Lane Keeping Assist System

Ciara Power
BSc in Applied Computing
Waterford Institute of Technology
Waterford, Ireland
ciarapower88@gmail.com

***Abstract*— this document outlines the design and implementation of a Lane Keeping Assist (LKA) system, modelled in Simulink. This is an important Advanced Driver Assistance System (ADAS) for an automated vehicle in an Intelligent Transport System. Vision based lane recognition systems are used extensively throughout the automotive industry to provide this ADAS system.**

Keywords—detection, steering, Simulink, Matlab, Hough

I. INTRODUCTION

Lane detection systems with keep assist are a huge part of the automotive industry at present, using many approaches and algorithms to provide an effective and accurate ADAS system. This project investigates using image processing and Hough algorithms to accurately detect lanes in a video input. The lane detected is compared with the ego vehicle position to assess if the vehicle direction should be modified to stay within the center of the lane. The main goal of this project is to approximate a steering angle controller for the vehicle based on lane detection on a straight road.

II. IMAGE PREPROCESSING

A. *Video Input*

The sensor input chosen to for this system development, is an MP4 colour video input, found online [1], which shows real dashboard camera footage of a vehicle driving along a road, within a marked lane. This input is provided to my system via a Simulink multimedia block. I felt it was important to use a video input rather than a still image, as the algorithms and image processing techniques used must be suitable for varying real camera data in a moving vehicle. An example frame from the video used is shown in Fig. 1 below.

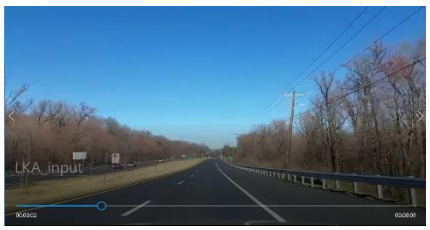

Figure 1 Video Input Image

B. *Region of Interest*

For this system, only the region of interest (ROI) requires to be processed to detect lanes. I cropped the input frames to a region that includes the road ahead of the ego vehicle, seen in Fig. 3, which would be similar for all camera data taken at this angle for any vehicle. This cropped image frame is then used for all further image processing and lane detection.

Figure 2 Full Frame

Figure 3 Cropped ROI

C. *Image Smoothing*

The image must be filtered to reduce noise and improve the performance of the edge detection algorithms [2]. The first step involves converting the image to grayscale, be using the Matlab rgb2gray function. Next, to eliminate the edge detector highlighting edges along different shades of gray on the same surface (road texture may be uneven), the image is binarized using imbinarize. In order to highlight the areas that are white and darken everything else, the threshold parameter is set to 0.85. Morphological functions are then used on the image, to remove any pixels that are not of interest [2]. The bwmorph function is used twice, first with parameter “clean”, which removes any isolated pixels in the image [3]. The “skel” parameter is also used with the bwmorph function, which removes pixels on object boundaries, leaving a singular skeleton line per object [3]. This is useful to ensure the edge detector is not picking up both edges of a singular lane marking.

Figure 4 Original Image

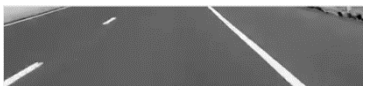

Figure 5 After rgb2gray

Figure 6 After imbinarize

Figure 7 After bwmorph clean

Figure 8 After bwmorph skel

III. Edge Detection

This step extracts all possible points from the image that are likely to become an object edge point. These points are then analyzed to remove the irrelevant points that do not comprise a valid edge [2]. There are four methods mainly used for this by previous researchers when using lane detection, Canny, Roberts, Prewitt and Sobel. A study done shows that Roberts is the most suitable in terms of speed and simplicity [2], and this choice is further backed up by tests ran on my data, showing a clear singular edge detected by Roberts.

