Cairo University

Faculty of Engineering

Electronics and Electrical Communication Engineering Department

Real-time Lane Detection

Project Documentation Year (2017-2018)

Under Supervision

Dr. Hassan Mostafa

Dr. Hossam Hassan

Project's Team

Ahmed Mahmoud Hussein

Loay Ehab Mostafa

Mohamed Reda Sabek

Mostafa Mohamed Mohamed Abdel Aleem

Table of Contents

List of Figures

Acknowledgement

- First, we would like to express all our thanks to Allah for the great help in completing this project.

- We would like to express our sincere thanks to Dr. Hossam Hassan and Dr. Hassan Mostafa for them guidance, continuous encouragement and generous help throughout the development of this work.

- And we would like to offer our special thanks to Eng. Mostafa Refay from Valeo , Eng. Abdelrahman Abu-Taleb , Eng. Khaled Ahmed from Axxelera for them technical help.

- Finally, we would also like to express our love, gratitude and appreciation to our parents for their endless love, encouragement, patience and prayers during this work and behind.

Abstract

With the increase in the number of vehicles, many intelligent systems have been developed to help drivers to drive safely. Lane detection is a vital element of driver assistance systems. Lane detection process has several major challenges, such as attaining robustness to inconsistencies in lighting and background clutter. To address these issues in this project, several image processing, computer vision algorithms and Deep learning algorithms will be implemented.

Application of the project:

1. **Lane Detection**: the suitable lane for the vehicle will be detected with a real time process.

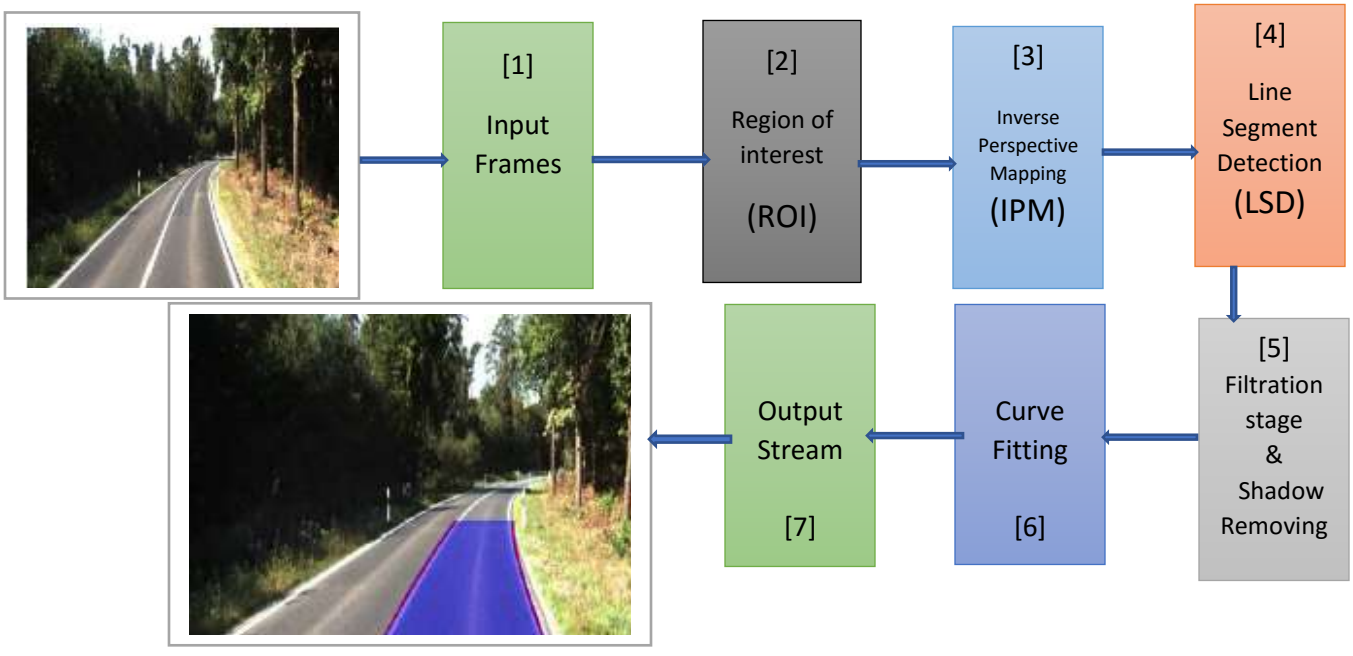
2. **Lane departure warning**:

Advanced Driver Assistance System (ADAS) provides safe and better driving. It helps to automate, adapt and enhance the driving experience. Most of the road accidents occur due to carelessness of driver. Advanced Driver Assistance System ensures safety and reduces driver workload. Whenever a dangerous situation is encountered, the system either warns the driver or takes active role by performing necessary corrective action to avoid an accident.

3. **Lane Keeping assist**:

Lane keeping assist is a feature that, in addition to lane departure warning system automatically take steps to ensure the vehicle stays in its lane.

Our Algorithm:



Chapter 1: Introduction

1. Problem Definition:

Nearly 1.3 million people die in road crashes each year, on average 3,287 deaths a day. An additional 20-50 million are injured or disabled. More than half of all road traffic deaths occur among young adults ages 15-44. Road traffic crashes rank as the 9th leading cause of death and account for 2.2% of all deaths globally. Up to 90% of accidents due to human factor.

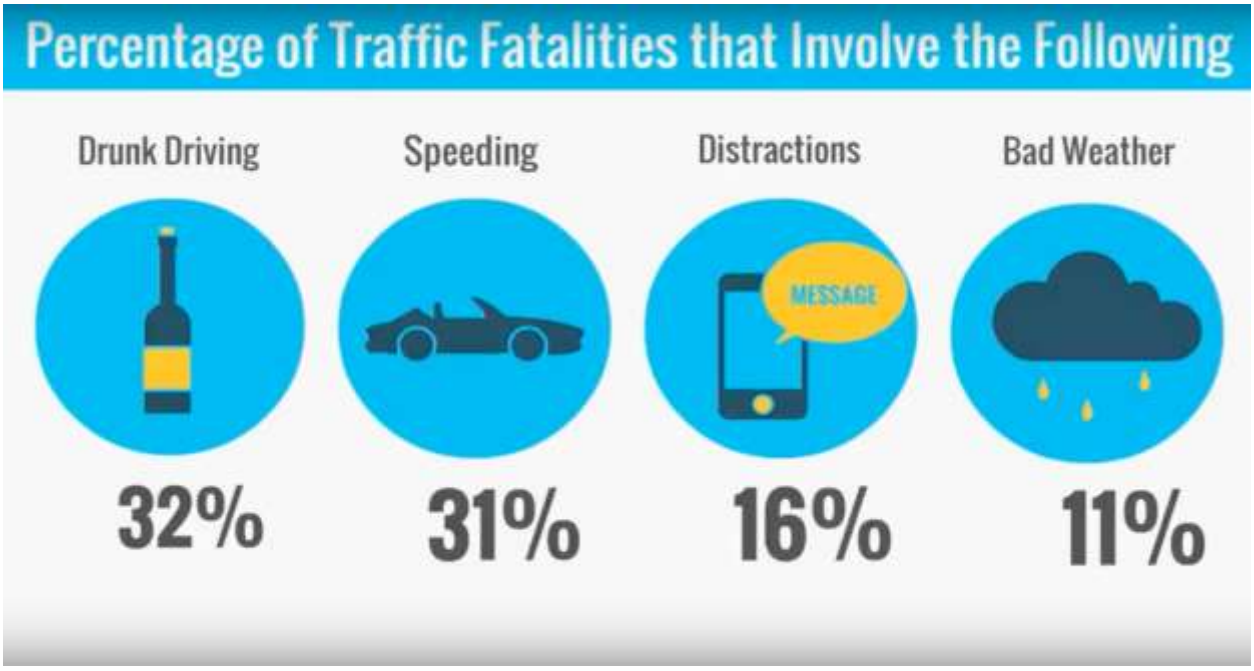


Figure 1:accidents as a percentage

Lane Departure represents the largest proportion as a reason for accident with 37%.

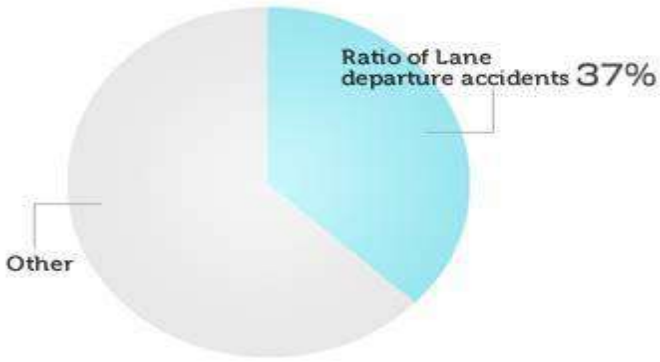


Figure 2:Ratio of Lane departure accidents as a reason of the accident

2. Problem impact:

For better, more secure driving and for luxury and with the advance of the technology and to save user time and effort, the companies produced Advanced Driver Assistance System (ADAS) for us. Lane Detection is a min block in ADAS system.

3.Who are the pioneers?

2008 | **Mohamed Aly** Real time and robust approach to detect lane markers in urban roads.
2008 | **Abdulhakam. AM. Assidiq, Othman O. Khalifa, Md. Rafiqul Isalm** and **Sheroz Khan** developed a vision based algorithm for lane detection.
2009 | **Chun-Wei Lin, Han-Ying Wang** and **Din-Chang Tseng** constructed an algorithm for lane detection with lateral inhibition and conjugate Gaussian model.
2009 | **Amol Borkar** updated a previous algorithm based on hough transform by incorporating an inverse perspective mapping and kalman filter.
2012 | **Amol Borkar, Monson Hayes** and **Mark T Smith** developed a new night-time lane detection system with efficient ground truth generation.
2013 | **H. Yoo, et al.** proposed an algorithm that is robust in illumination changes and suitable for both straight and curved roads.
2014 | **U. Ozgunalp and N. Dahnoun** proposed an algorithm that is robust in night and shadows.
2014 | **N.N. Ahmed Salim, X. Cheng and X. Degui**. A Robust Approach for Road Detection with Shadow Detection Removal Technique. Information Technology Journal, 13: 782-788.
2015 | Ammu M Kumar , Philomina Simon. Review of lane detection and tracking algorithms in advanced driver assistance system.
2016 | **Soonhong Jung**, **Junsic Youn** and **Sanghoon Sull** proposed an efficient method for reliably detecting road lanes based on spatiotemporal images.
2016 | **Bei He, Rui Ai**, **Yang Yan** and **Xianpeng Lang** proposed a Dual-View Convolutional Neutral Network framework for lane detection.
2016 | **Pallavi V. Ingale , Prof. K. S. Bhagat**. Comparative Study of Lane Detection Techniques.
2016 | Toan Minh Hoang and Kang Ryoung Park: Road Lane Detection by Discriminating Dashed and Solid Road Lanes Using a Visible Light Camera Sensor.

4.What has been accomplished?

2008 | Dr. Mohamed Ali published :Real time and robust approach to detect lane markers in urban roads

It first generates a top view of the road image using inverse perspective mapping, then the top view is filtered using selective oriented two dimensional Gaussian kernel, then the straight lines are detected using simplified Hough transform, which is followed by RANSAC line fitting which provides initial guess to the RANSAC spline fitting step, then a post processing step is done to localize the spline and extend it in the image. Note: This algorithm does not perform tracking. It can detect any number of lane boundaries in the image not just the current lane. In this paper, result is 20 fps with 90.89% correct rate according to the Caltech Dataset.

2013 | **N. Phaneendra** proposed a vision-based lane departure warning system. The main goal of this model was to implement an image processing algorithm for detecting lanes on the road and give a textual warning on departure from the lane. The lane departure decision making is based on distance between lanes and the center of the bottom in captured image coordinate, which needed less parameters. The lane detection performance has been improved by making use of Hybrid median filter with modified Hough transform, compared to the usual method of using Hough transform. The model proved to be efficient and feasible as compared to other systems.

Lane detection and tracking algorithm which can handle challenging scenarios such as faded lane markers, lane curvatures and splitting lanes In the initial step, a gradient detector and an intensity bump detector is used to eliminate the non-lane markers. Artificial Neural Networks (ANN) is applied on remaining samples for lane detection. The detected lane markers pixels are grouped using cubic splines. Hypotheses are generated from random set of line segments. RANSAC algorithm helps in validating the hypotheses. Particle filtering is used for lane tracking.

Variability of weather and road circumstances

Lane boundaries are successfully extracted in different conditions (sunny, cloudy, nighttime, shadowing, and rainy)

Note: Each developed algorithm works well on specific dataset, but do not work well on any other dataset. Also, some problems still not solved yet such as sharp curves in the foreground of the image and the accurate detection of the lanes under heavy rain also the captured frames are not that stable due to the vehicle movement.

2016 | Toan Minh Hoang and Kang Park published: Road Lane Detection by Discriminating Dashed and Solid Road Lanes Using a Visible Light Camera Sensor.
The model used a new algorithm using the approach of line segment detection – LSD instead of Hough Transform and also using fixed Region of interest – ROI instead of ROI based on vanishing points to reduce processing time. lanes always appear within the predetermined region of the image when the position and direction of the camera are fixed.

This approach gives high performance real-time output relative to Mohamed Aly approach and all previous approaches. The algorithm takes 33 mile second per frame processing but Mohamed Aly approach takes 50 mile second per frame with lower correct rate.

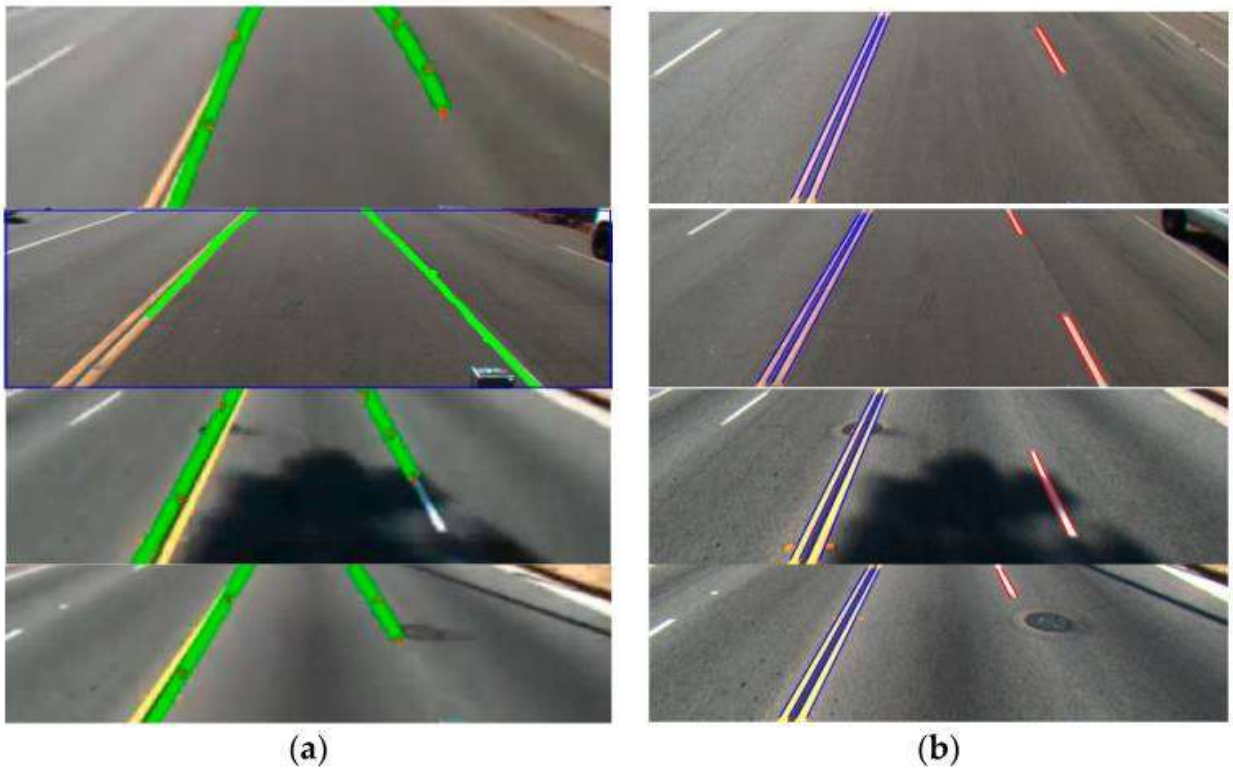


Figure 3:(a):M. Aly results , (b): LSD results

5. Project Description:

Lane detection is the process to locate lane markers on the road and then deliver these locations to an intelligent system. In intelligent transportation systems, intelligent vehicles cooperate with smart infrastructure to achieve a safer environment and better traffic conditions. The applications of a lane detecting system could be as simple as pointing out lane locations to the driver on an external display, to more complex tasks such as predicting a lane change in the instant future in order to avoid collisions with other vehicles. Some of the interfaces used to detect lanes include cameras, laser range images, LIDAR and GPS devices.

There are four aspects that need to be considered in the project: safety, accuracy, Real time response, of course cost and the ability to work in extreme conditions with results as good as possible.

There are various challenges in this field like parked and moving vehicles, bad quality lines, shadows of trees, buildings and other vehicles, sharper curves, irregular lane shapes, merging lanes, writings and other markings on the road, unusual pavement materials, dissimilar slopes causes problems in lane detection and the ability to produce satisfactory results in a real time.

Also, gaps in the existing work include nowadays various lane detection algorithms have been used for assisting the driver in Advanced Driver Assistance System (ADAS). Majority of these techniques have focused on the detection of straight lanes and curved lanes have been ignored. Thus, the gaps which exist in the literature are:

1. Majority of work is based on straight lane images i.e. curved lane images have been ignored.
2. The improvement in Hough Transform has been ignored for better Lane Detection.
3. The effect of fog and shadow in Lane Detection has also been ignored.

