlobArea has optimized the detection of heavy vehicles correctly and not detecting small vehicles. The final value of the parameters was given above. By the line bbox = step(blobAnalysis, filteredForeground), the detected heavy vehicles' bounding box parameters were gotten in the given foreground detected frame. A function "FindCenterLocationOfDetectedCars" was written to keep detected heavy vehicle positions in compact form. It takes the boundary box from blob analyses and the detected traffic lines array as input and returns center locations and the line number of detected cars in one matrix. The function itself is in the appendix part under the title "FindCenterLocationOfDetectedCars.m".

In the while loop, for each time the currentPosition matrix is updated by "PositionUpdate" function. It just does matrix manipulation and it is done to take differences in the positions between frames to calculate speeds. The function itself is in the appendix part under the title "PositionUpdate.m".

6. Speed Calculation

For the speed calculation basic formula is as following:

$$V = \frac{\Delta x}{\Delta t}$$

$Where$:

$V = average\ velocity$

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

$$\Delta x = displacement$$

$$\Delta t = change\ in\ time$$

As seen from the average velocity formula given above, displacement and change in time are required to calculate. The time is calculated as frame duration at the beginning. It is the time change between 2 consecutive frames. To calculate the displacement, at first, the pixel number was founded between one dashed traffic line from one random frame as follows:

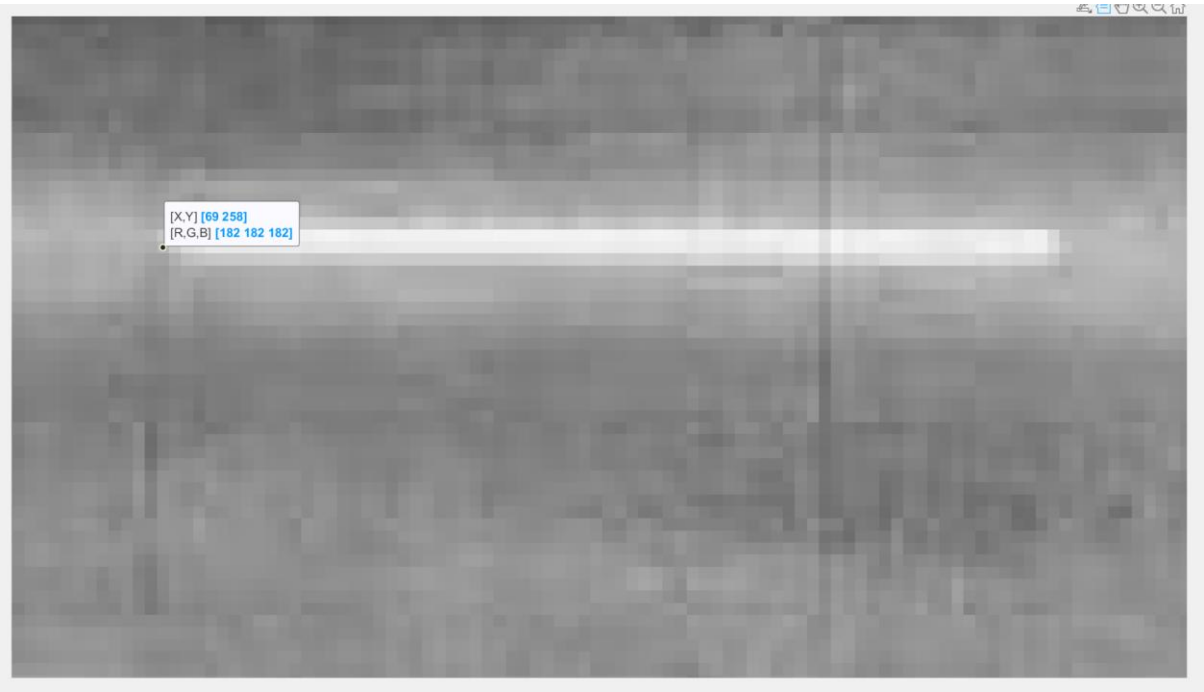

Figure 12:Dashed Traffic Line Start Pixel Location

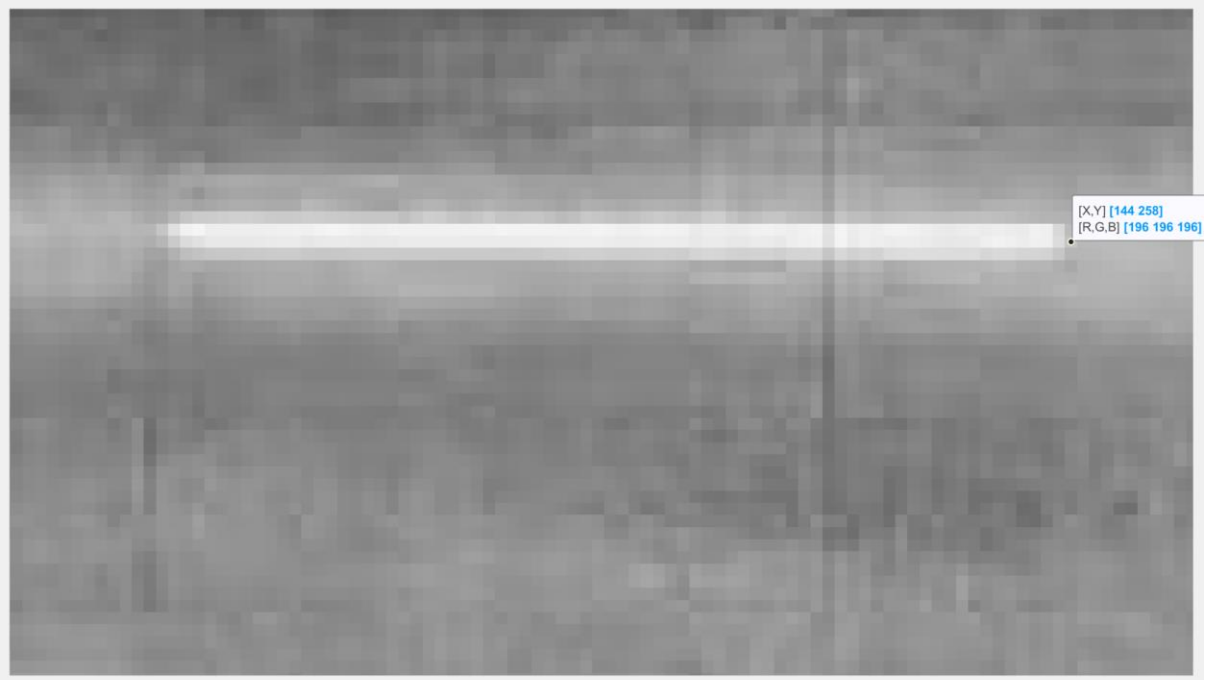

Figure 13: Dashed Traffic Line End Pixel Location

One dashed traffic line's length in the x direction is around 75 pixels (144-69). In real life, the length of the dashed traffic line is around 450 cm ("Trafik Kuralları," n.d.). Therefore, the distance between 2 pixels in real life was around 4.5 over 75 meters. The traveled distance is completed by taking the difference between the previous position and the current position, then multiplying it with the conversion parameter 4.5/7. After that when the displacement is divided by frameDuration, you get the average speed of the vehicle. The unit of the velocities is converted into km/h to make more sense in traffic. These operations were done with a function called SpeedCalculator.m. The function itself is in appendix part under title “SpeedCalculator.m” (Pornpanomchai & Kongkittisan, 2009).

7. Violation Detection

After all the previous processes, the time comes for the detection of the vehicles which violate the rules. Two rules were determined basically:

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

Rule 1: Each line has a specific speed limit for heavy vehicles. If any detected heavy vehicles exceed the limit, the speed of the heavy vehicle will be written in a red box. And the total number of the vehicle causing this violation will be printed on the left upper of the figure.

Rule 2: There are specific lines that are forbidden for heavy vehicles, If any heavy vehicle is detected on one of that traffic lines, instead of the speed of the vehicle, a “Wrong Lane” message will be written on the detected vehicle. And the total number of the heavy vehicle causing this violation will be printed on the left upper of the figure.

If there is no violation, the speed of the heavy vehicles will be written in the green box and for all cases, the total number of the detected vehicles will be written on the left upper of the figure.