Figure 9 Before Edge Detect

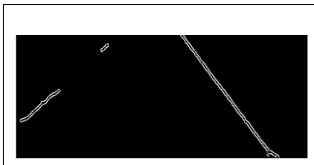

Figure 10 Canny

Figure 11 Roberts

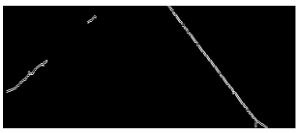

Figure 12 Prewitt

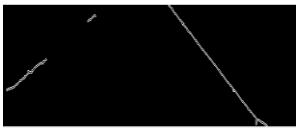

Figure 13 Sobel

Roberts edge detection method is based on applying horizontal and vertical filters consecutively. The results of these filters are then summed to give the final result [4]. This method requires an image with reduced noise to be accurate [4], which is attained from the image preprocessing stage of this system.

IV. Hough Lines Lane Detection

Once the edges are detected in the image, Hough Transform is used to find the lane lines. Each pixel in the image has a "vote", which is used to identify any prominent lines [5]. In matlab, the hough and houghpeaks functions are used for this purpose, shown in Fig. 14.

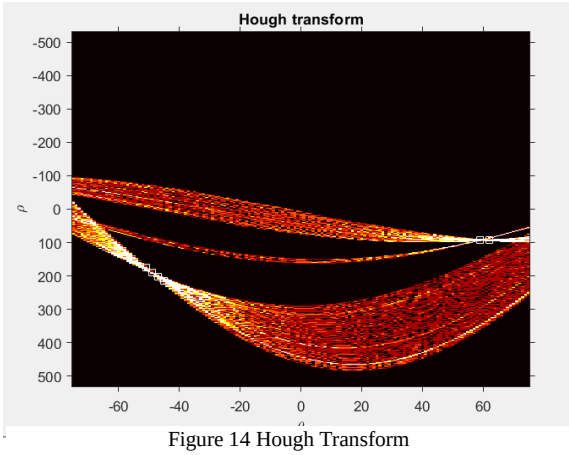

Figure 14 Hough Transform

To get the possible lane lines in the image, houghlines is used. These lines are then iterated through, and the 2 longest lines are compared to the current left/right lane line variables. The left/right lane line variables are set if the new line is longer than them (compared to left/right depending on slope of line). To increase stability, the average is calculated between the new and old lane line for both left and right, this is the chosen line. Also, if no new lane line is detected for this frame for either left or right, possibly due to a faded dashed line causing a large gap in the lane lines, the last known lane line is used (old left/right lane line). If the new left lane line has an x value greater than the right lane line x value, the last known lines are used. These selected lane lines for this image frame are then transformed and plotted on the full-scale image, extended to the image edges, shown in Fig. 15.

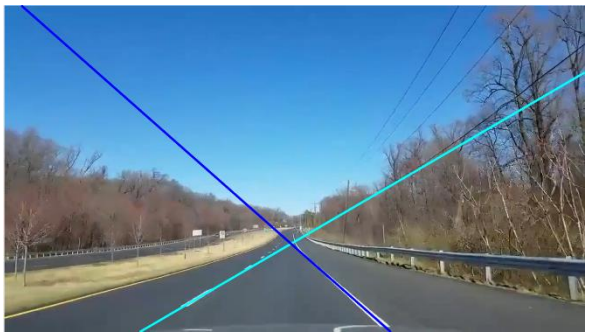

Figure 15 Hough Lines Chosen

V. Steering Angle Calculation

A. Plot Points of Interest

In order to make an approximate steering angle calculation, the lane line intersection point (red), car center point (green), lane center point (yellow), lane center line (yellow), lane center point ahead of car (pink) and lane line start points (white) are plotted on the image. These are all calculated and plotted via various Matlab functions, the output is shown in Fig. 16.