6. Approach:

Many approaches have been applied to lane detection, which can be classified as either feature-based or model-based. Feature-based methods detect lanes by low-level features like lane-mark edges. The feature-based methods are highly dependent on clear lane-marks, and suffer from weak lane marks, noise and occlusions. Model-based methods represent lanes as a kind of curve model which can be determined by a few critical geometric parameters. The model-based methods are less sensitive to weak lane appearance features and noise as compared to feature-based methods. But the model constructed for one scene may not work in another scene, which makes the method less adaptive. Additionally, for best estimation of model parameters, an iterative error minimization algorithm should be applied, which is comparatively time-consuming.

We will use the best approach which saves time and effort and gives the best result after experimenting different approaches.

The general method of lane detection is to first take an image of road with the help of a camera fixed in the vehicle. Then the image is converted to a grayscale image in order to minimize the processing time. Secondly, as presence of noise in the image will hinder the correct edge detection. Therefore, filters should be applied to remove noises like bilateral filter, Gabor filter, trilateral filter Then the edge detector is used to produce an edge image by using canny filter with automatic thresholding to obtain the edges. Then edged image is sent to the line detector after detecting the edges which will produces a right and left lane boundary segment. The lane boundary scan uses the information in the edge image detected by the Hough transform or Line Segment Detection (LSD) to perform the scan. The scan returns a series of points (line segments) on the right and left side. Finally, pair of hyperbolas is fitted to these data points to represent the lane boundaries. For visualization purposes the hyperbolas are displayed on the original color image.

The algorithm undergoes various changes and detection of patterns in the images of roads for detecting the lanes.

Beside the computer vision algorithms, we will use Deep learning algorithms using convolutional neural network - CNN to implement Real-time Lane Detection comparing between the two approaches. Till now, the computer vision approaches achieve better performance in the application of the Lane Detection.

7. Objectives and Outcomes:

1. The goal of this work is to offer a real time product that is compatible with multiple environments and reduce the gaps in the existing work and to face the challenges that are imposed on the field.

2. As it is implied this is a part of a big project that aims to Driver assistance in autonomous cars, so we work on this module while other teams work on other modules like car detection so we aim to integrate our work in a full autonomous car that contain all these modules.

3. The product will enhance safety of the driver and of other cars and pedestrians too and contributes to decreasing the rate of accidents in highways.

4. Improve the algorithms used in the project to give better speed to have the ability to work in a practical environment.

8. Market

Automated driving is increasingly being considered the key technology to address societal problems caused by the proliferation of automobiles around the world. The development of automated driving has been ongoing since at least the 1950s. However, it has accelerated in the last decade, enabled by advancements in computational architectures and sensing technology, along with dramatic cost reductions. These advancements, combined with vehicle electrification and ubiquitous connectivity, are enabling automated driving to rapidly become viable.

Figure 4:Full autonomous car manufacturers

Pioneers: GM, FORD, Daimler, Tesla, Waymo.

Future of Autonomous Cars:

In 2015, full Autonomous vehicles (AVs) are being developed for consumers.

By 2030, Consumers begin to adopt full autonomous cars. The after-sales service landscape is reshaped, Insurers shift from covering individuals to covering technical failures, Supply chain and logistics are redefined.

By 2050, full autonomous cars become the primary means of transport. AVs free up to 50 minutes a day for drivers. Parking space is reduced by billions of square meters. Vehicle crashes fall by 90%, saving billions of dollars. AV technology accelerates development of robots for consumer use.

9. Project Applications

1. **Lane Detection**: the suitable lane for the vehicle will be detected with a real time process.

2. **Lane departure warning**:

Advanced Driver Assistance System (ADAS) provides safe and better driving. It helps to automate, adapt and enhance the driving experience. Most of the road accidents occur due to carelessness of driver. Advanced Driver Assistance System ensures safety and reduces driver workload. Whenever a dangerous situation is encountered, the system either warns the driver or takes active role by performing necessary corrective action to avoid an accident.

3. **Lane Keeping**:

Lane keeping assist is a feature that, in addition to lane departure warning system automatically take steps to ensure the vehicle stays in its lane.

10. Tools

- Python Programming language.
- C++11 Programming language.
- g++ (C++ compiler version 5.4).
- CMake (version 3.8)
    - CMake is an open-source, cross-platform family of tools designed to build, test and package software.
    - CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native *Makefiles* and workspaces that can be used in the compiler environment.

- OpenCV library (version 3.2).
- Contains hundreds of programming functions mainly aimed at real-time computer vision. Uses OpenBLAS (Open source implementation of Basic Linear Algebra Subprograms).
- Supports OpenCL, OpenGL and CUDA.
- OpenCV is used in Camera calibration, image processing, image and video I/O, basic GUI.
- Data analysis (classification/regression, including neural networks).
- Optimization and nonlinear solvers.
- Interpolation and linear/nonlinear least-squares fitting.
- Linear algebra (direct algorithms, EVD/SVD), direct and iterative linear solvers, Fast Fourier Transform and many other algorithms (numerical integration, ODEs, statistics, special functions).
- GNU Bash.
- Ubuntu (version 14.04).
- Nvidia Jetson TX2 Embedded Kit.
- ELP Camera (3MP – 2.8-12mm – USB 2.0).
- Tornado 17” LCD.
- Opencv for cuda

11. Project Implementation

Proposed System:

- A car mounted camera that captures a video feed.
- Embedded kit where the algorithm is executed.
- LCD monitor to display a GUI.
- An accurate and fast lane detection algorithm that is capable to work in different conditions of road and weather.

Such a system can be integrated with other ADAS systems for autonomous cars manufacturing.

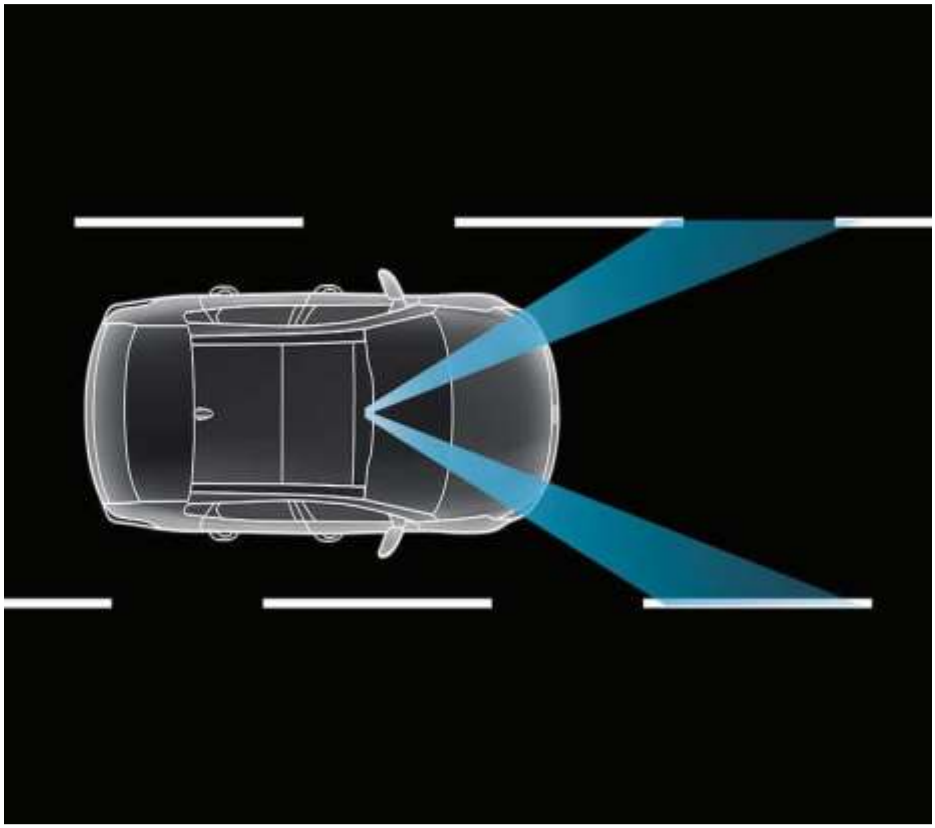


Figure 5: Project implentation

Chapter 2: Software Layer:

Computer vision approaches:

1. Mohamed Aly ‘s paper

Paper overview:

Mohamed Aly’s paper is the first robust paper in the field of Lane Detection using computer vision approach which published in 2008. It is considered as a common reference for the future papers and researches in the field of Lane Detection.

Lane Detection has received considerable attention since the mid-1980s. Techniques used varied from using monocular to stereo vision using low-level morphological operations to using probabilistic grouping and B-snakes. However, most of these techniques were focused on detection of lane markers on highway roads, which is an easier task compared to lane detection in urban streets. Lane detection in urban streets is especially a hard problem. Challenges include: parked and moving vehicles, bad quality lines, shadows cast from trees, buildings and other vehicles, sharper curves, irregular/strange lane shapes, emerging and merging lanes, sun glare, writings and other markings on the road (e.g. pedestrian crosswalks), different pavement materials, and different slopes.

This paper presents a simple, fast, robust, and effective approach to tackle this problem. It is based on taking a top-view of the image, called the Inverse Perspective Mapping (IPM). This image is then filtered using selective Gaussian spatial filters that are optimized to detecting vertical lines. This filtered image is then thresholded robustly by keeping only the highest values, straight lines are detected using simplified Hough transform, which is followed by a RANSAC line fitting step, and then a novel RANSAC spline fitting step is performed to refine the detected straight lines and correctly detect curved lanes. Finally, a cleaning and localization step is performed in the input image for the detected splines.

This paper provides a number of contributions. First of all, it’s robust and real time, running at 50 Hz on 640x480 images on a typical machine with Intel Core2 2.4 GHz machine. Second, it can detect any number of lane boundaries in the image not just the current lane. it can detect lane boundaries of neighboring lanes as well. This is a first step towards understanding urban road images. Third, it presents a new and fast RANSAC algorithm for fitting splines efficiently. Finally, the paper presents a thorough evaluation of our approach by employing hand-labeled dataset of lanes and introducing an automatic way of scoring the detections found by the algorithm.

The Algorithm used in Mohamed Aly’s paper:

1. IPM
2. Image filtering and thresholding
3. Progressive Probabilistic Hough Transform (PPHT)
4. RANSAC line fitting

Results:

Figure 6: M. Aly results

Conclusion:

The paper proposed an efficient, real time, and robust algorithm for detecting lanes in urban streets. The algorithm is based on taking a top view of the road image, filtering with Gaussian kernels, and then using line detection and a new RANSAC spline fitting technique to detect lanes in the street, which is followed by a post-processing step. The algorithm can detect all lanes in still images of urban streets and works at high rates of 50 Hz. We achieved comparable results to other algorithms that only worked on detecting the current lane boundaries, and good results for detecting all lane boundaries.

1.1. Input Stream

The algorithm runs on a demonstrating datasets videos, thus the first stage is to read the video frames and apply the algorithm on these video frames.
Input Stream stage can be initialized by video file or camera device. The input frames are resized to 800x480 to suit our GUI's size and to achieve better performance. Dynamic frame rate (FPS) is used to deliver real-time experience to the user. Dynamic frame rate is calculated from the difference between actual inter-frame time and processing time in this stage.

1.2.IPM (Inverse Perspective Mapping)

It is the first step in this approach and it's main roles are

- Get rid of the perspective effect in the image, and so lanes that appear to converge at the horizon line are now vertical and parallel. This uses our main assumption that the lanes are parallel (or close to parallel) to the camera.
- We can focus our attention on only a sub-region of the input image, which helps in reducing the run time considerably.

Figure 8: Input Frame

Figure 7:output Frame after IPM

Pin hole camera model:

The pinhole camera model describes the mathematical relationship between the coordinates of a 3D point and its projection onto the image plane of an ideal pinhole camera, where the camera aperture is described as a point and no lenses are used to focus light.

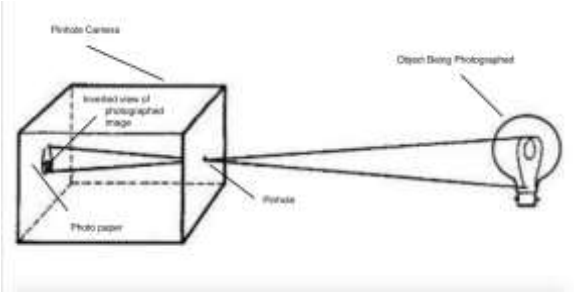


Figure 9: Pin hole camera model

In the previous figure, a simple illustration to the pin hole model is shown, containing an object in 3D -real world- and how the image is formed.

Perspective Imaging:

In the following figure, the shown coordinate system has the image plane introduced in front of the COP -center of projection- so that the image will not be inverted, which is mathematically easier to deal with.
The world point $(x,,z)$ is mapped to the point (x',y',d) on the image plane.
To calculate a point on image plane that is corresponding to a point in the world we need to calculate x' and y'.
From the similar triangles (**ABC** and **ADE**), we get that: $(x',',z')=(-d\ x/z,-d\ y/z,-d)$
We get the projection (the location of the point on the image plane) by throwing out the last coordinate (image plane is 2D), so the image plane point is $(-d\ x/z,-d\ y/z)$.

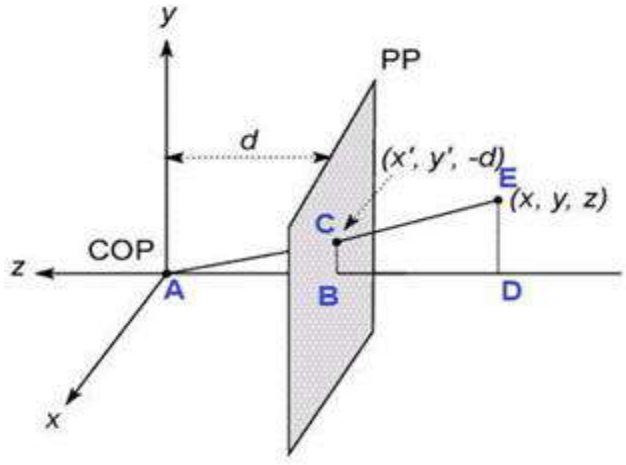


Figure 10 IPM: Mapping point from 3D to 2D

The perspective projection is represented by this matrix multiplication

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z/f \end{bmatrix} \Rightarrow \left(f\frac{x}{z}, f\frac{y}{z} \right) \Rightarrow (u, v)$$

Homogeneous coordinates:

To get an image plane point, we divided x and y by z, the issue is that this transformation is not a linear transformation as z is not constant. Division by constant is a linear operation, so to solve this problem we add one more coordinate which is a constant number, which is called homogeneous coordinates.

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad (x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Image to Image Projection:

A geometric transform is a vector function T that maps the pixel ($\boldsymbol{x}$,) to a new position ($\boldsymbol{x}'$,$\boldsymbol{y}'$). The transformation equations are either known in advance such as translation, rotation or can be determined from known original and transformed images such as affine and homoghraphy transformations.
Homoghraphy transformations is what we care about.

it is represented by this matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \cong \begin{bmatrix} sx' \\ sy' \\ s \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective Transformation:

A transformation that maps lines to lines but does not necessarily preserve parallelism as shown in figure 22, using 3x3 matrix called homography matrix

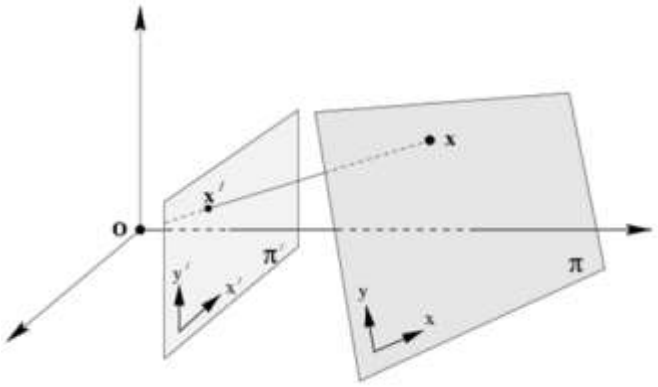


Figure 11: Projection

Least Squares Method:

Least squares method is used to solve and find values of homoghraphy matrix to do the necessary plan transformation.
Consider having the equation: $A\,\bar{x}=\bar{b}$
Where A = $[\bar{a}1\ \bar{a}2\cdots\bar{a}k]$, $\bar{x}$ = transpose($[x1x2\vdots xk]$) and you want to find $\bar{x}$ which satisfies the equation. $x1\ \bar{a}1+\ x2\ \bar{a}2+\ \ldots+xk\bar{a}k=\ \bar{b}$
The problem is that no linear combination for $x1\ \bar{a}1$, $x2\ \bar{a}2$, … , $xk\bar{a}k$ can give $\bar{b}$.
$\bar{b}$ is not in the (A) as illustrated in the following figure.
To solve this problem, find $\bar{\boldsymbol{x}}*$**,** where $\boldsymbol{A}\bar{\boldsymbol{x}}*$ is as close to $\bar{\boldsymbol{b}}$ as possible. The closest vector in the subspace (A) to a vector $\bar{\boldsymbol{b}}$ outside. This subspace is the projection $\bar{\boldsymbol{b}}$ onto (A) as illustrated in the following figure.
$\bar{\boldsymbol{x}}*$ is called *least square solution*.