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

RESULTS

1- Speed Exceeding Detection for Heavy Vehicles

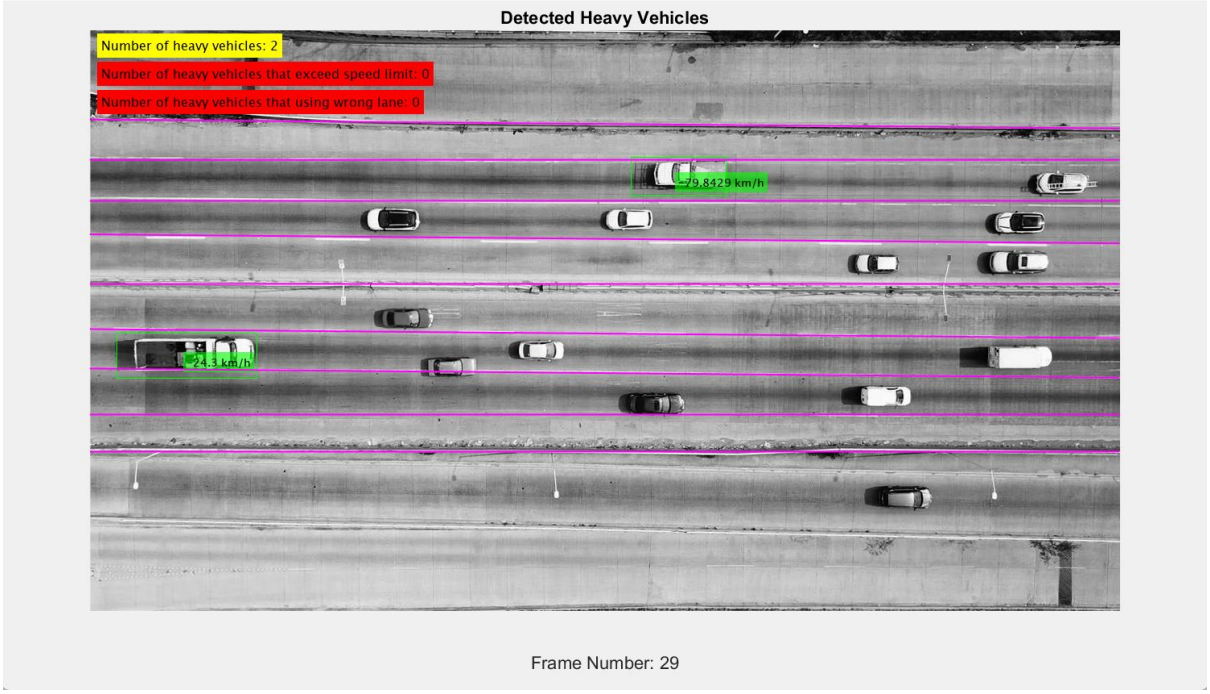

Figure 14. Two heavy vehicles were detected and there is no traffic violation

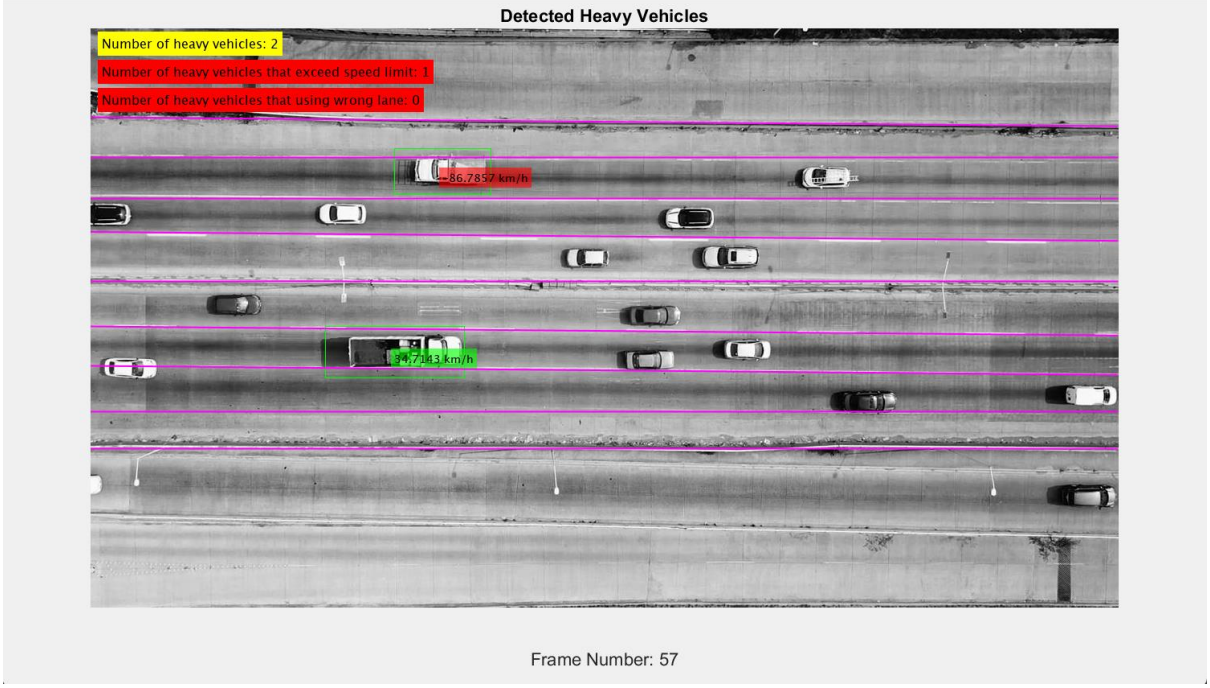

Figure 15. Two vehicles were detected and one of them exceeded the speed limit

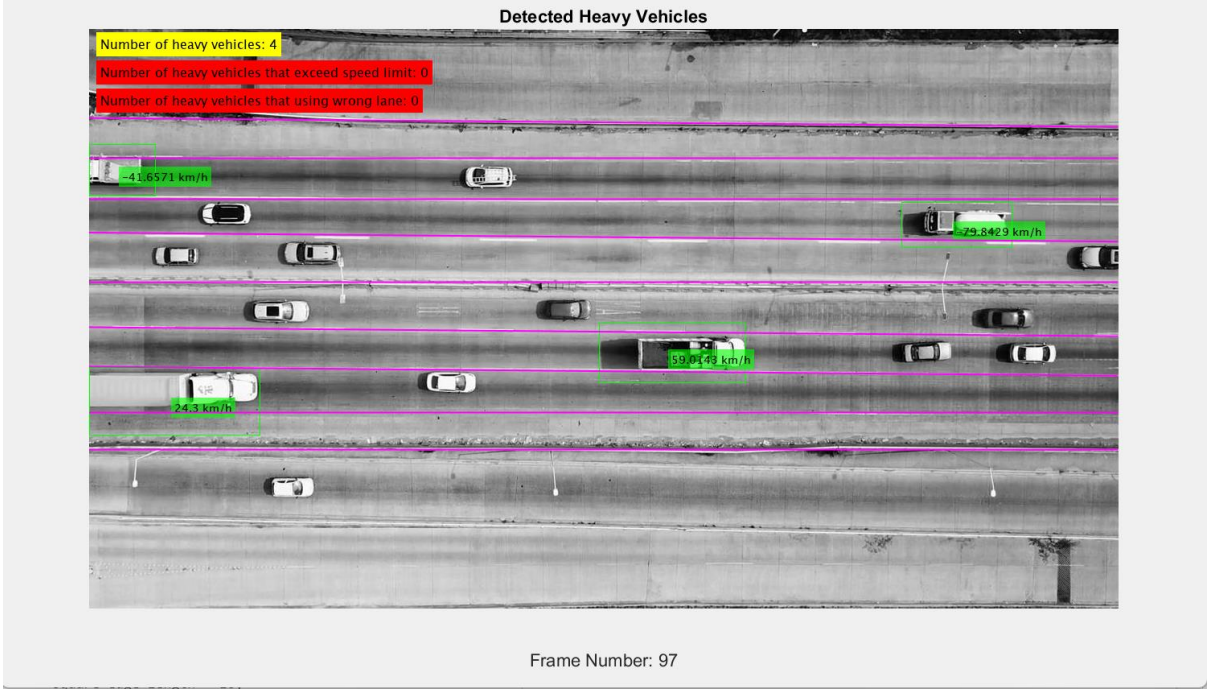

Figure 16. Four heavy vehicles were detected and there is no traffic violation

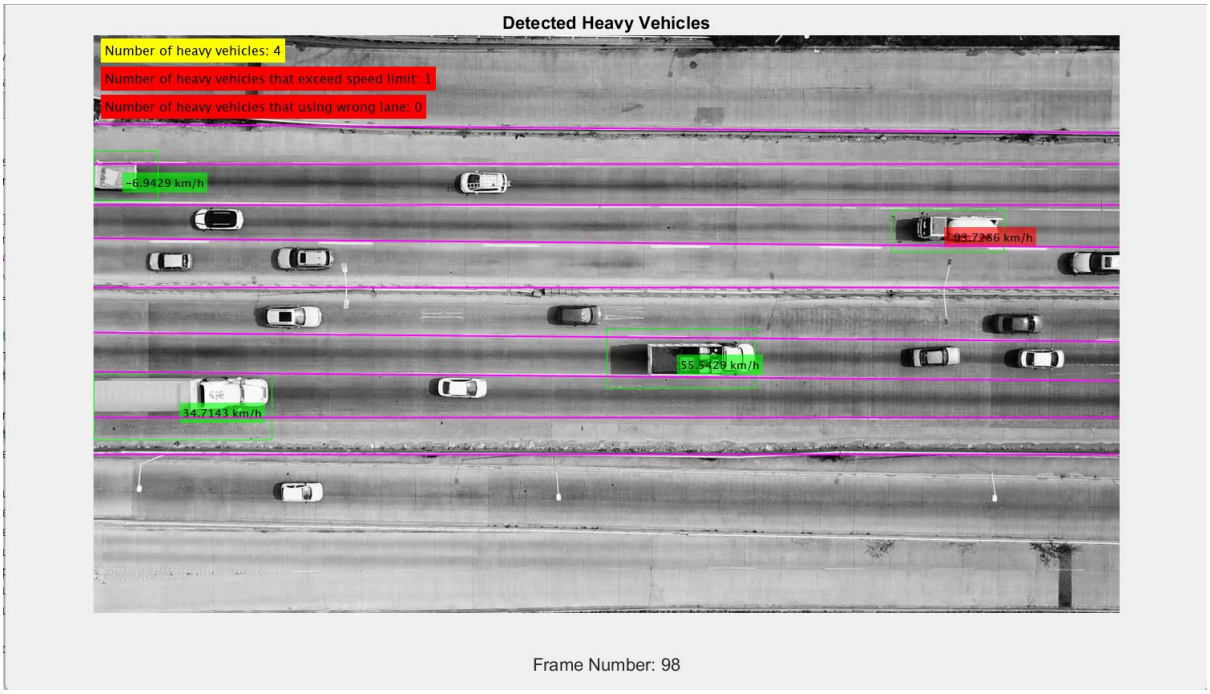

Figure 17. Four heavy vehicles were detected and one of them exceeded the speed limit

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

2- Using Wrong Lane Detection for Heavy Vehicles

Important Note: Heavy vehicles should only use the nearside lane and they can only use the middle lane to overtake another vehicle, but they cannot use the left lane. In our video, we don't have this kind of traffic rule violation. Because of this reason, we applied this rule for middle lane to show that algorithm works and can be applied.