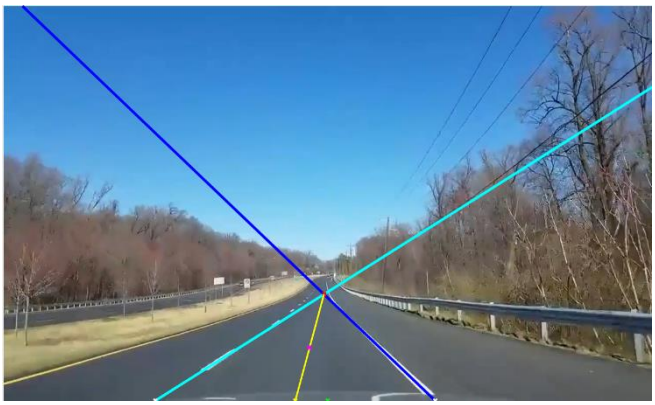

Figure 16 Points of interest plotted on image

B. Angle Calculation

To approximate the angle of steering required to keep the vehicle on the center lane line, the atan2d function is used. This calculates the angle between two points, in relation to

the horizontal x axis. This is used for calculating the angle between the car center point, and the lane center point ahead of the car, shown in Fig. 17. This angle represents the change in the vehicle direction required to center the vehicle at that point on the road ahead.

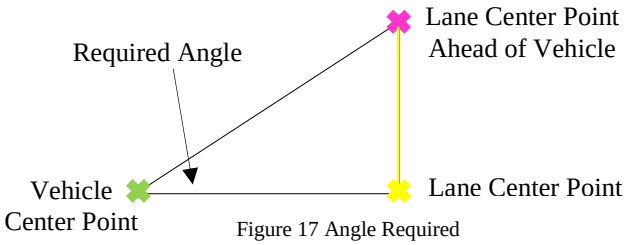

Figure 17 Angle Required

The Matlab function atan2d gives this angle within a [-180, 180] range. For steering angle of the vehicle steering wheel, I am using the angle descriptions as shown in Fig. 18. The use of this is as follows, the steering wheel is at 90° when the vehicle is driving straight, the wheel is at 0° when the vehicle is turning fully right, and the wheel is at 180° when the vehicle is turning fully left.

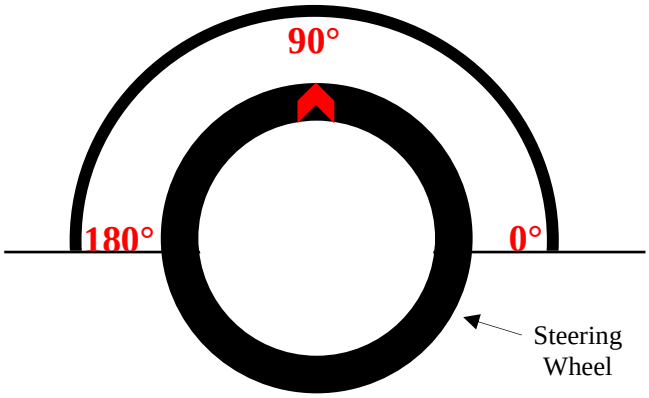

Figure 18 Steering Wheel Angle Reference

The sign of the atan angle depends on the values of x and y passed into the function, according to the diagram in Fig. 19.