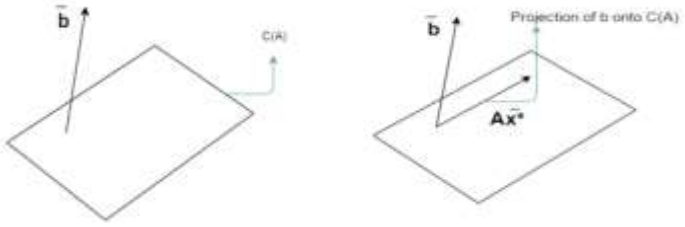


Figure 12: IPM: Least square projection

$A\bar{x}_* = proj_{C(A)}\bar{b}$
$A\bar{x}_* - \bar{b} = proj_{C(A)}\bar{b} - \bar{b}$
$proj_{(A)}\bar{b} - \bar{b}$=orthogonal to $C(A)$ (shown in the next figure). $\therefore A\bar{x}_* - \bar{b} \in$Set of all vectors orthogonal to all vectors $in\ (A)$
$\therefore A\bar{x}_* - \bar{b} \in (A_T)$, $N(A_T)$ $null\ space\ of\ A$ transpose $then\ A_T(A\bar{x}_* - \bar{b})=0$ $A_TA\bar{x}_* - A_T\bar{b}=0$
$A_TA\bar{x}_* = A_T\bar{b}$, and this will always have a solution.
Finally, to get the homoghraphy matrix, solve the equation: $A_TA\bar{x}_* = A_T\bar{b}$

Where $\bar{x}_*$ is the homoghraphy matrix, $\boldsymbol{A}$ is the matrix of input points and $\bar{\boldsymbol{b}}$ is the matrix of destination points.

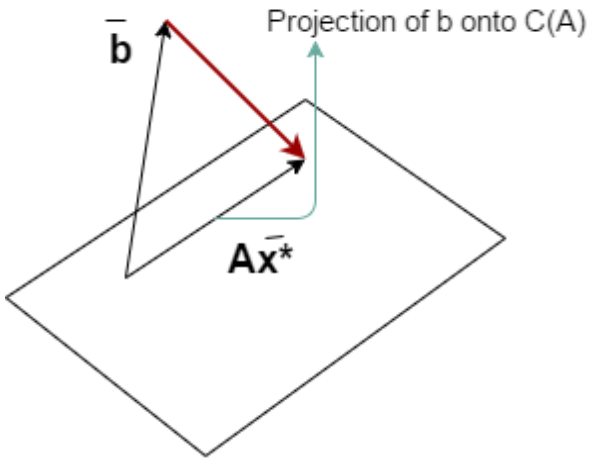


Figure 13: Projection of b

Results

Figure 15: Input Frame

Figure 14:Output Frame after IPM

1.3. Canny Edge

Canny Overview

Finds edges by looking for local maxima of the gradient of **f(x, y)**. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. Therefore, this method is more likely to detect true weak edges.

Canny Algorithm

- The image is smoothed using a Gaussian filter with a specified standard deviation, to reduce noise.
- The local gradient, **[gx2 + gy2]** and edge direction, **tan-1 (gy /gx)**, are computed at each point. An edge point is defined to be a point whose strength is locally maximum in the direction of the gradient.
- The edge points determined in the previous step give rise to ridges in the gradient magnitude image. The algorithm then tracks along the top of these ridges and sets to zero all pixels that are not actually on the ridge top to give a thin line in the output, a process known as non-maximal suppression.

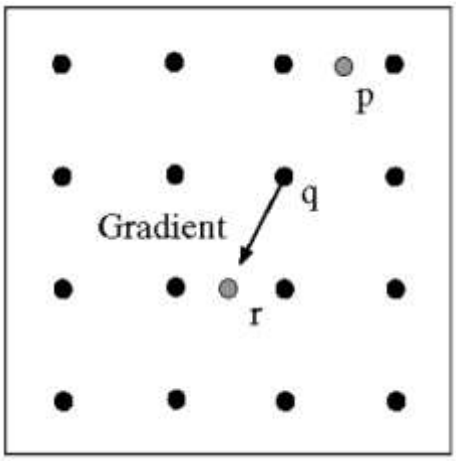


Figure 16:Canny: Non-maximal suppression

At **'q'** we have a maximum if the value is larger than those at **'p'** and **'r'**.

Figure 19: origional image

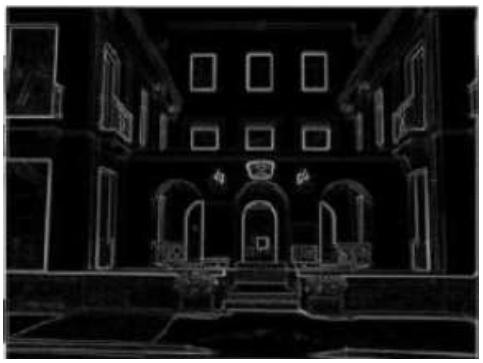


Figure 17 : Gradient Magnitude

Figure 18 : Non-maximum suppression

Then, A threshold is applied on the edge pixels, by so-called hysteresis thresholding, which is based on using two thresholds, **T1** and **T2**, with **T1 < T2**. Ridge pixels with values greater than **T2** are said to be "strong" edge pixels. Ridge pixels with values between **T1** and **T2** are said to be "weak" edge pixels.

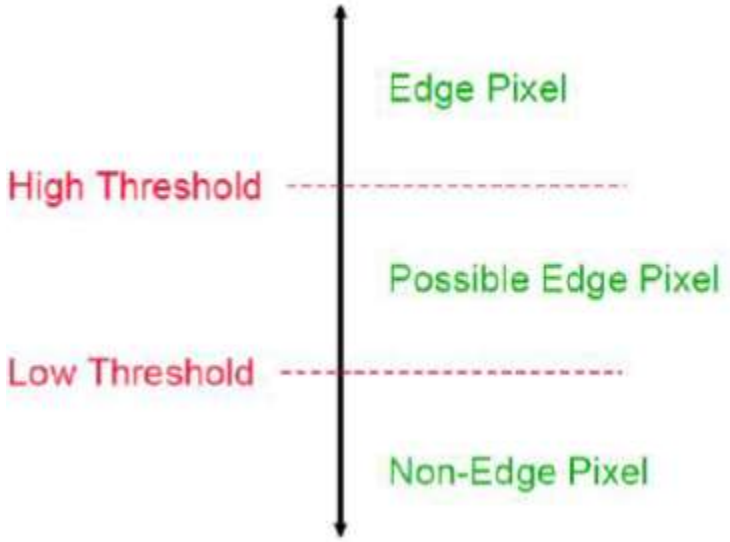


Figure 20: Canny: Hysteresis threshold

Algorithm of canny edge:

1. convert pic to gray scale.
2. Noise reduction using Gaussian filter.
3. Compute gradient magnitude and angle
4. Non-maximum suppression
5. Hysteresis thresholding.

Figure 21: step 1 :After grayscale nad Gaussian filter

Figure 22: step 2 : Compute gradient magnitude and angle

Figure 23: step 3:Non-maximum suppression

Figure 24: step 4:Hysteresis thresholding

1.4. Hough Transform

Hough Transform Overview
In practice, the resulting pixels of edge detection seldom characterize an edge completely because of noise, breaks in the edge from non-uniform illumination, and other effects that introduce spurious intensity discontinuities. Thus, edge detection algorithms typically are followed by linking procedures to assemble edge pixels into meaningful edges. One approach for linking line segments in an image is the Hough transform.

Linear Representation
Many lines pass through **(xi, yi)**,all of which satisfy the slope-intercept line equation $yi = axi + b$ for some values of **a** and **b**. Writing this equation as $b = -axi + yi$ and considering the ab-plane (also called parameter space). A second point **(xj, yj)** also has a line in parameter space associated with it, and this line intersects the line associated with **(xi , yi)** at **(a', b')** where **a'** is the slope and b' the intercept of the line containing both **(xi , yi)** and **(xj, yj)** in the xy-plane. In fact, all points contained on this line have lines in parameter space that intersect at **(a', b')**.
As result any point in xy-plane is represented by a line in parameter/ Hough space, and any line in xy-plane is represented by a point in parameter/ Hough space.
The problem in the linear representation is that **a** and **b** can take infinite values. Thus the polar representation of line is used to overcome this problem.

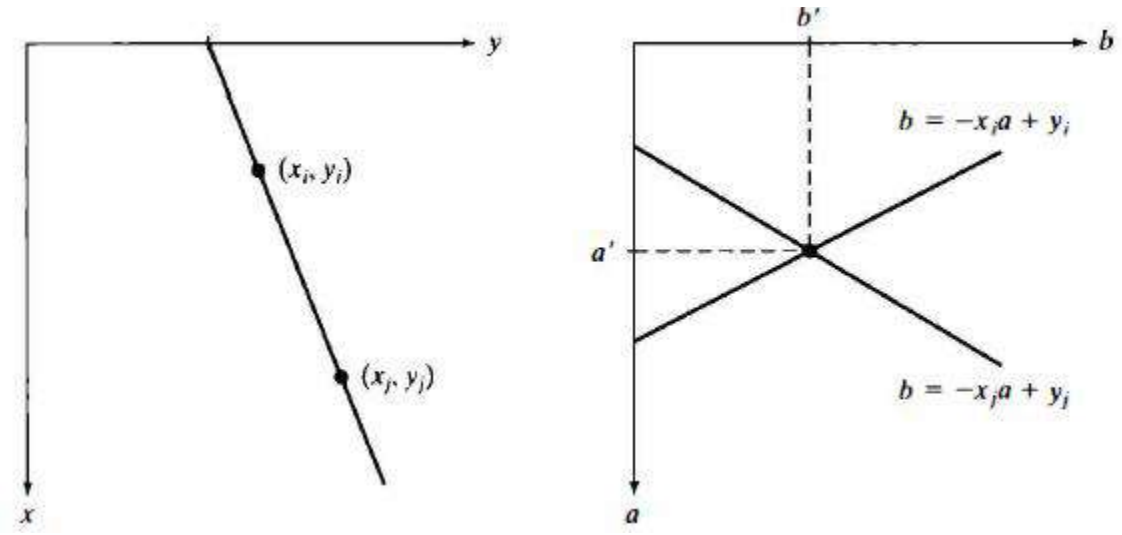


Figure 25:Hough Transform: x-y plane and parameter space

Polar Representation

The Polar representation of a line:

$$x\cos\theta + y\sin\theta = \rho$$

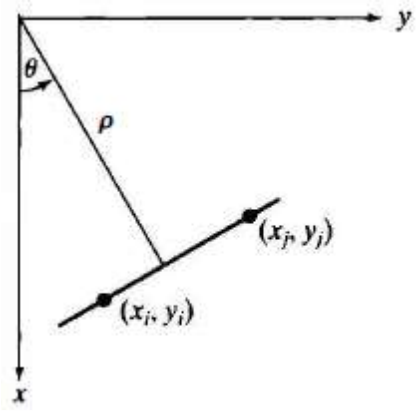


Figure 26: Hough Transform: Parameterization of line in x-y plane

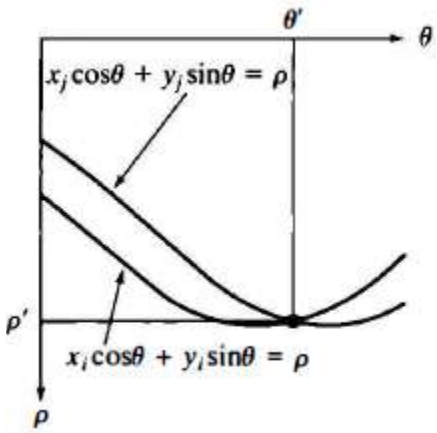


Figure 27: Hough Transform: ρϑ-plane

The intersection point **(ρ', θ')** corresponds to the line that passes through both **(x_i, y_i)** and **(x_j , y_j)**. In order to pick the intersection point, the *ρϑ -plane* is divided into cells called accumulator cells.

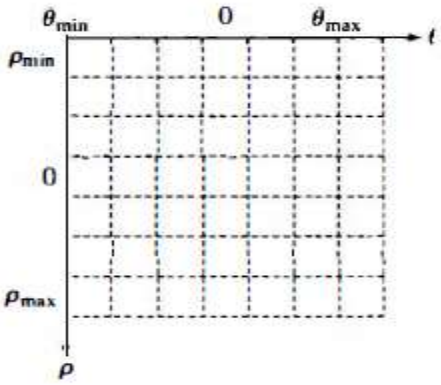


Figure 28:Hough Transform: Division of the ρϑ-plane into accumulator cells

Where **[ρmin, ρmax]** and **[θmin, θmax]** are the expected ranges of the parameter values. Usually, the range of values is $-D < p < D$ and $-90° < \theta < 90°$, where **D** is the farthest distance between opposite corners in the image.
Let each edge point vote in image space vote for a set of possible parameters in Hough space, then accumulate votes in discrete set of cells, parameters with the most votes indicate line in image space (xy-plane).

Hough transform Algorithm

- Initialize **H[ρ, θ] = 0**, where **H** is an accumulator array (votes).
- For each edge point **I[x, y]** in the image:

For each **θ = [θmin to θmax]**, calculate $\rho = x \cos\theta - y \sin\theta$ then increment **H[ρ, θ]**.

- Find the values of **(ρ', θ')** where **H[ρ', θ']** is maximum.
- The detected line in the image is given by $\rho' = x \cos\theta' - y \sin\theta'$.

Hough transform results

Figure 29: Lane detection ouptput using Hough transform

2. Kang Park and Toan Minh paper (LSD paper):

This paper (Road Lane Detection by Discriminating Dashed and Solid Road Lanes Using a Visible Light Camera Sensor) implemented by Kang Park and Toan Minh achieved high real time performance comparing with Mohamed Ali paper performance. This algorithm was robustly implemented in 2016. We added some blocks to the algorithm to improve the algorithm performance.

1. ROI (region of interest)
2. LSD (Line Segment Detection) & curve fitting.
3. Filtration

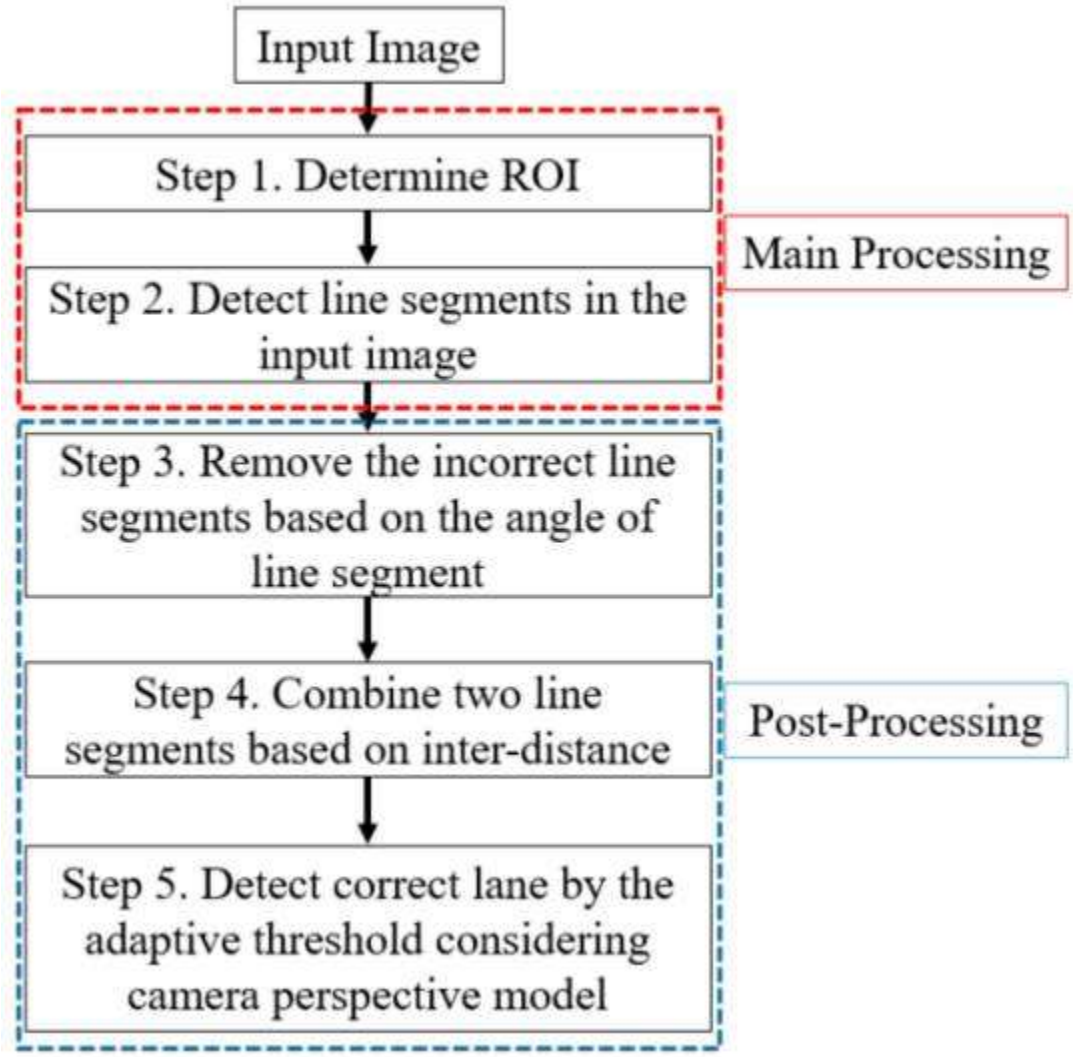


Figure 30: Road Lane Detection by Discriminating Dashed and Solid Road Lanes Using a Visible Light Camera Sensor

Postprocessing

Filtration

The detected lines in photo contain a lot of incorrect lines in regions of photo outside the lanes and those need to be removed and filtration is done here with high weight compared to MALY paper and the next used approach because this approach doesn't use IPM so it doesn't have a big perspective view of the photo to get the desired area only. so photo is divided into 2 parts : a lower part with y coordinates from one third of the height of the photo to the height of the photo and a higher part which extends from the 0 coordinated of the photo (origin point at top left) and it extends to one third of the height of the photo each has different approaches in filtration of line segments as the lower part it has an easier approach as lines there are filtered due to their length and orientation or angle and the ranges are found practically to be from -75 to -60 for left region of the lane and from 35 to 50 for the right part in lower region of the area.