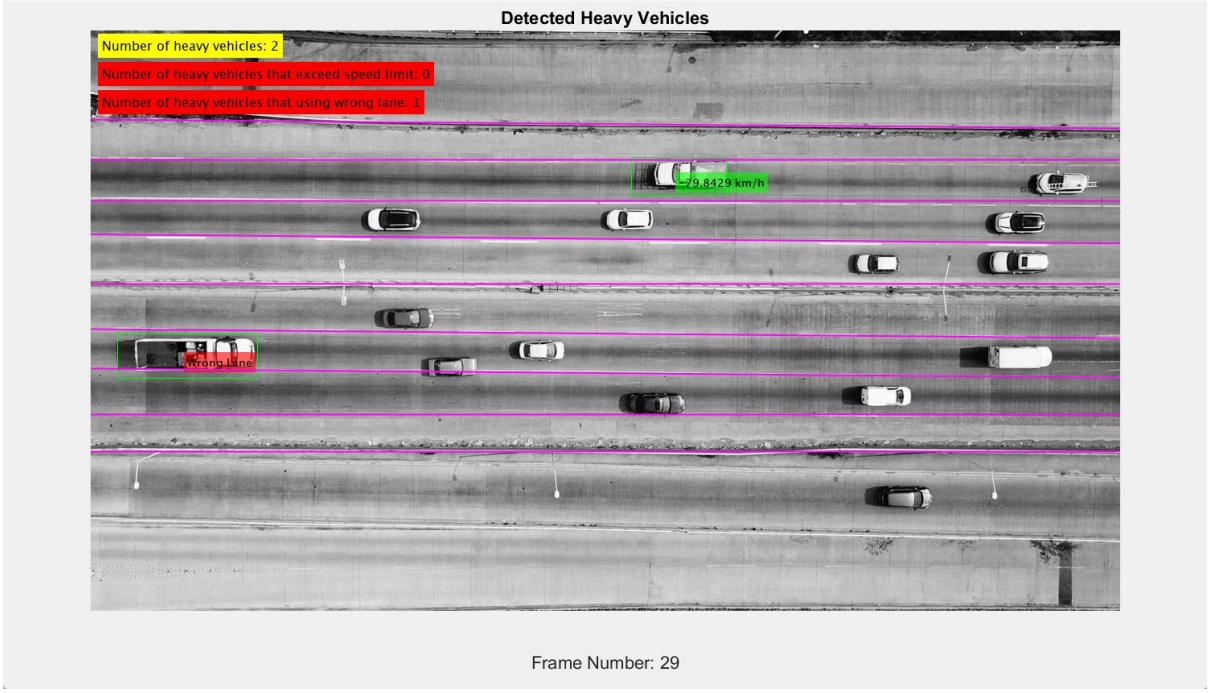

Figure 18. Two heavy vehicles were detected and one of them using the wrong lane

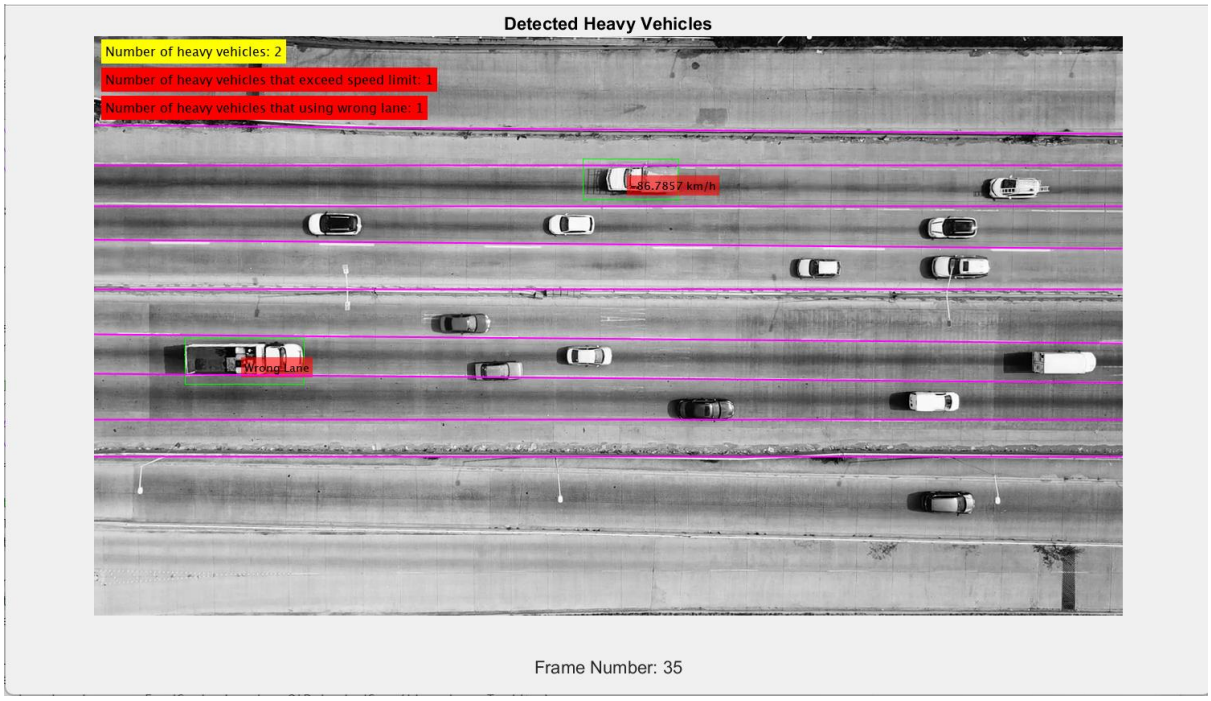

Figure 19. Two heavy vehicles were detected and one of them exceeded the speed limit while the other one used the wrong lane