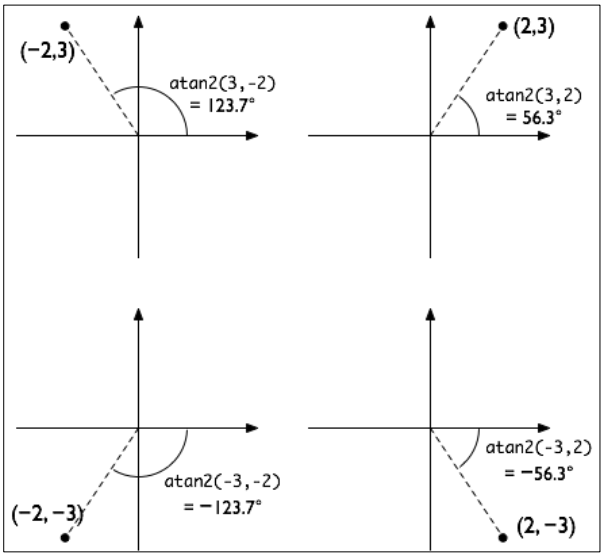

Figure 19 Example atan values

For my data, the atan x value is the difference between the lane center point ahead of vehicle x value, and the x value of the vehicle center point. The atan y value is the difference between the lane center point ahead of vehicle y value, and the y value of the vehicle center point.

For this application, due to Matlab plots having increasing y values in a downward direction, the y atan value will always be negative. From the diagram in Fig. 19, this indicates values will have a range [0, -180]. To rectify this and transform the angle to my steering wheel angle reference range [0, 180], the atan angle is inverted.

With this design, if the vehicle center (the origin point in Fig. 19) is to the right of the reference point, the angle will be obtuse. In contrast, if the vehicle center point is to the right of the reference point, the atan angle will be acute. If directly aligned, the angle will be 0°. To link this with the steering wheel reference, if the car is to the right of the lane center, the atan angle will be obtuse, which indicates the steering wheel should be turned to that angle, turning left. This is similar to an acute atan angle, the steering wheel will turn right, moving the vehicle towards the center of the lane. This angle is printed on the image along with the turn direction of the wheel, shown in Fig. 20, 21, 22.

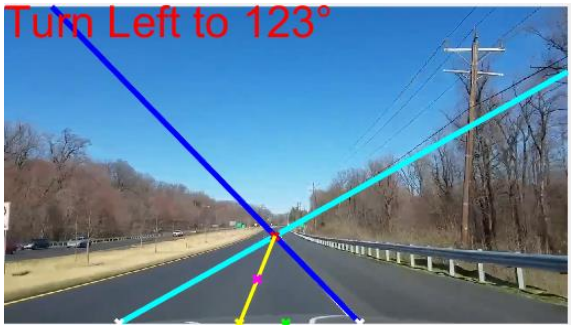

Figure 20 Left Steering Direction

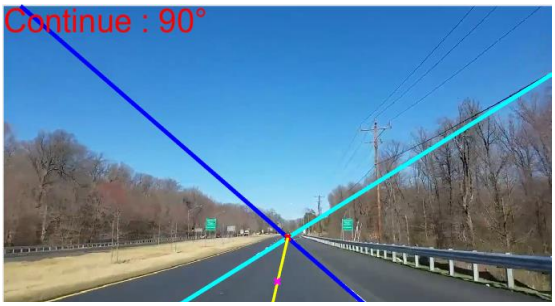

Figure 21 Continue Steering Direction

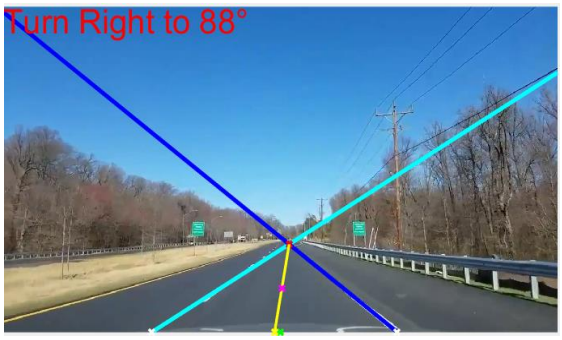

Figure 22 Right Steering Direction

VI. Simulink Model

A. *Lane Keep Assist System*

The above processing and calculations are all done via Matlab code, located in the lane_keep_assist function block in the model, shown in Fig. 24.

B. *Controller*

For the controller in this system, I made three different attempts at a suitable model.