While the higher part of the photo is filtrated by 2 approaches the first one uses threshold of angles determined from the information obtained from the lower part of the photo and if no lines were detected in the lower part of the photo. Thresholds are set from trial and error from the statistics of tried photos before to find the angle (trial and error). While the second approach is more similar to region growing algorithm, which is used for image segmentation in general where first the lines are sorted from bottom to top to keep starting from the part which was previously processed by the code and then a seedline is found and the seedline is chosen as the line with the most resemblance in angle or orientation with the lower part of photo which was previously processed by the algorithm and the average angle of the filtered lines in the lower part is found and seedline is chosen such that it is the line with the least absolute difference to the lines detected in the lower region and then lines of the top region of the photo are scanned with their order and the line with the smallest absolute difference to the seedline are pushed into a vector of the filtered lines and then is chosen as a seedline and so on.

Line combination algorithm

Lines filtered are combined and determined if they are a curve or line due to the difference in orientation between the two line to be combined and distance between the starting point of the top line and the ending point of the bottom line as if the difference between the angle of two lines is less than or equal a certain threshold and thresholds are found practically to be 2 degrees for the difference in angle and 3 pixels for the difference between the starting point and the ending point.

And based on these differences the lines are combined for several cases shown in the figure below

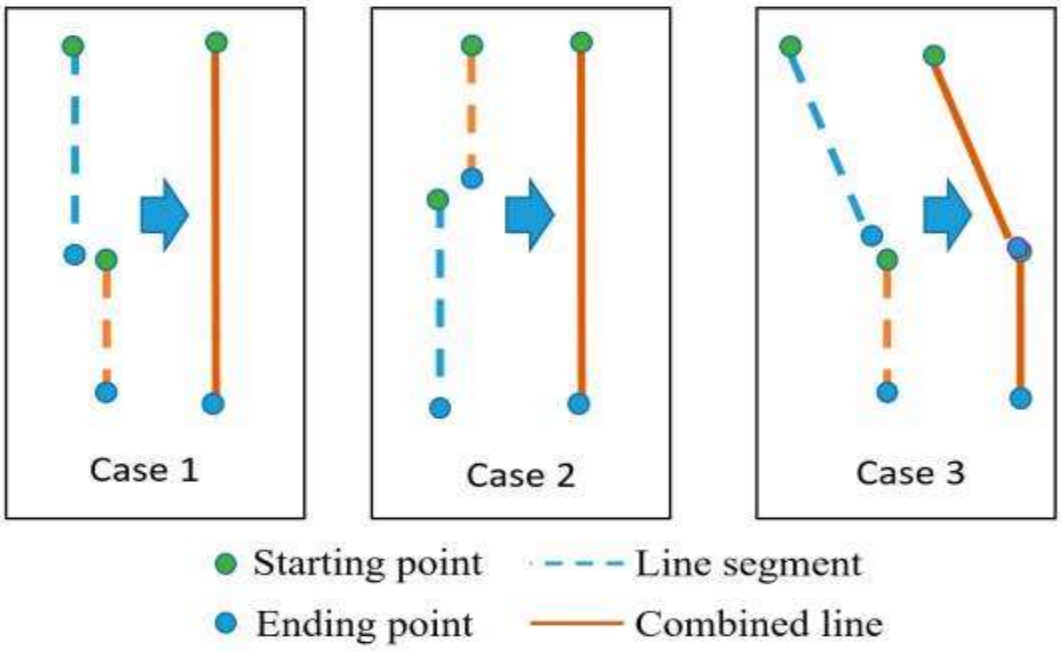


Figure 31: Line combination cases

some results:

Figure 32:LSD results for curved frame

2.1. Region of interest (ROI)

The input images usually include unnecessary information and road conditions need to be analyzed only in a region of interest (ROI) to reduce the amount of computation. So, we can focus our attention on only a sub-region of the input image, which helps in reducing the run time considerably and Processing resources. This block wasn't included in the Mohamed Aly's algorithm but we will use ROI to provide real-time approach.
There are commonly three ROI approaches used for Lane Detection algorithms. We studied them carefully.

2.1.1. Vanishing point approach:

1. The input here is the output from the block used for line segment detection (LSD or PPHT).
2. Divide the input into left region and right region, getting left/right lines.
3. We will extend every left/right line and get all the intersection points between the left lines and right lines. So, Num. of intersection points = Num. of right lines X Num. of left lines
4. We will consider Threshold Num. of intersection points and moving sliding window (41X41) with step of 15 pixels to catch the threshold Num. of intersection points and if the sliding window catches the threshold or higher it will approx. the position of the vanishing point. -Threshold Num. of intersection points = Num. of right lines X Num. of left lines /4
5. We will extend vertical line which pass on the vanishing point and parallel to the central line, we will consider this line is central lane line.

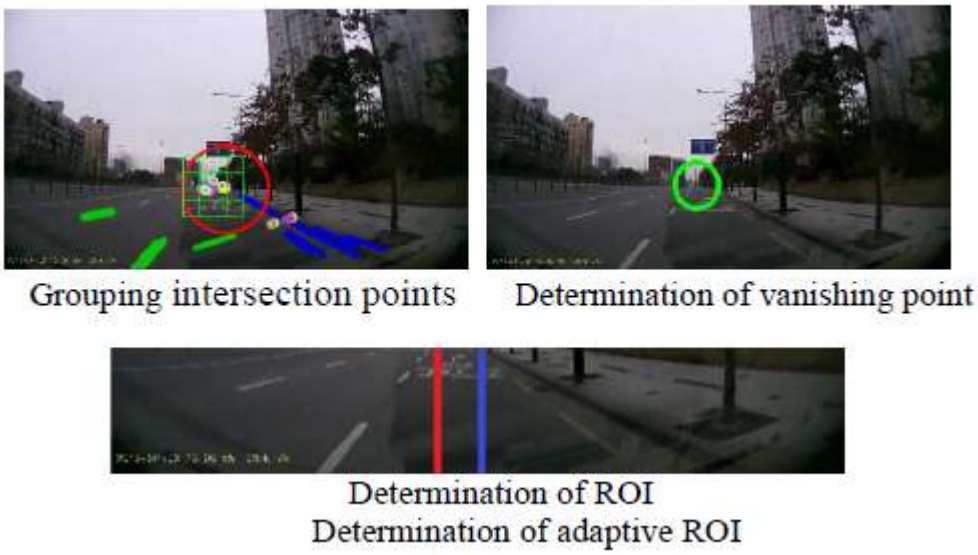


Figure 33:steps to determine ROI based on vanishing point

Output :

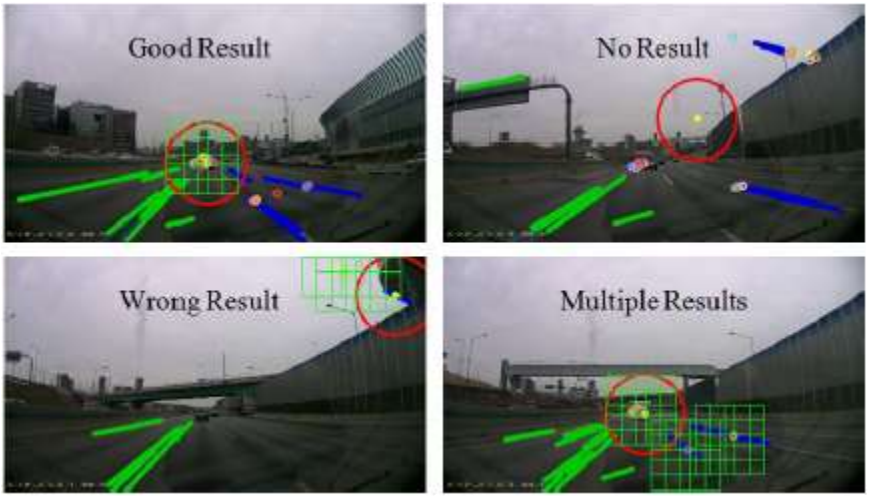


Figure 34:Results of vanishing point ROI

Advantage:

1. It can be used for each vehicle regardless its height and the position of the camera.
2. In the highway or in fixed roads the vanishing point is almost fixed. So, if we used simple tracking method we will easily predict the vanishing point and the central line.

Disadvantage:

1. This approach includes the lane central line not the left/right lines.
2. It will provide good results in the case of good road and good lane markers.
3. No results: if the road includes many noisy line segments, there will be no threshold existing.
4. Multiple results: if the road includes many noisy line segments, there will be more than one threshold existing.
5. Wrong results: if the road includes many noisy line segments, there will be threshold existing in wrong position.
6. Lots of processing overheads and processing time.

2.1.2. Seed-Line Algorithm:

1. The input here is the output from the block used for line segment detection (LSD or PPHT).
2. Divide the input into left region and right region, getting left/right lines.
3. Left/right lines are ordered according to their position from bottom to top.
4. For each region choose seed line to be the initial line, also to has a neighboring line.
5. For each region, loop over ordered lines, for each line calculate horizontal distance from seed line $\Delta \boldsymbol{x}$, and the angle $\boldsymbol{\theta}$ of the line. The line is added to the region if $\Delta \boldsymbol{x}$ is less than α1 (empirically chosen 40) and $\Delta \boldsymbol{\theta}$ is less than α2 (empirically chosen 15). The initial region angle $\boldsymbol{\theta\ region}$ is the angle of the seed line, and each time a line is added to the region, $\boldsymbol{\theta\ region}$ is updated to the average angle of all lines in the region.

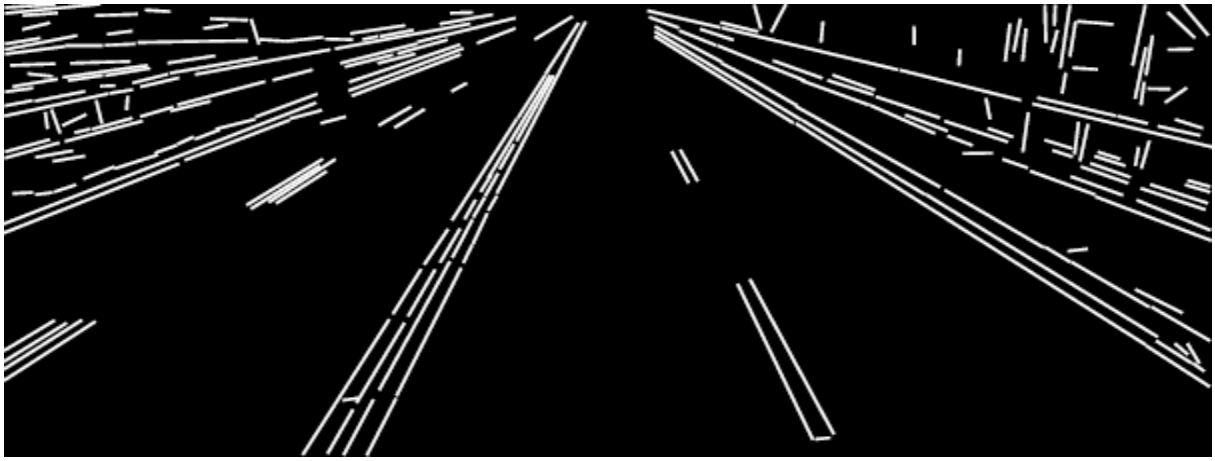


Figure 35:output image from our LSD block

Algorithm: Region Growing

Input: $lines_{filtered}$
Output: $Region_{Left}$, $Region_{Right}$

$lines_{left}$, $lines_{right}$ ← classify *line* according to its $\boldsymbol{x}$ value
$lines_{orderedLeft}$, $lines_{orderedRight}$ ← **order** ($lines_{left}$, $lines_{right}$, BOTTOM_TOP)
$seedLine_{left}$, $seedLine_{right}$ ← **findSeed** ($lines_{orderedLeft}$, $lines_{orderedRight}$)
$\boldsymbol{\theta}_{region}$ = angle of *seedLine*
foreach *line* **in** ($lines_{orderedLeft}$, $lines_{orderedRight}$) **do**
    $\Delta \boldsymbol{x}$ = horizontal distance between *line* and *seedLine*
    $\boldsymbol{\theta}$ = angle of the *line*
    $\Delta \boldsymbol{\theta} = |\ \boldsymbol{\theta}_{region} - \boldsymbol{\theta}|$
    if $\Delta \boldsymbol{x}$ <= α1 **and** $\Delta \boldsymbol{\theta}$ <= α2
        add *line* to *Region*
        seedLine= line
        $\boldsymbol{\theta}_{region}$ = average angle value of all the lines in *Region*
    end
end

Figure 36: ROI based on Seed-line Algorithm

Output:

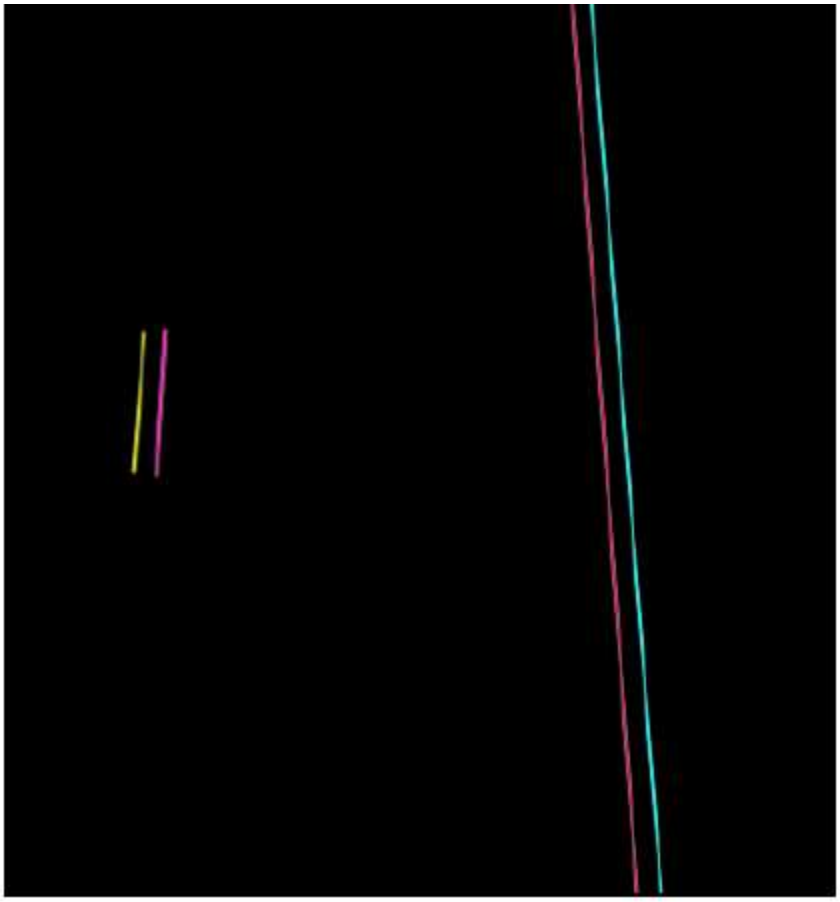


Figure 37:ROI detected based on Seed-line approach

Advantage:

1. More Accurate results than vanishing point algorithm.
2. This approach includes the left/right lines not only the lane central line.
3. It can be used for each vehicle regardless its height and the position of the camera.

Disadvantage: Lots of processing overheads and processing time.

2.1.3. Fixed ROI:

Previous researches defined the ROI based on vanishing points takes a lot of processing time and might determine an incorrect ROI if the vanishing points were incorrect. Our project is mainly focused on detecting the starting and ending positions of straight and curve lanes with the discrimination of dashed and solid lanes within the ROI. Here we will use a simple approach to apply ROI. We will manually apply suitable fixed ROI on the input image before using line segment detection (LSD) or Progressive Probabilistic Hough Transform (PPHT). Here, we will avoid too much processing on unnecessary information of the roads in the Non-road region.

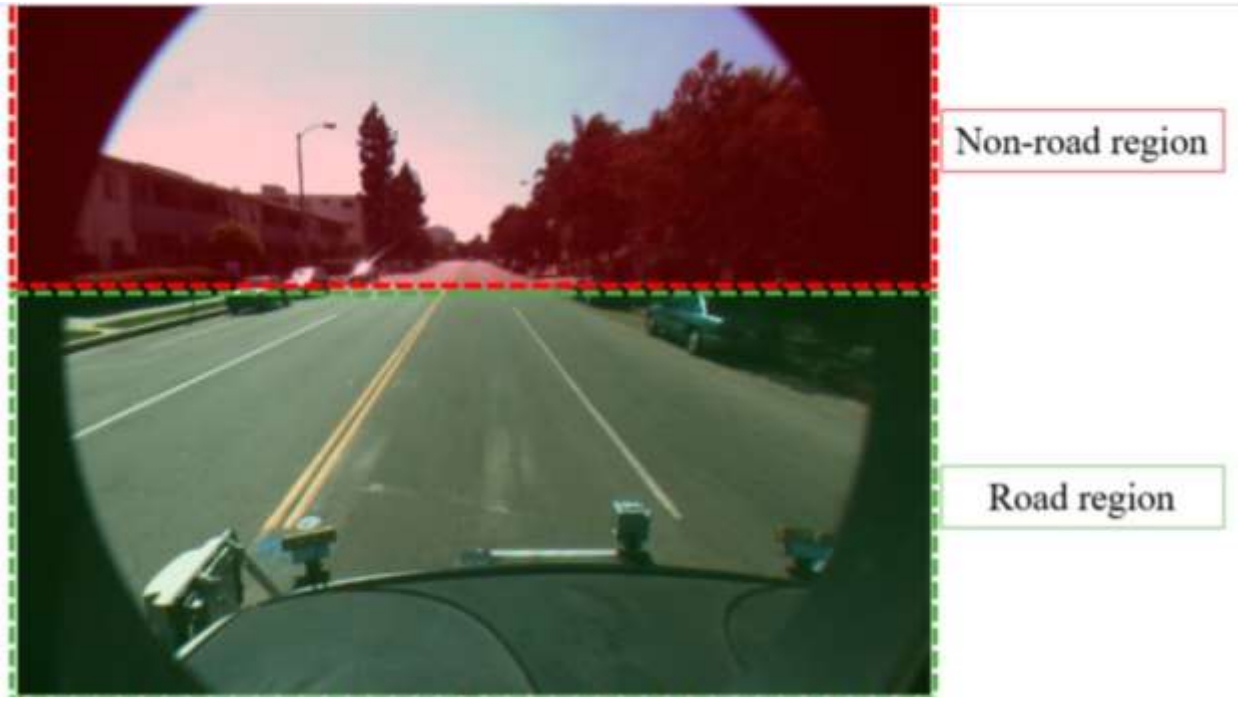


Figure 38:Camera input frame

Output:

Figure 39: (a) Input image of the camera (b) Apply fixed ROI on the input (c) The Output

Advantage:

1. Suitable approach for real time applications, here, we can apply the fixed ROI almost in no processing time.
2. This approach includes the left/right lines not only the lane central line.
3. Fixed ROI avoids processing overheads on unnecessary information in the road so we can decrease all processing time in the future blocks (LSD block, Filtration block, Curve fitting block, Tracking, Lane Departure warning). So, we can easily reach real-time.