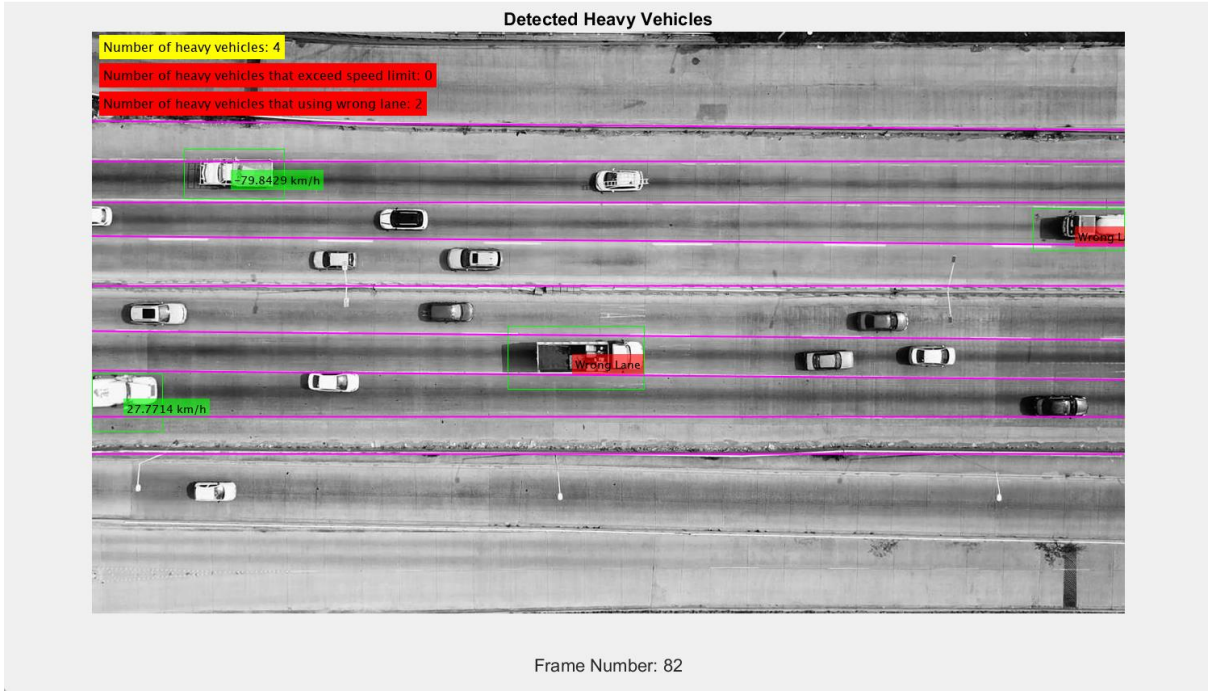

Figure 20. Four heavy vehicles were detected and two of them using the wrong lane

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

DISCUSSION

In conclusion, the results obtained by these algorithms are sufficient. Goals are achieved. The steps worked correctly. Everything started with detecting houghlines in the video frames. According to the results of houghlines, we divided the road into lanes. The next step was finding moving objects in the video frame. We used Foreground Detector for this purpose. The camera was stable and not shaking. Therefore, we got enough sufficient results from Foreground Detector. With these algorithms, we achieved to detection of heavy vehicles and tracking them. After that, it was time to calculate the speed of heavy vehicles. By using time and the position change of the heavy vehicles, we calculated their speeds. After that, we made some small research about the rules for heavy vehicles. According to this research, we learned that heavy vehicles should not exceed 85km/h. Our video has 3 separate lanes for each way. So, we decided to give a speed limit to the nearside lane of 85 km/h and the middle lane to 90 km/h because we considered heavy vehicles can overtake another vehicle, and to do this their speed should be more. We determined the speed limits for each lane on the road depending on these rules for heavy vehicles. Also, we determined the lanes that heavy vehicles can use. From research, we learned that heavy vehicles should only use the nearside lane and they can only use the middle lane to overtake another vehicle, but they cannot use the left lane. We printed the warnings on the video screen according to these rules. In the end, we completed the project as it should be.

Figure 14 represents a successful heavy detection in case there is no violation. The number of the detected vehicles was printed at the left upper of the figure. Since there was no speed violation and wrong lane violation, the number of detected vehicles in violation was zero. As shown in the figure, the direction of the velocity of the vehicles was also detected.

The upper 4 lanes of traffic flow through the left, and the below 4 lines flow through the right. If the velocity of the detected vehicle is minus that means the vehicle moves to the left.

Figure 15 represents the speed violation. The speed limit for line 2 was 85 km/h. Since the heavy vehicle exceeds the speed limit, its velocity is written in a red box and the number of heavy vehicles that exceed the speed limit becomes 1. The speed limit of line 6 was entered as 90 and the detected vehicle moved 34.7 km/h. This is why its velocity is written in a green box.

Figure 16 represents the detection of 4 heavy vehicles with no violation. Since the number of detected vehicles are increased, the number of detected heavy vehicle number at the upper left is updated from 2 to 4.

Figure 17 shows another speed violation of heavy vehicles. In this case, the number of detected vehicles was 4.

The code itself also detects the cars that move in the wrong lane. In traffic rules, heavy vehicles should not use the leftmost traffic line. However, this case is not included in the found video. To show the case, lane 6 was selected as forbidden for heavy vehicles, and the code was run again. As shown in figure 18, the heavy vehicle in line 6 has a “Wrong Lane” error itself and the number of vehicles in the wrong way became 1 at the left upper of the figure.

Figure 19 shows both speed and wrong lane violation case detection in one figure. As expected, the numbers of both violations became 1 and in terms of the violation, the error messages were given in the figure.

All these results show that the code and implementation work successfully. The detections can work parallelly.

REFERENCES

1. *Buch, N., Velastin, S. A., & Orwell, J. (2011). A review of computer vision techniques for the analysis of urban traffic. IEEE Transactions on Intelligent Transportation Systems, 12(3), 920-939. https://doi.org/10.1109/tits.2011.2119372*
2. *Detect cars using gaussian mixture models*. (n.d.). MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink. https://www.mathworks.com/help/vision/ug/detecting-cars-using-gaussian-mixture-models.html
3. *Foreground detection using gaussian mixture models - MATLAB*. (n.d.). MathWorks - Makers of MATLAB and Simulink - MATLAB & Simulink. https://www.mathworks.com/help/vision/ref/vision.foregrounddetector-system-object.html
4. Klette, R. (2014). *Concise computer vision* (Vol. 233). London: Springer.
5. Pornpanomchai, C., & Kongkittisan, K. (2009). Vehicle speed detection system. *2009 IEEE International Conference on Signal and Image Processing Applications*. https://doi.org/10.1109/icsipa.2009.5478629
6. *Trafik Kuralları*. (n.d.). Baskent Surucu Kursu. https://www.baskentsurucukursu.com.tr/ders-trafik-kurallar-368.html
7. *Unel M.* (2022). EE 417 Computer Vision Course [pdf- Slides]. Sabanci University
8. (n.d.). YouTube. https://www.youtube.com/watch?v=zaRJVA8959A