1) Kinematic Steering

This model is a Simulink block, which takes the steering angle as input in radians. I tried to use this to model the vehicle wheels turning, by converting my angle in range [0,180], to the kinematic steering range [-90, 90], shown in Fig. 23. The angle of the wheels when the simulation runs is shown in Fig. 25 and Fig. 26. These wheel angles correspond to the video, as the lane detection calculates to turn left for most of the video, with a short period of turn right instructions towards the end.

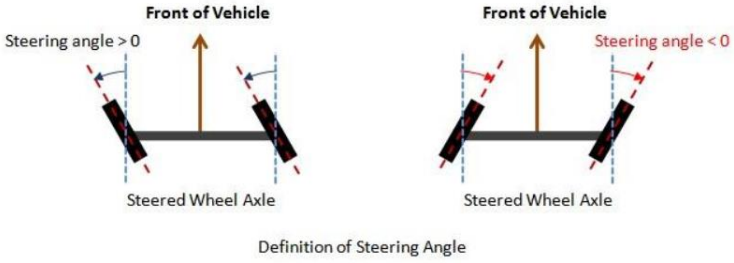

Figure 23 Steering Angles [6]

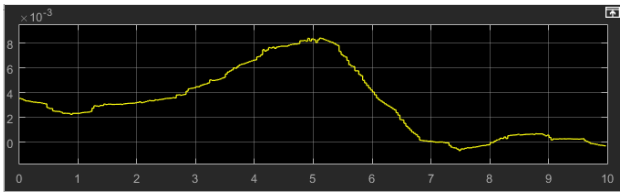

Figure 25 Left Wheel Angle

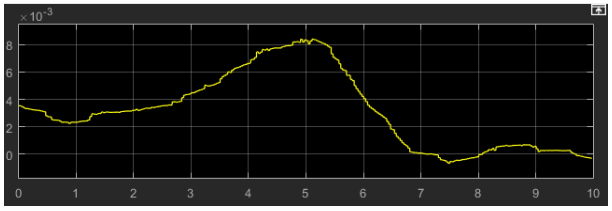

Figure 26 Right Wheel Angle

2) Subtraction Controller

My second attempt at a controller model involves a closed loop Simulink subtraction block, to get the difference between the required angle, and the current angle. I placed a Simulink convert block to simulate the motor changing the actual vehicle steering, with an output that would be the actual measured steering angle. This did not suit the project, as I could not simulate the actual steering changes.