Disadvantage:

1. We must manually change the position of the Fixed ROI to be suitable for each vehicle.
2. The position of fixed ROI may be changed if uphill and downhill appear in the road.

Conclusion :

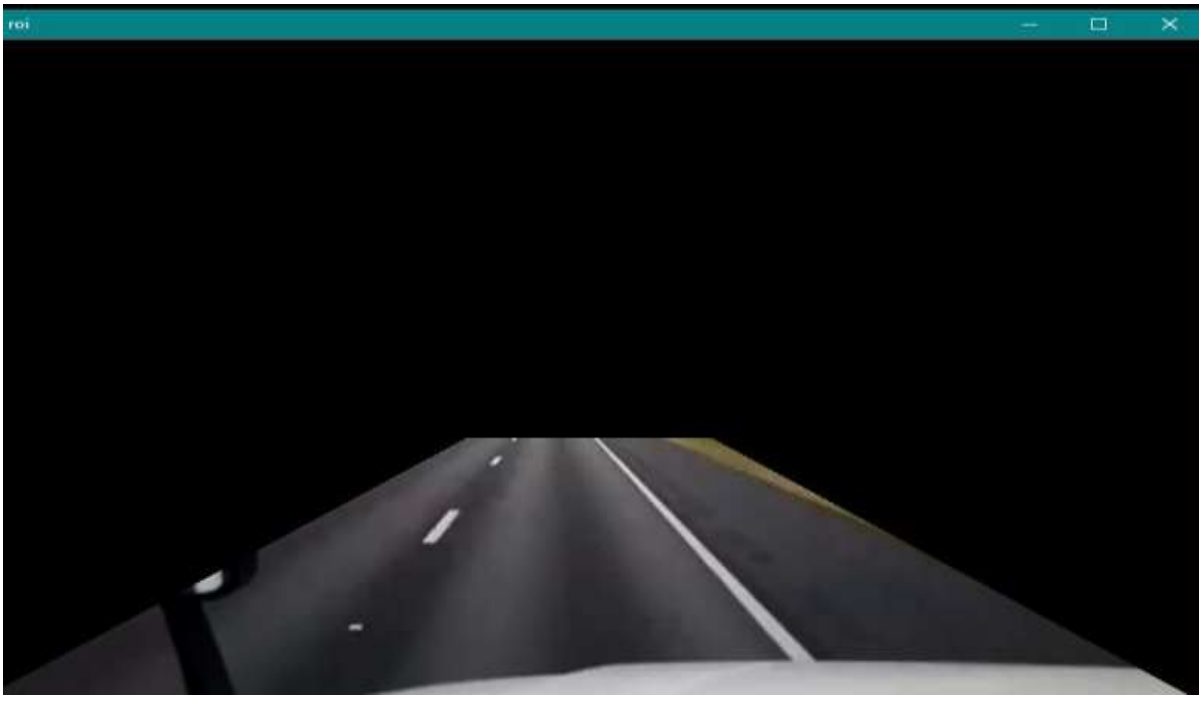
We will use the third approach (Fixed ROI) as it is very simple and suitable for our proposed solution. It's more suitable for real- time application.

ROI results

Figure 40: Fixed ROI dimensions

Figure 41: Input frame

Output of ROI block:



2.2. Line Segment Detection (LSD)

LSD is aimed at detecting locally straight contours on images. This is what we call line segments. Contours are zones of the image where the gray level is changing fast enough from dark to light or the opposite. Thus, the gradient and level-lines of the image are key concepts for LSD. The algorithm starts by computing the level-line angle at each pixel to produce a level-line i.e., a unit vector such that all vectors are tangent to the level line going through their base point. Then, this is segmented into connected regions of pixels that share the same level-line angle up to a certain tolerance . These connected regions are called line support regions.

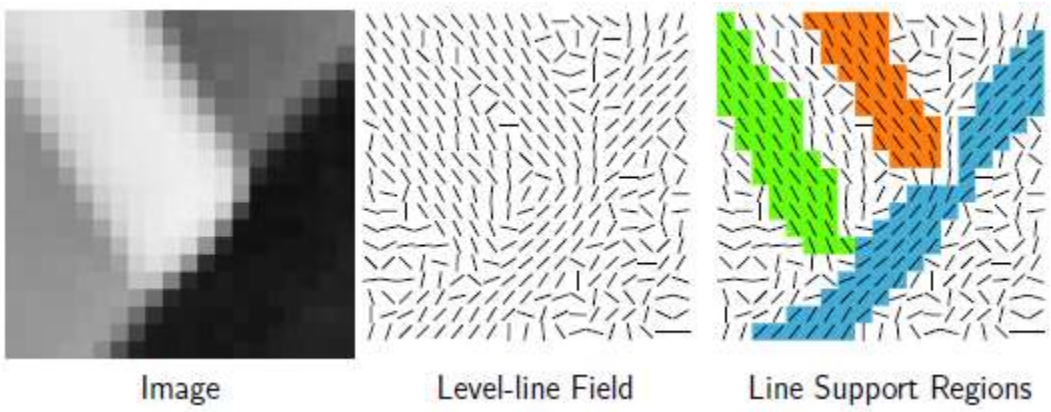


Figure 42 : LSD

Each line support region (a set of pixels) is a candidate for a line segment. The corresponding geometrical object (a rectangle in this case) must be associated with it.
Each rectangle is subject to a validation procedure. The pixels in the rectangle whose level-line angle corresponds to the angle of the rectangle up to a tolerance _ are called aligned points. The validation step is based on the a contrario approach and the Helmholtz principle. The so-called Helmholtz principle states that no perception (or detection) should be produced on an image of noise. Accordingly, the a contrario approach proposes to define a noise or a contrario model H0 where the desired structure is not present. Then, an event is validated if the expected number of events as good as the observed one is small on the a contrario model. In other words, structured events are defined as being rare in the a contrario model.

Another definition states that it is not possible to observe a regular structure in a noise.

Given an image i and a rectangle r, we will note k(r; i) the number of aligned points and n(r) the total number of pixels in r. Then, the expected number of events which are as good as the observed one is

$$N_{test} \cdot P_{H_0}[k(r, I) \geq k(r, i)]$$

where the number of tests Ntest is the total number of possible rectangles being considered, PH0 is the probability on the a contrario model H0 (that is defined below), and I is a random image

following H0. The H0 stochastic model fixes the distribution of the number of aligned points k(r; I), which only depends on the distribution of the level-line field associated with I. Thus H0 is a noise model for the image gradient orientation rather than a noise model for the image.

The a contrario model H0 used for line segment detection is therefore defined as a stochastic model of the level-line field satisfying the following properties:

The level lines of pixels are independent random variable and and the level line angles at pixels are uniformly distributed between 0 and 2pi so where LLA(j) is the level-line angle at pixel j. Under hypothesis H0, the probability that a pixel on
the a contrario model is an aligned point is

$$p = \frac{\tau}{\pi}$$

and, as a consequence of the independence of the random variables LLA(j), k(r; I) follows a binomial distribution. Thus, the probability term PH0 [k(r; I) > k(r; i)] is given by :

$$P_{H_0}\left[k(r,I) \geq k(r,i)\right] = B\left(n(r), k(r,i), p\right)$$

where B(n; k; p) is the tail of the binomial distribution:

$$B(n,k,p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

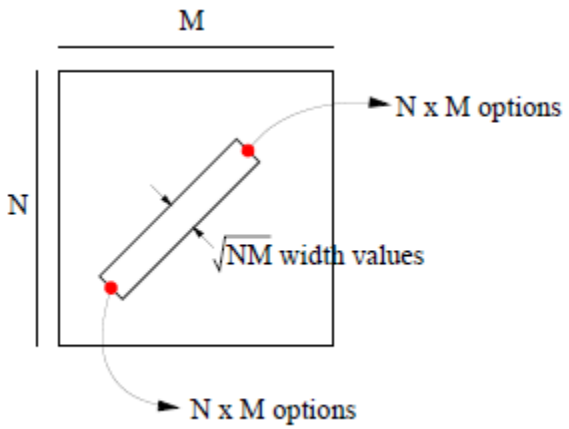


Figure 43: Determine number of tests

From the above figure the number of tests is given by (NM) power ($\frac{5}{2}$)

The precision p is initially set to the value; but other values are also tested to cover the relevant range of values. We will note the number different p values potentially tried. Each rectangle with each p value is a different test. Thus, the final number of tests

Is (NM) power (2.5) * γ

Finally, we define the Number of False Alarms (NFA) associated with a rectangle r on the image I as:

$$\mathrm{NFA}(r,i) = (NM)^{5/2}\gamma \cdot B\Big(n(r), k(r,i), p\Big).$$

This corresponds to the expected number of rectangles which have a sufficient number of aligned points to be as rare as r under H0. When the NFA associated with an image rectangle is large, this means that such an event is expected on the a contrario model, i.e., common and thus not a relevant one. On the other hand, when the NFA value is small, the event is rare and probably a meaningful one. A threshold " is selected and when a rectangle has NFA (r; i) < threshold it is called "-meaningful rectangle and produces a detection.

LSD Algorithm

Algorithm 1: LSD: Line Segment Detector

input: An image I.
output: A list *out* of rectangles.

$I_S \leftarrow \mathrm{ScaleImage}(I, S, \sigma = \frac{\Sigma}{S})$
$(\mathrm{LLA}, |\nabla I_S|, \mathrm{OrderedListPixels}) \leftarrow \mathrm{Gradient}(I_S)$
$\mathrm{Status} \leftarrow \begin{cases} \mathrm{USED}, & \text{pixels with } |\nabla I_S| \leq \rho \\ \mathrm{NOT\ USED}, & \text{otherwise} \end{cases}$
foreach *pixel* $P \in \mathrm{OrderedListPixels}$ **do**
  if $\mathrm{Status}(P) = \mathrm{NOT\ USED}$ **then**
    $region \leftarrow \mathrm{RegionGrow}(P, \tau)$
    $rect \leftarrow \mathrm{Rectangle}(region)$
    while $\mathrm{AlignedPixelDensity}(rect, \tau) < D$ **do**
      $region \leftarrow \mathrm{CutRegion}(region)$
      $rect \leftarrow \mathrm{Rectangle}(region)$
    end
    $rect \leftarrow \mathrm{ImproveRectangle}(rect)$
    $nfa \leftarrow \mathrm{NFA}(rect)$
    if $nfa \leq \varepsilon$ **then**
      Add $rect \rightarrow out$
    end
  end
end

Figure 44: LSD Algorithm

Image scaling

The result of LSD is different when the image is analyzed at different scales or if the algorithm is applied to a small part of the image. This is natural and corresponds to the different details that one can see if an image is observed from a distance or if attention is paid to a specific part. As a result of the a contrario validation step, the detection thresholds automatically adapt to the image size that is directly related to the number of tests. The scale of analysis is a choice left to the user, who can select it by cropping the image. Otherwise LSD processes automatically the entire image. The first step of LSD is, nevertheless, to scale the input image to 80% of its size. This scaling helps to cope with aliasing and quantization artifacts (especially the staircase effect) present in many images. Blurring the image would produce the same effect but affecting statistics of an image in the a contrario model: some structures would be detected on a blurred white noise. When correctly sub-sampled, the white noise statistics are preserved. Note that the a contrario validation is applied to the scaled image and the N _M image size used in the NFA computation corresponds to an input image of size 1:25N _ 1:25M.

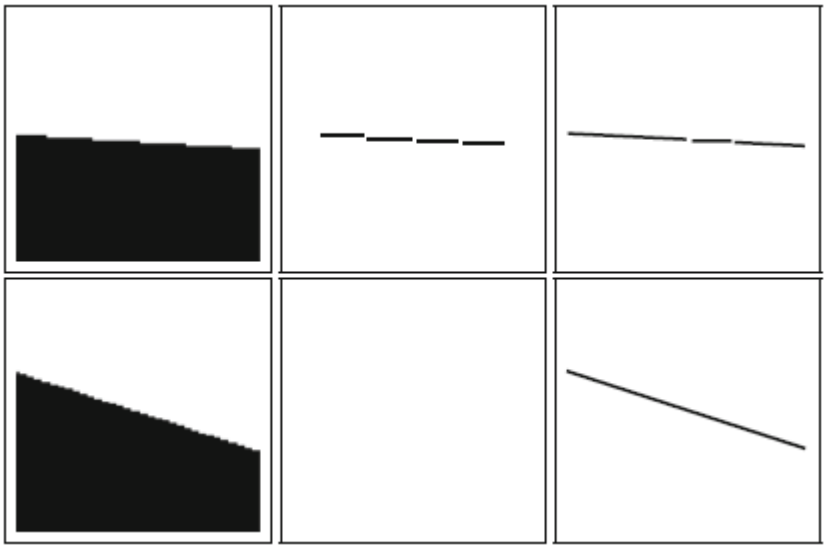


Figure 45: Staircase effect

Gradient computation

The image gradient is computed at each pixel using a 2x2 mask. Given:

$\ddots$	$\vdots$	$\vdots$	$\cdots$
$\cdots$	$i(x,y)$	$i(x+1,y)$	$\cdots$
$\cdots$	$i(x,y+1)$	$i(x+1,y+1)$	$\cdots$
$\cdots$	$\vdots$	$\vdots$	$\ddots$

where i(x; y) is the image gray level value at pixel (x; y), the image gradient is computed as

$$g_u(u,v) = \frac{x(u+1,v)+x(u+1,v+1)-x(u,v)-x(u,v+1)}{2},$$

$$g_v(u,v) = \frac{x(u,v+1)+x(u+1,v+1)-x(u,v)-x(u+1,v)}{2}.$$

And the LLA is computed as

$$\lambda(u,v) = \arctan\left(\frac{g_u(u,v)}{-g_v(u,v)}\right)$$

And the gradient magnitude is computed by

$$|\nabla x(u,v)| = \sqrt{g_u^2(u,v)+g_v^2(u,v)}.$$

Gradient Pseudo-Ordering

LSD is a greedy algorithm and the order in which pixels are processed has an impact on the result. Pixels with high gradient magnitude correspond to the more contrasted edges. In an edge, the central pixels usually have the highest gradient magnitude. hence it makes sense to start looking for line segments at pixels with the highest gradient magnitude. Sorting algorithms usually require O(n log n) operations to sort n values. However, a simple pixel pseudo-ordering is possible in linear-time. To this aim, 1024 bins are created corresponding to equal gradient magnitude intervals between zero and the largest observed value on the image. Pixels are classified into the bins according to their gradient magnitude. LSD uses first seed pixels from the bin of the largest gradient magnitudes; then it takes seed pixels from the second bin, and so on until exhaustion of all bins. 1024 bins are enough to sort almost strictly the gradient values when the gray level values are quantized in the integer range [0,255].

Gradient Threshold

Pixels with small gradient magnitude correspond to at zones or slow gradients. Also, they naturally present a higher error in the gradient computation due to the quantization of their values. In LSD the pixels with gradient magnitude smaller than p are therefore rejected and not used in the construction of line-support regions or rectangles.