Osman Melih Kurt - 26497
Okan Arif Güvenkaya - 26780

APPENDIX

Term_Project.m

```
% EE 417 Computer Vision Term Project
% Osman Melih Kurt  26497
% Okan Arif Guvenkaya   26780

clear all; close all; clc; % Clear all the previous variables

% Uploading the video
fileNameGrayScale = "4K_Highway_Traffic_Seen_Through_Drone_GrayScale.avi"; % define file name
video = VideoReader(fileNameGrayScale);    % upload the video

% Get information about the video
duration = video.Duration;          % get duration of the video
frameNum = video.NumFrames;         % get frame number
frameDuration = duration/frameNum;  % get the time duration between 2 frames

% Hough Line Detection Parameters
edgeDetector = "LoG";       % edge detector type
thresholdRatio = 0.2;       % threshold
peakNum = 12;               % peak number of the hough lines
fillGap = 1000;             % fill gap amount between 2 lines
minLength = 10;             % minimum lenght to detect line
vidFrame = read(video,1);   % get the specific video frame as an image

lines = HoughLineDetection(vidFrame, edgeDetector, thresholdRatio, peakNum, fillGap, minLength); % Get
hough line of the frame

trainingFrameNumber = frameNum; % number of frames to be trained for background model
gaussianNumber = 5;             % number of Gaussian modes in the mixture model
foregroundDetector = vision.ForegroundDetector('NumGaussians', gaussianNumber,'NumTrainingFrames',
trainingFrameNumber); % foreground detector

currentPositions= zeros(8,2); % keep positions to calculate speed

lineSpeedLimits = [0 85 90 0 0 90 85 0]; % Speed limits of each traffic line in km/h

text1 = "Number of heavy vehicles: ";
text2 = "Number of heavy vehicles that exceed speed limit: ";
text3 = "Number of heavy vehicles that using wrong lane: ";

for i = 1:109 % frameNum

    frame = read(video, i); % read the the new frame in the frame
    foreground = step(foregroundDetector, frame); % detect forefground

    % Get better foreground operation
    se = strel('square', 1); % morphological erosion operations to detect foreground better
    filteredForeground = imopen(foreground, se);

    % Apply Blob analysis to detect heavy vehicles in the traffic
    blobAnalysis = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
        'AreaOutputPort', false, 'CentroidOutputPort', false, ...
        'MinimumBlobArea', 3500); % blob anlysis computes statistics for connected regions
    bbox = step(blobAnalysis, filteredForeground); % detect the areas frome foreground detected frame

    % Plotting Operations
    linesTraffic = DrawLines(lines, frame); % Determine the special traffic lines to be consider
    result = insertShape(frame, 'Rectangle', bbox, 'Color', 'green'); % draw the rectangle to detected
heavy vehicles

    numCars = size(bbox, 1); % count the heavy vehicles
    result = insertText(result, [10 5], text1 + numCars, 'BoxOpacity', 1, 'FontSize', 16); % write the
number of cars on top left corner of the frame
    figure(1); imshow(result); title('Detected Heavy Vehicles'); xlabel("Frame Number: " + i); hold
on;
```

```
    if (numCars >= 1)
        tempPositions = currentPositions;
        center_locations = FindCenterLocationOfDetectedCars(bbox,linesTraffic);
        currentPositions = PositionUpdate(center_locations,currentPositions);
        speedsAndLocations = SpeedCalculator(tempPositions, currentPositions, frameDuration);
        num_speedExceed = 0;
        num_wrongLine = 0;

        for k=1:size(speedsAndLocations(:,1))
            if(speedsAndLocations(k,1) ~= 0)
                square_edge_length = 10;
                left_up_position = [speedsAndLocations(k,1)-square_edge_length/2
speedsAndLocations(k,2)-square_edge_length/2];
                string_speed = speedsAndLocations(k,3) + " km/h";
                speedLimit = lineSpeedLimits(k);

                if k == 4 || k == 5 || k == 1 || k == 8 % wrong line for heavy vehicles
                    result = insertText(result, left_up_position, "Wrong Lane", 'BoxColor','r',
'FontSize', 14);
                    num_wrongLine = num_wrongLine + 1;
                elseif abs(speedsAndLocations(k,3)) > speedLimit % speed limit check
                    result = insertText(result, left_up_position, string_speed, 'BoxColor','r',
'FontSize', 14);
                    num_speedExceed = num_speedExceed + 1;
                else
                    result = insertText(result, left_up_position, string_speed, 'BoxColor','g',
'FontSize', 14);
                end
                imshow(result);
            end
        end
        result = insertText(result, [10 40], text2 + num_speedExceed, 'BoxOpacity', 1, 'FontSize', 16,
'BoxColor', 'red');
        result = insertText(result, [10 75], text3 + num_wrongLine, 'BoxOpacity', 1, 'FontSize', 16,
'BoxColor', 'red');
        imshow(result);
    end
end
```

ConvertVideoRGB2Gray.m

```
%{
This function converts the RGB video into GrayScale.
The output graysacele video in  ".avi" format.
While converting into grayscale, some of the changes can be happen in video
as change in frate etc.
%}

clear all; close all; clc; % Clear all the previous variables

% Uploading the video
fileName = "4K_Highway_Traffic_Seen_Through_Drone.mp4"; % define file name
video = VideoReader(fileName); % upload the video

video_new = VideoWriter(filenName + "_Grayscale"); % define VideoWriter object
open(video_new); % to write the video you need to open it

% Converting Video into Grayscale
while hasFrame(video) % iterate over each frame in original RGB video
    vidFrame = readFrame(video); % get the frame iteratively

    [row, col, ch] = size(vidFrame); % Getting size of an image
    if(ch == 3) % Checking image is rgb or grayscale
        vidFrame = rgb2gray(vidFrame); % Convert image into grayscale if it is not
```

```
    end
    writeVideo(video_new,vidFrame); % write the video frame into video_new
end
close(video_new); % when the writing operation is done, close the VideoWriter object
```