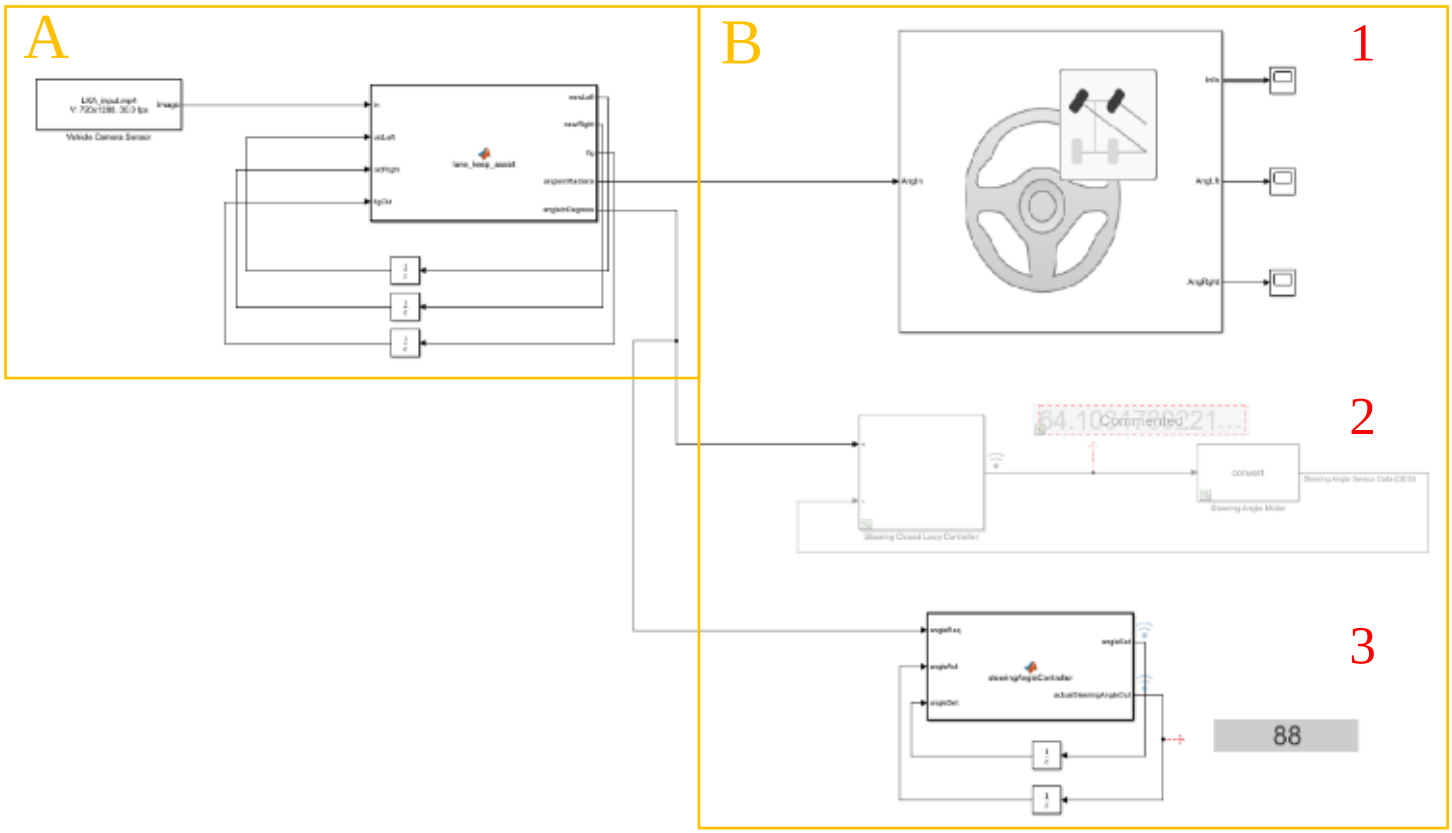

Figure 24 Simulink Model

3) Coded Closed Loop Controller

The third model I attempted uses a function block to simulate the controller of the steering system. This controller takes the required and measured steering angles as input, and calculates the required action to take. For example, if the required steering angle is greater than the measured angle, the actual angle is increased by 1°. Similarly, it is decreased by 1° if the measured angle is greater than the required angle. This simulates the wheel as it is being turned to the required angle. This works as expected, the display shows the actual steering angle increase and decrease to line up with the required angle.

VII. Benchmark Testing

The following benchmarks were set out before implementing this system.

A. Detect One Lane Line

This benchmark relates to the system detecting one lane line on the image provided. The system passes this benchmark, as it can detect one lane line on the image.

B. Detect Two Lane Lines

This benchmark relates to the system's capabilities in detecting two lane lines (left and right) on an image. The system succeeds in passing this benchmark, as seen the images above, two lane lines are detected on either side of the ego vehicle.

C. Detect White Lane Lines

This benchmark relates to the system detecting white road marking on the road in the image. This is important as this is the predominant colour used for lane lines. My system passing this benchmark, as it can detect white lines in the video.

D. Detect Non-White Lane Lines

This benchmark involves the system detecting other lane line colours, such as yellow. Unfortunately, due to binarizing the image with the selected threshold, white is the only colour line detected. This is an improvement to be made in future versions of the system, as it is required for an accurate LKA system. This failure is shown in Fig. 28.