Assuming a quantization noise n and an ideal image i we observe:

$$\tilde{i} = i + n \qquad \nabla\tilde{i} = \nabla i + \nabla n.$$

So the threshold is set up as

$$\rho = \frac{q}{\sin \tau}$$

Where the above law is driven from the following picture

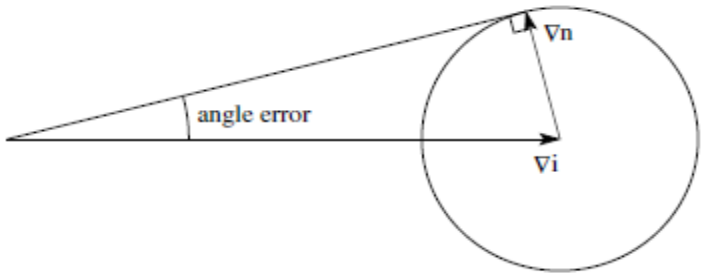


Figure 46: threshold calculation

Our implemented LSD Results:

Figure 47:LSD result for dashed and solid lanes (noon time)

Figure 48: LSD results for curved Lanes (noon time)

Figure 49: LSD results for night time

Figure 50: LSD before filteration

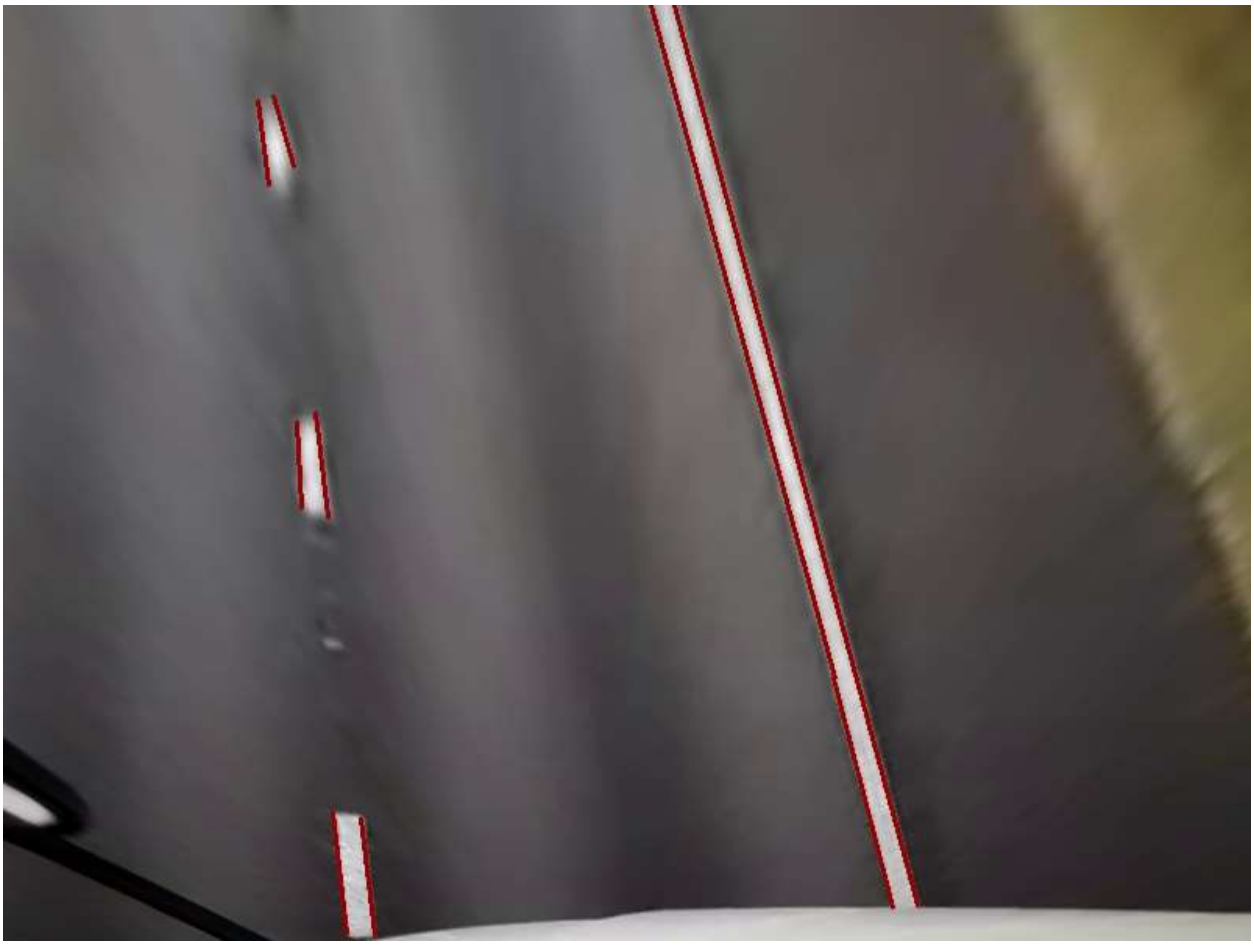


Figure 51: LSD output after filteratio

2.3. Region Growing

Algorithm 2: RegionGrow

input : A level-line field LLA, a seed pixel P, an angle tolerance τ, and a Status variable for each pixel.

output: A set of pixels: *region*.

```
1  Add P → region
2  θ_region ← LLA(P)
3  S_x ← cos(θ_region)
4  S_y ← sin(θ_region)
5  foreach pixel P ∈ region do
6  |  foreach pixel Q ∈ Neighborhood(P) and Status(Q) ≠ USED do
7  |  |  if AngleDiff(θ_region, LLA(Q)) < τ then
8  |  |  |  Add Q → region
9  |  |  |  Status(Q) ← USED.
10 |  |  |  S_x ← S_x + cos(LLA(Q))
11 |  |  |  S_y ← S_y + sin(LLA(Q))
12 |  |  |  θ_region ← arctan(S_y/S_x).
13 |  |  end
14 |  end
15 end
```

Figure 52: Region Growing

Starting from a pixel in the ordered list of unused pixels, the seed, a region growing algorithm is applied to form a line-support region. Recursively, the unused neighbors of the pixels already in the region are tested, and the ones whose level-line angle is equal to the region angle region up to tolerance _ are added to the region. The initial region angle theta region is the level-line angle of the seed point, and each time a pixel is added to the region the region angle value is updated to

$$\arctan\left(\frac{\sum_j \sin(\text{level-line-angle}_j)}{\sum_j \cos(\text{level-line-angle}_j)}\right)$$

Rectangular Approximation

A line segment corresponds to a geometrical event, a rectangle. Before evaluating a line-support region, the rectangle associated with it must be found. The region of pixels is interpreted as a solid object and the gradient magnitude of each pixel is used as the mass of that point. Then, the center of mass of the region is selected as the center of the rectangle and the main direction of the rectangle is set to the first inertia axis of the region. Finally, the width and length of the rectangles are set to the smallest values that make the rectangle to cover the full line-support region.

$$c_x = \frac{\sum_{j \in \text{Region}} G(j) \cdot x(j)}{\sum_{j \in \text{Region}} G(j)}$$

$$c_y = \frac{\sum_{j \in \text{Region}} G(j) \cdot y(j)}{\sum_{j \in \text{Region}} G(j)}$$

where G(j) is the gradient magnitude of pixel j, and the index j runs over the pixels in the region. The main rectangle's angle is set to the angle of the eigenvector associated with the smallest eigenvalue of the matrix

$$M = \begin{pmatrix} m^{xx} & m^{xy} \\ m^{xy} & m^{yy} \end{pmatrix}$$

$$m^{xx} = \frac{\sum_{j \in \text{Region}} G(j) \cdot (x(j) - c_x)^2}{\sum_{j \in \text{Region}} G(j)}$$

$$m^{yy} = \frac{\sum_{j \in \text{Region}} G(j) \cdot (y(j) - c_y)^2}{\sum_{j \in \text{Region}} G(j)}$$

$$m^{xy} = \frac{\sum_{j \in \text{Region}} G(j) \cdot (x(j) - c_x)(y(j) - c_y)}{\sum_{j \in \text{Region}} G(j)}.$$

NFA Computation

A key concept in the validation of a rectangle is that of p-aligned points, namely the pixels in the rectangle whose level-line angle is equal to the rectangle's main orientation, up to a tolerance ῑ. The precision p is initially set to the value taw/pi, but other values are also tested and a total of different values for p are tried. The total number of pixels in the rectangle is denoted by n and the number of p-aligned points is denoted by k (we drop r and i when they are implicit to simplify the notation). Then, the number of false alarms (NFA) associated with the rectangle r is

$$\text{NFA}(r) = (NM)^{5/2} \gamma \cdot \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

where N and M are the number of columns and rows of the image (after scaling), and B(n; k; p) is the binomial tail.

$$B(n,k,p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

The rectangles with NFA(r) < threshold are validated as detections.

Computational Complexity

Performing a Gaussian sub-sampling and computing the image gradient, both can be performed with a number of operations proportional to the number of pixels in the image. Then, pixels are pseudo-ordered by a classification into bins, operation that can be done in linear time. The computational time of the line-support region finding algorithm is proportional to the number of visited pixels, and this number is equal to the total number of pixels in the regions plus the border pixels of each one.

Thus, the number of visited pixels remains proportional to the total number of pixels of the image.

The rest of the processing can be divided into two kinds of tasks. The first kind, for example summing the region mass or counting aligned points, are proportional to the total number of pixels involved in all regions. The second kind, like computing inertia moments or computing the NFA value from the number of aligned points, are proportional to the number of regions. Both the total number of pixels involved and the number of regions are at most equal to the number of pixels. All in all, LSD has an execution time proportional to the number of pixels in the image.

2.4. Filtration Stage:

Filtration process is done on the output from the LSD step to eliminate the false line segments detected.
False line segments are those characterized by:

- Inclined line segments below or above certain threshold of angle theta, those line segments usually form a kind of noisy lines that may affect the efficiency of detecting lanes.
- Length of line segment is below a certain threshold, those line segments should be removed to maintain the longest lines which form the lane.
- Shadows on the lane from nearby vehicles or from the surroundings are detected as false line segments which shall be removed for better accuracy.

The filtration process is done through two major functions which are (*Eliminate false line segments – Eliminate shadows' Noise*)

1. Eliminate False Line Segments

For each line detected from the LSD stage, the inclination angle (theta) of each line is calculated. Then, the length of each line segment is calculated using the Euclidian Distance.
Line segments whose inclination angle is not in the range of threshold and length below the threshold value are eliminated. Line segments which pass these conditions are used through the next stages for lane detection.

Eliminate Shadows' Noise

Shadows produce false line segments, which cause faults and errors in detecting lanes, so they should be eliminated.

Shadows elimination is done by the following steps:

- Convert video frame from RGB format to HSV color space.
- Split the image into H(Hue),S(Saturation),and v (Value)
- Detect shadow areas from the Value and check for the position of every LSD Lines.

LSD Results:

Figure 53: LSD output before filteration

Figure 54: LSD after filteration

2.5. Curve Fitting

Overview

The goal of using curve fitting is to identify the coefficients of first/second order equation that 'fits' the data well, to get the curve equation.

The process of finding the equation of the curve of best fit, which may be most suitable for predicting the unknown values, is known as curve fitting. Therefore, curve fitting means an exact relationship between two variables by algebraic equations. There are following methods for fitting a curve.

I. Graphic method II.
II. Method of group averages III.
III. Method of moments IV.
IV. Principle of least square.

In this project, we use the Principle of least square. Because the graphical method has the drawback that the straight line drawn may not be unique but principle of least squares provides a unique set of values to the constants and hence suggests a curve of best fit to the given data. The method of least square is probably the most systematic procedure to fit a unique curve through the given data points.

Curve fitting and solution of equation

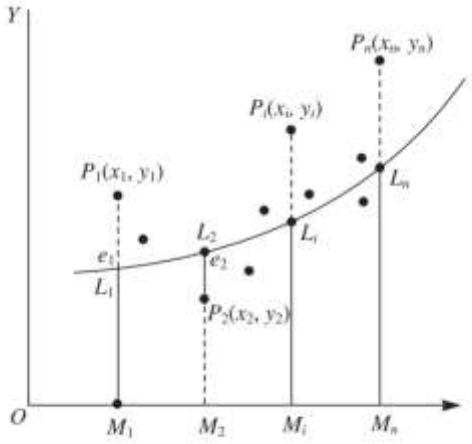


Figure 55: Curve fitting point

A mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. The sum of the squares of the offsets is used instead of the offset absolute values because this allows the residuals to be treated as a continuous differentiable quantity. The function's input are points detected from previous stage and the function's output are the coefficients of the curve/line equation based on number of input points. Generalizing from a straight-line equation: $y = a_0 + a_1 x + \cdots + a_k x^k$

The residual is given by:

$$R^2 = \sum_{i=1}^{n} (y_i - (a_0 + a_1 x + \cdots + a_k x^k))^2$$

y_i: is the term corresponding to the actual value, and $(a_0 + a_1 x + \cdots + a_k x^k)$ is the term corresponding to the expected value "the assumed curve".

Computing the partial derivatives with respect to $a_1\ a_2 \ldots a_k$ respectively, in order to minimize the difference between the real value and the expected value -finding the minimum squares' areas-

$$\frac{\partial(R^2)}{\partial a_0} = -2 \sum_{i=1}^{n} \left[y - \left(a_0 + a_1 x + \cdots + a_k x^k\right)\right] = 0$$

$$\frac{\partial(R^2)}{\partial a_1} = -2 \sum_{i=1}^{n} \left[y - \left(a_0 + a_1 x + \cdots + a_k x^k\right)\right] x = 0$$

$$\frac{\partial(R^2)}{\partial a_k} = -2 \sum_{i=1}^{n} \left[y - \left(a_0 + a_1 x + \cdots + a_k x^k\right)\right] x_k = 0$$

These equations lead to:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \cdots & \sum_{i=1}^{n} x_i^k \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \cdots & \sum_{i=1}^{n} x_i^{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^k & \sum_{i=1}^{n} x_i^{k+1} & \cdots & \sum_{i=1}^{n} x_i^{2k} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \vdots \\ \sum_{i=1}^{n} x_i^k y_i \end{bmatrix}$$

Solve for finding coefficients $a_1\ a_2\ \ldots a_k$.

Note that $\boldsymbol{k}$ is the order, and n is the number of input points.

Curve fitting Algorithm

Input: $Region_{Left}$, $Region_{Right}$
Output: $finalPts_{Left}$, $finalPts_{Right}$

foreach *line* in ($Region_{Left}$, $Region_{Right}$)
| $pts \leftarrow$ **convertFromLineToPoint** (line)
end
foreach *pt* in (pts_{Left}, pts_{Right})
if $pts.length() <= 6$
| $polynomialOrder = 2$ // first order
else
| $polynomialOrder = 3$ // second order
end
$coeffs \leftarrow$ **polynomialFit** (pts_{Left}, pts_{Right})
$Y_{new} \leftarrow$ **selectYPts** ()
$X_{new} \leftarrow$ **substitute** (Y_{new})
$finalPts_{Left}$, $finalPts_{Right} \leftarrow$ **makePts** (Y_{new} , X_{new})

Figure 56: Curve fitting algorithm

Results

Figure 57: Curve fitting block output

2.6. Lane Keeping assist

After efficiently detecting the lane line left and right boundaries, a lane keeping assist system can be introduced to warn driver from any unintentional lane drifts.
The system measures the absolute distance between the center of the lane and the left lane boundary and compares it with the absolute distance between the center of the lane and the right lane boundary. If any absolute distance is bigger than the other with some threshold the system warns the driver and an indication is showed on the screen guiding the driver into the center of the lane again.

System allows lane change if and only if the line boundary is dashed and not solid. The solid lines will have a red boundary line while the dashed lines will have a green boundary line to inform the driver of the available lane change direction.

Algorithm:

1. **IF** (center to left line > (center to right line + threshold))
2. **Then** (warn the driver and show guiding arrow pointing to the left till car is approximately at the center of the lane)
3. **Else IF** (center to right line > (center to left line + threshold))
4. **Then** (warn the driver and show guiding arrow pointing to the right till car is approximately at the center of the lane)

Possible cases:

Case1: The driver approximately drives around the center of lane.

Figure 58:Car is approximately in the center of lane

Case 2: The driver moves to the left lane away from the center of lane.

Figure 59: car moves to the Left

Case 3: The driver moves to the left lane away from the center of lane.

Figure 60:Driver moves to the right

2.7. Output Stream

This is the final stage of the algorithm where the visualization of the result happens. The visualization procedure is as the following:

- Lane's lines/curves are drawn clearly on the left side and right side.
- The region between the two lines/curves are marked and filled for better lane visualization with green color.
- In case of dashed line/curve, it's drawn with dark green color. And the solid line/curve is drawn with red color.
- In case of the solid line/curve, the car cannot depart the current lane and cross the side of this line/curve, but for the dashed line/curve, the car can depart.

A white arrow is drawn on the shaded area between the two lanes to indicate the direction of the movement, if the car is drifting to the right the arrow direction is to the left and vice versa

Figure 61:Output stream after Lane Detection

Algorithm Summary

Figure 62: Input frame

Figure 63: After ROI

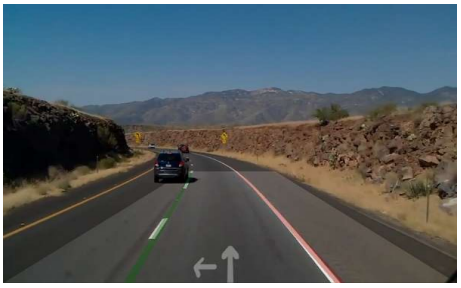


Figure 64: Lane Keeping assist

Figure 65: After ROI & IPM

Figure 67: Output stream

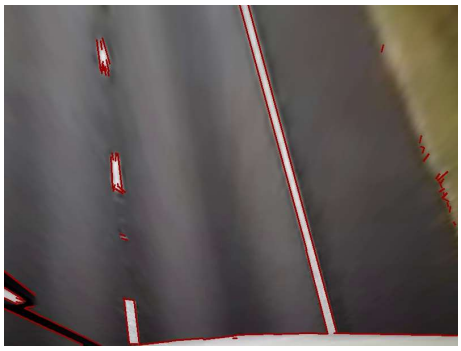


Figure 66: After LSD

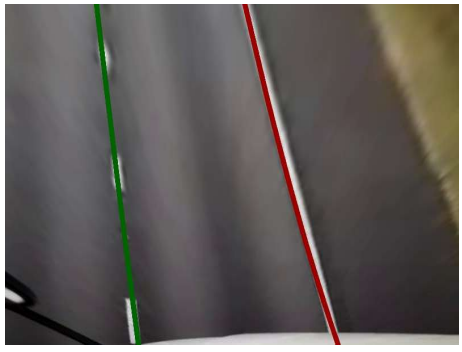


Figure 68: After Curve fitting

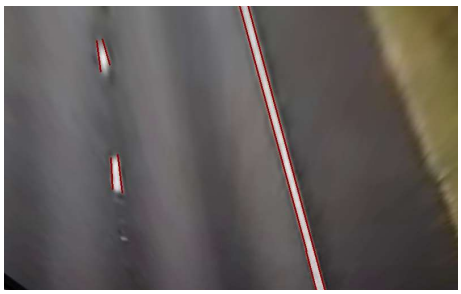


Figure 69: After filtration

Deep learning approach:

Deep learning techniques were used for our project and the core concepts of machine learning and deep learning used are explained below.

Core Concepts

Machine Learning

In machine learning we take some data ,then train a model on that data, and eventually use the trained model to make predictions on new data. The process of training a model can be seen as a learning process where the model is exposed to new, unfamiliar data step by step. At each step, the model makes predictions and gets feedback about how accurate its generated predictions were. This feedback, which is provided in terms of an error according to some measure (for example distance from the correct solution), is used to correct the errors made in prediction.