HoughLineDetection.m

```
%{
This function gets single image and houhgline parameters as an input
Detect houghlines in the image
Return the lines struct that contain
        point1 = start point [x,y]
        point2 = end point [x,y]
        theta = theta angle
        rho = rho distance
%}
function lines = HoughLineDetection(I, edgeDetector, thresholdRatio, peakNum, fillGap, minLength)
    % Converting Grayscale
    [row, col, ch] = size(I); % Getting size of an image
    if(ch == 3) % Checking image is rgb or grayscale
     I = rgb2gray(I); % Convert image into grayscale if it is not
    end

    I = double(I); % Required conversion for pixelwise processes
    I_edges = edge(I, edgeDetector); % Canny Edge Detection
    [H, theta, rho] = hough(I_edges, "RhoResolution", 0.5, "Theta", -90:0.5:89);
    threshold = ceil(thresholdRatio*max(H(:))); % threshold
    P = houghpeaks(H, peakNum,"Threshold", threshold); % peak numbers
    I = uint8(I); % Converting the double array to image format
    lines = houghlines(I_edges,theta,rho,P,'FillGap', fillGap,'MinLength', minLength); % Find the
lines on the image
    figure; imshow(I), hold on;
end
```

DrawLines.m

```
%{
This function gets hough lines, and eliminate will be not used ones
Draw lines goes throuhg hough line orientation from leftmos to rightmost of
the input image
Return the linesTraffic struct that contain
        point1 = start point [x,y]
        point2 = end point [x,y]
        theta = theta angle
        rho = rho distance
%}
function linesTraffic = DrawLines(linesTraffic, I)
    % Since we look for main 8 lines of the highway, the others need to be neglected
    % neglected lines index are
    neglected_lines_index = [1 8 3];

    linesTraffic = linesTraffic([2,4,5,6,7,9,10,11,12]);

    x_p_start = zeros(length(linesTraffic));
    x_p_end = zeros(length(linesTraffic));
    y_p_start = zeros(length(linesTraffic));
    y_p_end = zeros(length(linesTraffic));

    for line_index = 1:1:length(linesTraffic)
        % Get rho and theta of the hough lines
```

```
        rho = linesTraffic(line_index).rho; % get rho
        theta = linesTraffic(line_index).theta; % get theta

        % Find the horizontal line goes through from leftmost to rightmost
        x_p = 0:size(I,2);
        y_p = (rho - x_p*cosd(theta))/sind(theta);

        % Ploting Lines Through Image
        x_p_start = x_p(1);
        x_p_end = x_p(end);
        y_p_start = y_p(1);
        y_p_end = y_p(end);

        plot([x_p_start, x_p_end], [y_p_start, y_p_end] ,'LineWidth',1,'Color','magenta'); hold on;
    end
end
```

FindCenterLocationOfDetectedCars.m

```
%{
This function gets selected hough lines for traffic lines
and bounding box of the detected heavy vehicles.
Find the center locations of the boundiing box
and find the traffic line that detected heavy vehicle in
%}
function center_locations = FindCenterLocationOfDetectedCars(bbox, linesTraffic)
    boundaryBox = double(bbox);
    line_y_axis = [];
    line = [];

    for i = 1:length(linesTraffic)
        trafficLine = abs(linesTraffic(i).point1(2) + linesTraffic(i).point2(2))/2;
        line_y_axis = [line_y_axis trafficLine];
    end

    line_y_axis = sort(line_y_axis);
    center_locations_x = [];
    center_locations_y = [];
    line_nums = [];

    for i = 1:size(boundaryBox(:,1))
        % Find center coordinates of the rectangles
        center_x = boundaryBox(i,1) + boundaryBox(i,3)/2;
        center_y = boundaryBox(i,2) + boundaryBox(i,4)/2;
        line_num = 0;

        % Find the traffic line that center location in
        for k = 1:length(line_y_axis)-1
            if((center_y > line_y_axis(k)) && ((center_y < line_y_axis(k+1))))
                line_num = k;
                break;
            end
        end

        center_locations_x = [center_locations_x center_x];
        center_locations_y = [center_locations_y center_y];
        line_nums = [line_nums line_num];
    end

    center_locations = [center_locations_x' center_locations_y' line_nums'];
end
```

PositionUpdate.m

```
%{
This function update the current position of the vehicles for speed
calculation.
%}
function currentPositions = PositionUpdate(center_locations,currentPositions)
currentPositions= zeros(8,2);
    for i = 1:size(center_locations)
        lineNum = center_locations(i,3);
        currentPositions(lineNum,1) = center_locations(i,1);
        currentPositions(lineNum,2) = center_locations(i,2);
    end
end
```

SpeedCalculator.m

```
%{
This function previous and current positions of the vehicles and time
change as an input.
It calculaates the velocities and return current velocity and positions in
matrix form.
%}
function speedsAndLocations = SpeedCalculator(tempPositions, currentPositions, frameDuration)
    % dashed traffic line lenght in real life around 450 cm = 4.5 m
    % one dashed line length is around 75 pixels in x axis in one frame
    % therefore lentght between 2 pixel = 4.50/70 m
    deltaLength = 4.5/70; % distance between 2 pixel in meter [m]
    deltaTime = frameDuration; % time between two frames
    traveledDistance = deltaLength*(currentPositions(:,1) - tempPositions(:,1));
    speeds = traveledDistance/deltaTime;
    speeds = 3.6*speeds; % m/s to km/s conversion
    speedsAndLocations = [currentPositions speeds];
end
```