Figure 27 Before Lane Detect

Figure 28 After Lane Detect

E. Detect Lane Lines from Head On Angle

This benchmark outlines that the system is required to detect lane lines from a typical head on angle, most commonly taken as input from camera sensors at the front of a vehicle. This benchmark is passed, as from the video input used, the angle is suitable for lane detection.

F. Detect Lane Lines from Various Angles

This benchmark outlines detecting lane lines from slightly different angled points of view. This is partially passed, as a test image of a varying angle produced successful results, however another image did not. This is due to the thresholds specified for angle of lines when setting left/right lane lines.

Figure 29 Failed Detection

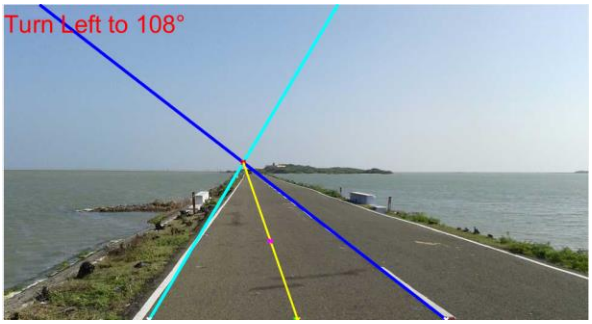

Figure 30 Successful Detection

G. Produce Accurate Steering Direction

This benchmark involves the calculation of the steering angle for the steering wheel. From the test video input, this benchmark is passed, as the turn directions and corresponding angles are accurate for my reference steering wheel turning angles and the lanes detected. The system indicates the extent in which to turn left, turn right, or continue straight, when required, shown in output images Fig. 20, 21, and 22 above.

H. Provide a Suitable Simulated Steering Controller

This benchmark relates to the steering controller that takes the lane keep assist calculated output as its input. This must simulate a controller for the steering system of a vehicle. This benchmark is partially passed. Although the system does not have a simulated vehicle moving corresponding to the controller decisions, I have implemented a Kinematic Steering example to show how the wheels may be turned, and have also implemented a coded simulated closed loop controller that indicates how to compare the actual steering wheel angle with the LKA instructed steering wheel angle.

VIII. Conclusion

Having completed this project, the lane keep assist system modeled in Simulink fits its purpose set out by the original goal. Most benchmarks established have been passed, which results in a system of accepted performance. Many improvements may be made to make the system suitable for

varying road conditions, which is outside the scope of this project.

REFERENCES

[1] OneShell UMD, Lane Detection Test Video 01. 2018.

[2] M. Samuel, M. Mohamad, S. Saad and M. Hussein, "Development of Edge-Based Lane Detection Algorithm using Image Processing", JOIV : International Journal on Informatics Visualization, vol. 2, no. 1, p. 19, 2018. Available: https://www.researchgate.net/publication/323198429_Development_of_Edge-Based_Lane_Detection_Algorithm_using_Image_Processing. [Accessed 20 December 2018].

[3] MathWorks, "bwmorph", Image Processing Toolbox User's Guide, 2005. [Online]. Available: http://matlab.izmiran.ru/help/toolbox/images/bwmorph.html. [Accessed: 28- Dec- 2018].

[4] Roborealm, "Roberts Edge", *Roborealm.com*. [Online]. Available: http://www.roborealm.com/help/Roberts_Edge.php. [Accessed: 20- Dec- 2018].

[5] U. Sinha, "The Hough Transform", *AI Shack*.

[6] "Definition of Vehicle Heading and Steeing Angle", Street.umn.edu. [Online]. Available: http://street.umn.edu/VehControl/javahelp/HTML/Definition_of_Vehicle_Heading_and_Steeing_Angle.htm. [Accessed: 01- Jan- 2019].

[7]"What are atan and atan2 used for in games?", Game Development Stack Exchange. [Online]. Available: https://gamedev.stackexchange.com/questions/14602/what-are-atan-and-atan2-used-for-in-games. [Accessed: 02- Jan- 2019].