The learning process is often a game of back-and-forth in the parameter space: If you tweak a parameter of the model to get a prediction right, the model may have in such that it gets a previously correct prediction wrong. It may take many iterations to train a model with good predictive performance. This iterative predict-and-adjust process continues until the predictions of the model no longer improve.

Feature Engineering

Feature engineering is the art of extracting useful patterns from data that will make it easier for Machine learning models to distinguish between classes. For example, you might take the number of greenish vs. bluish pixels as an indicator of whether a land or water animal is in some picture. This feature is helpful for a machine learning model because it limits the number of classes that need to be considered for a good classification.

Feature engineering is the most important skill when you want to achieve good results for most predictions tasks. However, it is difficult to learn and master since different data sets and different kinds of data require different feature engineering approaches. Only crude guidelines exist, which makes feature engineering more of an art than a science. Features that are usable for one data set often are not usable for other data sets (for example the next image data set only contains land animals). The difficulty of feature engineering and the effort involved is the main reason to seek algorithms that can learn features; that is, algorithms that automatically engineer features.

While many tasks can be automated by Feature Learning (like object and speech recognition), feature engineering remains the single most effective techniques to perform difficult tasks.

Feature learning algorithms find the common patterns that are important to distinguish between classes and extract them automatically to be used in a classification or regression process. Feature learning can be thought of as Feature learning done automatically by algorithms. In deep learning, convolutional layers are exceptionally good at finding good features in images to the next layer to form a hierarchy of nonlinear features that grow in complexity (e.g. blobs, edges -> noses, eyes, cheeks -> faces). The final layer(s) use all these generated features for classification or regression (the last layer in a convolutional net is, essentially, multinomial logistic regressions).

Deep Learning

In hierarchical Feature Learning , we extract multiple layers of non-linear features and pass them to a classifier that combines all the features to make predictions. We are interested in stacking such very deep hierarchies of non-linear features because we cannot learn complex features from a few layers. It can be shown mathematically that for images the best features for a single layer are edges and blobs because they contain the most information that we can extract from a single non-linear transformation. To generate features that contain more information we cannot operate on the inputs directly, but we need to transform our first features (edges and blobs) again to get more complex features that contain more information to distinguish between classes.

It has been shown that the human brain does exactly the same thing: The first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges and blobs while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.

While hierarchical feature learning was used before the field deep learning existed, these architectures suffered from major problems such as the vanishing gradient problem where the gradients became too small to provide a learning signal for very deep layers, thus making these architectures perform poorly when compared to shallow learning algorithms (such as support vector machines).

The term deep learning originated from new methods and strategies designed to generate these deep hierarchies of non-linear features by overcoming the problems with vanishing gradients so that we can train architectures with dozens of layers of non-linear hierarchical features. In the early 2010s, it was shown that combining GPUs with activation functions that offered better gradient flow was sufficient to train deep architectures without major difficulties. From here the interest in deep learning grew steadily.

Logistic Regression

Regression analysis estimates the relationship between statistical input variables in order to predict an outcome variable. Logistic regression is a regression model that uses input variables to predict a categorical outcome variable that can take on one of a limited set of class values, for example “cancer” / “no cancer”, or an image category such as “bird” / “car” / “dog” / “cat” / “horse”.

Logistic regression applies the logistic sigmoid function (see Figure 1) to weighted input values to generate a prediction of which of two classes the input data belongs to (or in case of multinomial logistic regression, which of multiple classes).

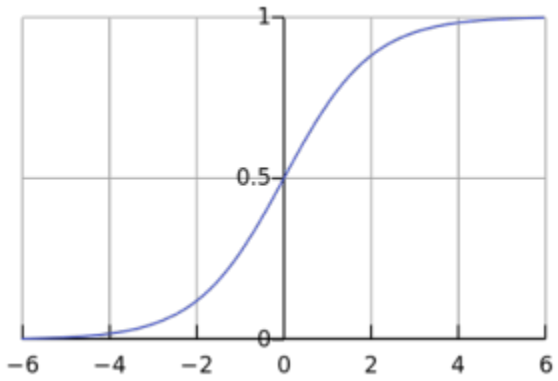


Figure 70: The logistic sigmoid function

Logistic regression is similar to a non-linear perceptron or a neural network without hidden layers. The main difference from other basic models is that logistic regression is easy to interpret and reliable if some statistical properties for the input variables hold. If these statistical properties hold one can produce a very reliable model with very little input data. This makes logistic regression valuable for areas where data are scarce, like the medical and social sciences where logistic regression is used to analyze and interpret results from experiments. Because it is simple and fast it is also used for very large data sets.

In deep learning, the final layer of a neural network used for classification can often be interpreted as a logistic regression. In this context, one can see a deep learning algorithm as multiple feature learning stages, which then pass their features into a logistic regression that classifies an input.

Artificial Neural Network

An artificial neural network is a structure which takes some input data and transforms this input data by calculating a weighted sum over the inputs and applies a non-linear function to this transformation to calculate an intermediate state. The three steps above constitute what is known as a layer, and the transformative function is often referred to as a unit. The intermediate states—often termed features are used as the input into another layer.

Through repetition of these steps, the artificial neural network learns multiple layers of non-linear features, which it then combines in a final layer to create a prediction.

The neural network learns by generating an error signal that measures the difference between the predictions of the network and the desired values and then using this error signal to change the weights (or parameters) so that predictions get more accurate.

Artificial Neuron

The term artificial neuron—or most often just neuron—is an equivalent term to unit, but implies a close connection to neurobiology and the human brain while deep learning has very little to do with the brain (for example, it is now thought that biological neurons are more similar to entire multilayer perceptron's rather than a single unit in a neural network). The term neuron was encouraged after the last AI winter to differentiate the more successful neural network from the failing and abandoned perceptron. However, since the wild successes of deep learning after 2012, the media often picked up on the term "neuron" and sought to explain deep learning as mimicry of the human brain, which is very misleading and potentially dangerous for the perception of the field of deep learning. Now the term neuron is discouraged and the more descriptive term unit should be used instead.

Activation Function

An activation function takes in weighted data (matrix multiplication between input data and weights) and outputs a non-linear transformation of the data. For example, the rectified linear activation function (RELU) (essentially set all negative values to zero) which takes the maximum of the input or 0 and its main advantage is it makes the training process fast . The difference between units and activation functions is that units can be more complex, that is, a unit can have multiple activation functions (for example LSTM units) or a slightly more complex structure

The difference between linear and non-linear activation functions can be shown with the relationship of some weighted values: Imagine the four points A1, A2, B1 and B2. The pairs A1 / A2, and B1 / B2 lie close to each other, but A1 is distant from B1 and B2, and vice versa; the same for A2.

With a linear transformation the relationship between pairs might change. For example, A1 and A2 might be far apart, but this implies that B1 and B2 are also far apart. The distance between the pairs might shrink, but if it does, then both B1 and B2 will be close to A1 and A2 at the same time. We can apply many linear transformations, but the relationship between A1 / A2 and B1 / B2 will always be similar.

In contrast, with a non-linear activation function we can increase the distance between A1 and A2 while we decrease the distance between B1 and B2. We can make B1 close to A1, but B2 distant from A1. By applying non-linear functions, we create new relationships between the points. With every new non-linear transformation, we can increase the complexity of the relationships. In deep learning, using non-linear activation functions creates increasingly complex features with every layer.

In contrast, the features of 1000 layers of pure linear transformations can be reproduced by a single layer (because a chain of matrix multiplication can always be represented by a single matrix multiplication). This is why non-linear activation functions are so important in deep learning.

Layer

A layer is the highest-level building block in deep learning. A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer. A layer is usually uniform, that is it only contains one type of activation function, pooling, convolution etc. so that it can be easily compared to other parts of the network. The first and last layers in a network are called input and output layers, respectively, and all layers in between are called hidden layers.

Convolutional neural networks

Convolutional networks, also known as convolutional neural networks or CNNs, are a specialized kind of neural network for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. In this chapter, we will first describe what convolution is. Next, we will explain the motivation behind using convolution in a neural network. We will then describe an operation called pooling, which almost all convolutional networks employ. Usually, the operation used in a convolutional neural network does not correspond precisely to the definition of convolution as used in other fields such as engineering or pure mathematics. We will describe several variants on the convolution function that are widely used in practice for neural networks. We will also show how convolution may be applied to many kinds of data, with different numbers of dimensions.

We then discuss means of making convolution more efficient. Convolutional networks stand out as an example of neuroscientific principles influencing deep learning.

The Convolution Operation

In its most general form, convolution is an operation on two functions of a real- valued argument. To motivate the definition of convolution, we start with examples of two functions we might use. Suppose we are tracking the location of a spaceship with a laser sensor. Our laser sensor provides a single output x(t), the position of the spaceship at time t. Both x and t are real-valued, i.e., we can get a different reading from the laser sensor at any instant in time. Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function w (a), where a is the age of a measurement. If we apply such a weighted average operation at every moment, we obtain a new function s providing a smoothed estimate of the position of the spaceship:

$$s(t) = \int x(a)w(t-a)da$$

This operation is called convolution. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t)$$

In our example, w needs to be a valid probability density function, or the output is not a weighted average. Also, w needs to be 0 for all negative arguments, or it will look into the future, which is presumably beyond our capabilities. These limitations are particular to our example though. In general, convolution is defined for any functions for which the above integral is defined, and may be used for other purposes besides taking weighted averages.

In convolutional network terminology, the first argument (in this example, the function x) to the convolution is often referred to as the input and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the feature map . In our example, the idea of a laser sensor that can provide measurements at every instant in time is not realistic. Usually, when we work with data on a computer, time will be discretized, and our sensor will provide data at regular intervals. In our example, it might be more realistic to assume that our laser provides a measurement once per second. The time index t can then take on only integer values. If we now assume that x and w are defined only on integer t.

we can define the discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. We will refer to these multidimensional arrays as tensors. Because each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but the finite set of points for which we store the values. This means that in practice we can implement the infinite summation as a summation over a finite number of array elements.

Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Convolution is commutative. The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases. The only reason to flip the kernel is to obtain the commutative property. While the commutative property is useful for writing proofs, it is not usually an important property of a neural network implementation. Instead, many neural network libraries implement a related function called the **cross-correlation**, which is the same as convolution but without flipping the kernel. Many machine learning libraries implement cross-correlation but call it convolution.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

Discrete convolution can be viewed as multiplication by a matrix. However, the matrix has several entries constrained to be equal to other entries. For example, for univariate discrete convolution, each row of the matrix is constrained to be equal to the row above shifted by one element.

This is known as a **Toeplitz matrix**. In two dimensions, a **doubly block circulant matrix** corresponds to convolution. In addition to these constraints that several elements be equal to each other, convolution usually corresponds to a very sparse matrix (a matrix whose entries are mostly equal to zero). This is because the kernel is usually much smaller than the input image. Any neural network algorithm that works with matrix multiplication and does not depend on specific properties of the matrix structure should work with convolution, without requiring any further changes to the neural network. Typical convolutional neural networks do make use of further specializations in order to deal with large inputs efficiently, but these are not strictly necessary from a theoretical perspective.

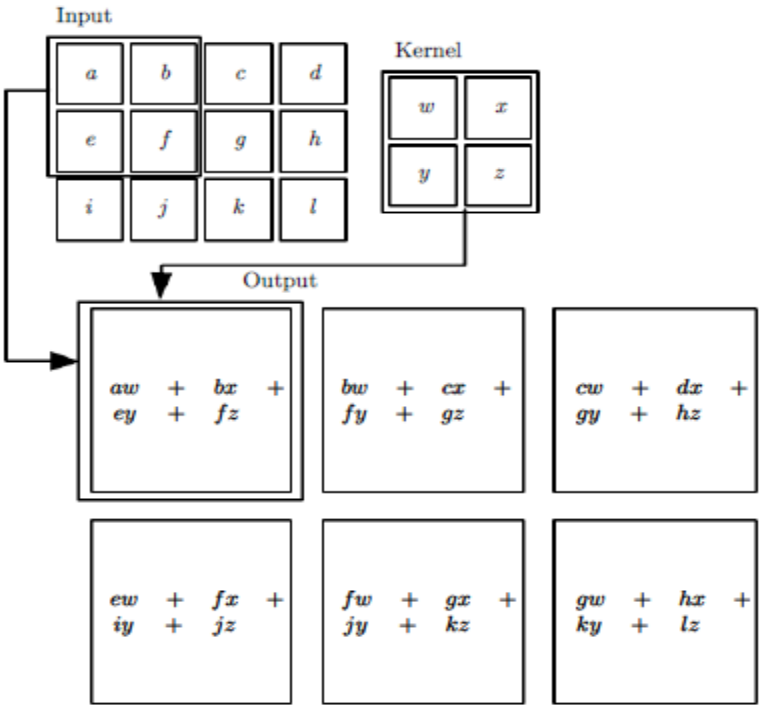


Figure 71: convolution of input with kernel without the need to kernel flipping.

Motivation

Convolution leverages three important ideas that can help improve a machine learning system: **sparse interactions**, **parameter sharing** and **equivariant representations**. Moreover, convolution provides a means for working with inputs of variable size. We now describe each of these ideas in turn.

Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit. This means every output unit interacts with every input unit. Convolutional networks, however,

typically have **sparse interactions** (also referred to as **sparse connectivity** or **sparse weights**). This is accomplished by making the kernel smaller than the input. For example, when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations. These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires m×n parameters and the algorithms used in practice have O(m × n) runtime (per example). If we limit the number of connections each output may have to k, then the sparsely connected approach requires only k × n parameters and O(k × n) runtime. For many practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m.

First , Parameter sharing refers to using the same parameter for more than one function in a model. In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer. It is multiplied by one element of the input and then never revisited. As a synonym for parameter sharing, one can say that a network has **tied weights**, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a convolutional neural net, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary). The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set. This does not affect the runtime of forward propagation—it is still O(k × n)—but it does further reduce the storage requirements of the model to k parameters. Recall that k is usually several orders of magnitude less than m. Since m and n are usually roughly the same size, k is practically insignificant compared to m×n. Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

In the case of convolution, the particular form of parameter sharing causes the layer to have a property called **equivariance** to translation. To say a function is equivariant means that if the input changes, the output changes in the same way.

Specifically, a function f (x) is equivariant to a function g if f(g(x)) = g(f (x)).

In the case of convolution, if we let g be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to g. For example, let I be a function giving image brightness at integer coordinates. Let g be a function mapping one image function to another image function, such that I = g(I) is the image function with I(x, y) = I(x − 1, y).

This shifts every pixel of I one unit to the right. If we apply this transformation to I, then apply convolution, the result will be the same as if we applied convolution to I, then applied the transformation g to the output. When processing time series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time.

Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.

In some cases, we may not wish to share parameters across the entire image. For example, if we are processing images that are cropped to be centered on an individual's face, we probably want to extract different features at different locations—the part of the network processing the top of the face needs to look for eyebrows, while the part of the network processing the bottom of the face needs to look for a chin.

Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

Pooling

In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the **detector** stage. In the third stage, we use a **pooling function** to modify the output of the layer further.

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. For example, the **max pooling** operation reports the maximum output within a rectangular neighborhood. Other popular pooling functions include the average of a rectangular neighborhood, the L2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel.

In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.

For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face. In other contexts, it is more important to preserve the location of a feature. For example, if we want to find a corner defined by two edges meeting at a specific orientation, we need to preserve the location of the edges well enough to test whether they meet.

The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network.

Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to.

Proposed deep learning approaches for lane detection

Convolutional neural networks (CNNs) are usually built by stacking convolutional operations layer-by-layer. Although CNN has shown strong capability to extract semantics from raw pixels, its capacity to capture spatial relationships of pixels across rows and columns of an image is not fully explored. These relationships are important to learn semantic objects with strong shape priors but weak appearance coherences, such as traffic lanes.

In this approach, we propose Spatial CNN (SCNN), which generalizes traditional deep layer-by-layer convolutions to slice-by-slice convolutions within feature maps, thus enabling message passings between pixels across rows and columns in a layer. Such SCNN is particular suitable for long continuous shape structure or large objects, with strong spatial relationship but less appearance clues, such as traffic lanes, poles, and Wall

Traditional methods to model spatial relationship are based on Markov Random Fields (MRF) or Conditional Random Fields (CRF).

The used model in our project consists of five convolutional layers in the convolution stage with two pooling layers embedded in this stage for enhancing the training time while also keeping the most important details of features and dropout regularization was used in the layers with large connections to prevent overfitting to the training data and this model is followed by the stage of deconvolution which learns the weights for returning the picture back after classification instead of just extracting parameters from the network and then using computer vision for stacking the photo together and this saves time but come on the cost of the performance.

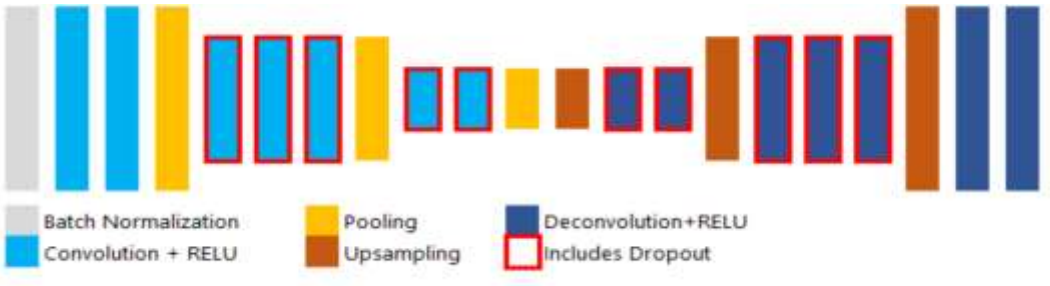


Figure 72: Deep Learning model

Training data and test data are first normalized by dividing them to 255 on each channel of colour or by dividing the only channel of picture if the training or testing is performed on only a single channel of the photos and stacking the results with the remaining channels.

Then in the convolutional layers , they are convoluted with kernels of learnt parameters and and then the output is upsampled to inverse the effect of pooling layer and deconvoluted to get the image from the filter outputs.

Labeling data

Data was labeled with both cases in the green channel and in all channels by drawing the lane region of road in it and using it to determine the road for the driver and samples of the labeled data are shown below. Data were chosen from various road cases like curved lanes and straight lanes to make the network able of detecting any type of lane.

Figure 73: sample1 of labeled data

Figure 74: sample2 of labeled data

Results below obtained from variety of lanes in Egypt (above) and china in different weather conditions.

Model can be improved by using more data for training and by adding more layers and of course following this with regularization and data augmentation in order to prevent overfitting and it can also be improved by increasing the variations in training data to be able to work robustly under various circumstances.

Deep Learning Results:

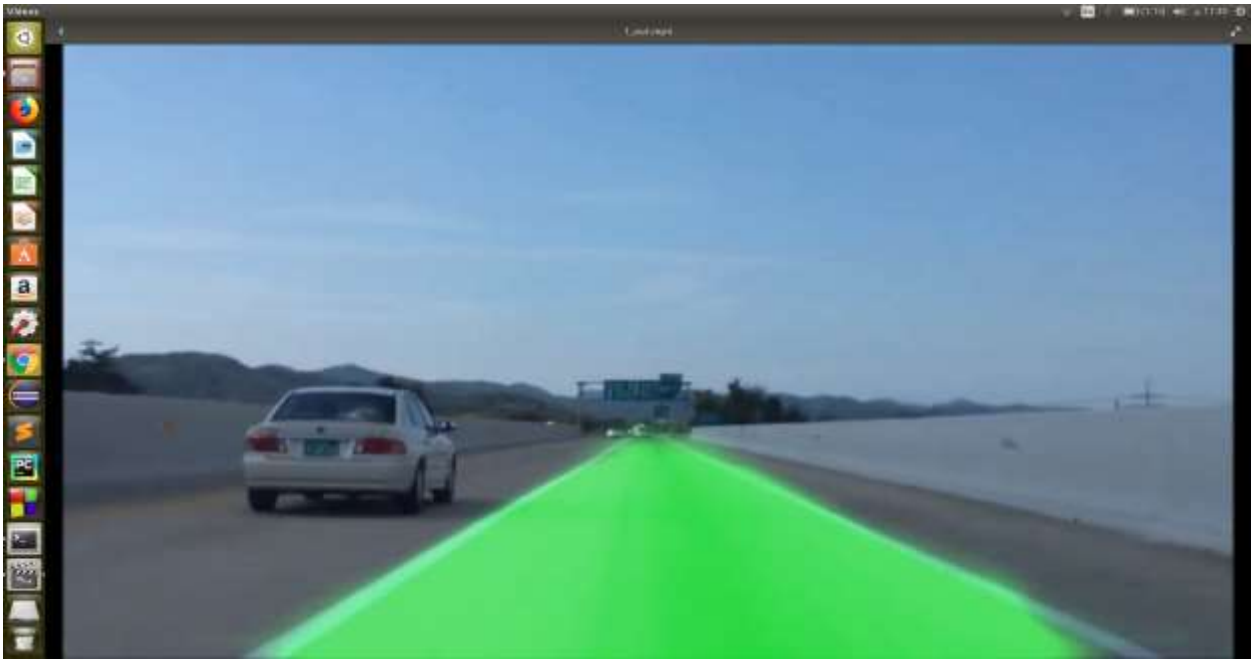


Figure 75: Sample result

Figure 76: sample result from Egypt - NA road

Performance on PC (Proposed computer vision approach):

CPU: i7 2.4 GHz - RAM: 6 GB - OS: Ubuntu Linux

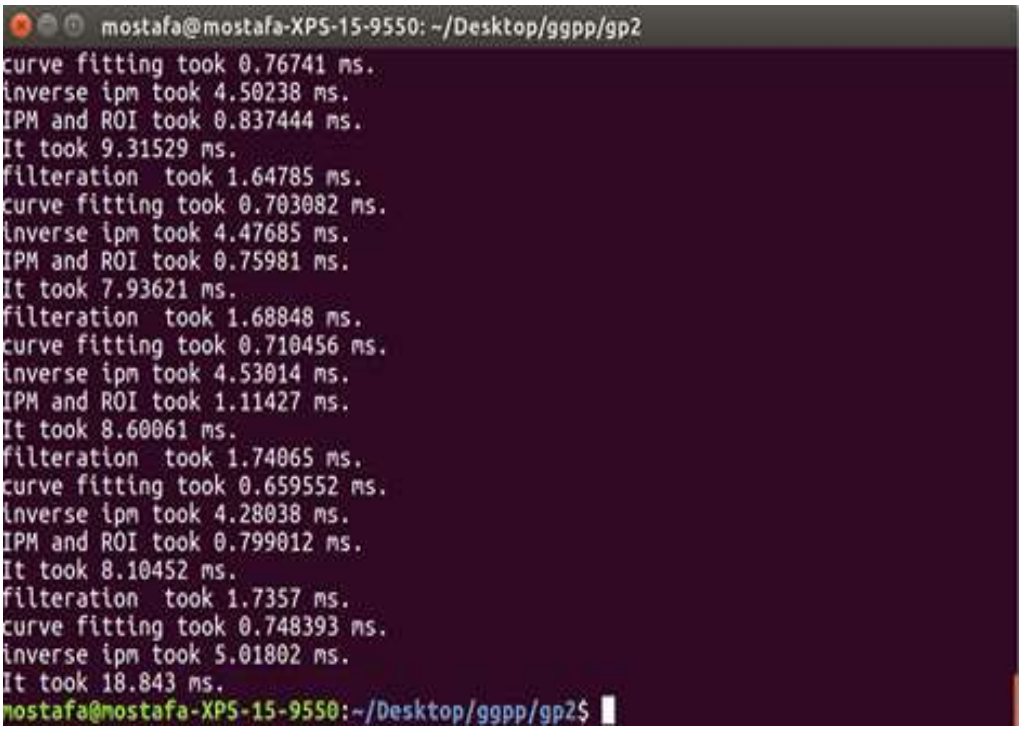


Figure 77: Performance on PC

Block	Timing performance
IPM & ROI	0. 799502 ms
LSD	7.8171 ms
Filtration	1.35336 ms
Curve fitting	0.660428 ms
Inverse IPM	3.346 ms

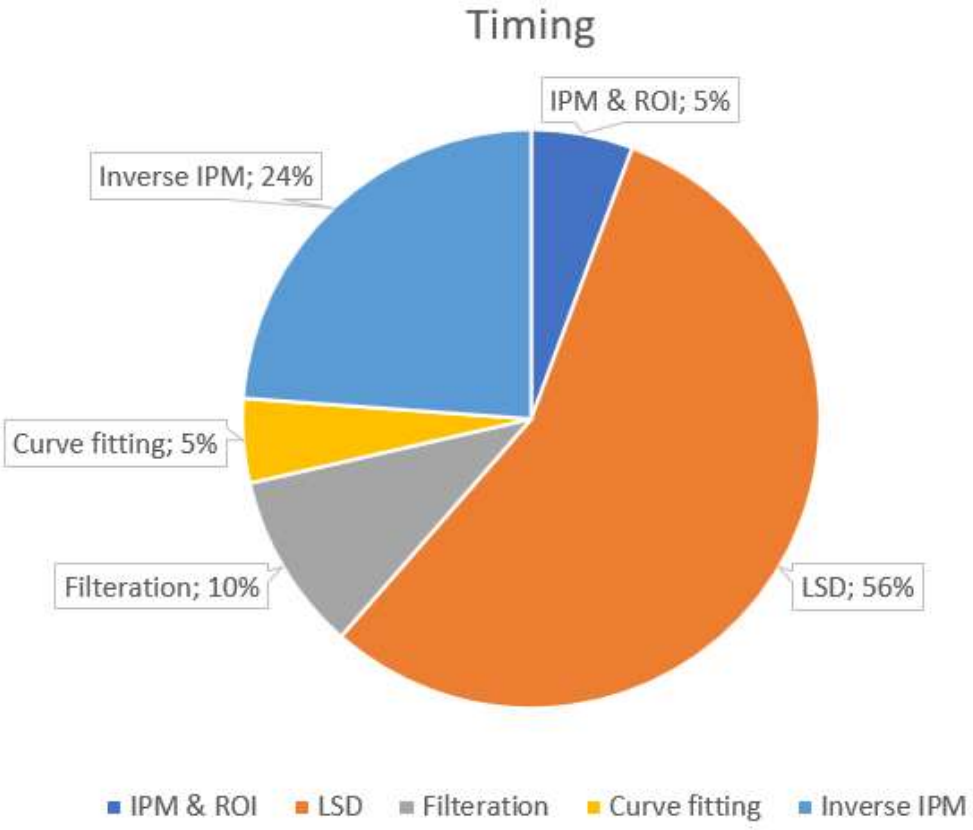


Figure 78: PC performance

The frame took 13.95 ms on PC. So, the performance arrives to 71 frame per second.

Chapter 3: Hardware Layer:

1. Nvidia Jetson TX1

Figure 79:Nvidia Jetson TX1

Overview:

Jetson TX2 is the fastest, most power-efficient embedded AI computing device. The latest addition to the industry-leading Jetson embedded platform, this 7.5-watt supercomputer on a module brings true AI computing at the edge. It's built around an NVIDIA Pascal™-family GPU and loaded with 8 GB of memory and 59.7 GB/s of memory bandwidth. It features a variety of standard hardware interfaces that make it easy to integrate it into a wide range of products and form factors.

Main advantages:

1. Using CUDA programming, has 256 CUDA cores.
2. Using Embedded Linux Ubuntu 16.04 LTS aarch64.
3. Available technical support and large community
4. Support OpenCV, Tensor Flow so it can run computer vision and AI approaches.
5. Up to 60 GB memory.
6. Easy to compile and Edit your code and easy to run.
7. All hardware, Software needed are available.
8. Support different programming languages: C – C++ - Python.

Hardware components:

	JETSON TX1
GPU	1 TFLOP/s 256-core Maxwell
CPU	64-bit ARM A57 CPUs
Memory	4 GB LPDDR4 | 25.6 GB/s
Storage	16 GB eMMC
Wifi/BT	802.11 2x2 ac/BT Ready
Networking	1 Gigabit Ethernet
Size	50mm x 87mm
Interface	400 pin board-to-board connector

Figure 80: Jetson TX1 H.W. components

Software Components:

NVIDIA JetPack 3.0 – Software Components	
Linux kernel 4.4	Ubuntu 16.04 LTS aarch64
CUDA Toolkit 8.0.56	cuDNN 5.1.10
TensorRT 1.0 GA	GStreamer 1.8.2
VisionWorks 1.6	OpenCV4Tegra 2.4.13-17
Tegra System Profiler 3.7	Tegra Graphics Debugger 2.3
Tegra Multimedia API	V4L2 Video Codec Interfaces

Figure 81: Jetson TX1 H.W. components

Jetson TX2 Camera specs:

Resolutions	Uncompressed YUV422
VGA	120 fps
HD (720p)	80 fps
Full HD (1080p)	80 fps
4K Ultra HD (3840x2160)	30 fps
4K Cinema (4096 x 2160)	30 fps
13MP (4192 x 3120)	20 fps

Sensor	:	1/3.2" Optical form factor AR1335 CMOS Image sensor
Focus Type	:	Fixed focus
Resolution	:	13.0 MP
Pixel size	:	1.1 µm x1.1 µm
Sensor Active Area	:	4208(H) x 3120(V)
Responsivity	:	4700 e/lux-sec
SNR	:	37 dB
Dynamic Range	:	69 dB
Output Format	:	Uncompressed UYVY streaming & Compressed MJPEG streaming
Shutter type	:	Electronic Rolling Shutter
Holder	:	S-Mount
ISP	:	External ISP
Array Size	:	4208x3120
DFOV	:	13M - 67°, 4K/1080p/720p - 53°, VGA - 67°(with the lens provided by e-con)

Main advantage of Nvidia Jetson TX1: CUDA programming:

In CUDA programming, we can use the resources of the GPU cores in the processing beside the CPU resources. So, we can increase the speed of executing from 10 to 50 times.

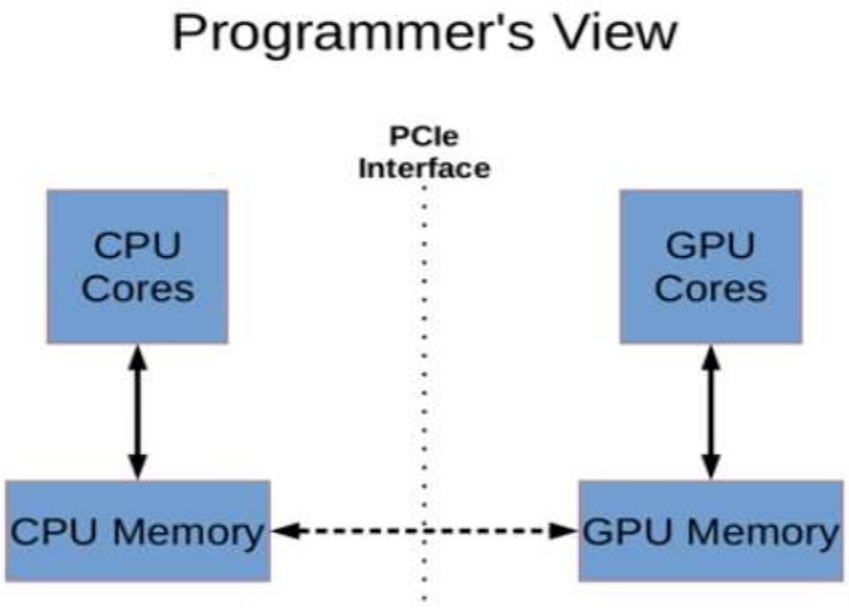


Figure 82:CUDA overview

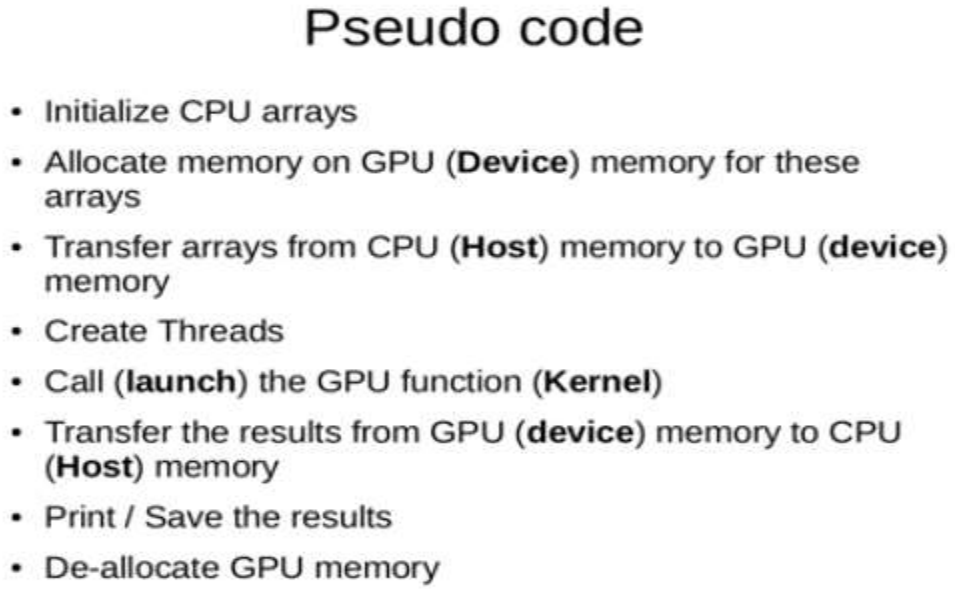


Figure 83:CUDA pesudo code

2. TI TDA3x

Figure 84:TDA3x

Overview:

TI's TDA3x System-on-Chip (SoC) is a highly optimized and scalable family of devices designed to meet the requirements of leading Advanced Driver Assistance Systems (ADAS). TDA3x SoC processors enable broad ADAS applications by integrating an optimal mix of performance, low power, smaller form factor and ADAS vision analytics processing that aims to facilitate a more autonomous and collision-free driving experience. The TDA3x SoC enables sophisticated embedded vision technology in today's automobile by enabling the industry's broadest range of ADAS applications including front camera, rear camera, surround view, radar, and fusion on a single architecture.

Features:

- Texas Instruments TDA3x SoC Processor
- FPD-Link III video inputs (4)
- HDMI video display input output
- Ethernet, CAN bus, and serial connectivity

Applications

- Mono or Stereo Front Camera
- Night Vision System
- LVDS Surround View
- Rear View Video Display
- Blind Spot Detection
- Pedestrian Detection
- Sensor Fusion
- Traffic Sign Recognition
- Remote Applications and Monitoring
- Lane Departure Warning

Specifications:

- heterogeneous & scalable architecture.
- fixed and floating-point dual-TMS320C66x generation of DSP cores.
- fully programmable Vision acceleration Pac (EVE).
- dual ARM®Cortex®-M4 cores.
- Image signal processor (ISP).
- displays, CAN and multi camera interfaces.

SPECIFICATIONS

SoC	TI TDA3x
DDR	1GB + ECC
QSPI	512Mbit
FRAM (EEPROM)	512Kbit
Ethernet Gbit (DP83867)	1 standard RJ-45 port
Video/Data Input Ports	HDMI, CSI2 ADAS hub (4 ports)
Display Output	1 HDMI
CAN	1 port
UART (USB to UART bridge)	1 console, 1 expansion
uSD Card	1 SDIO
Buffered GPIO	2 in and 2 out
Component Temperature	0C to 70C commercial
Power (Input)	Standard 6V to 12VDC, barrel jack connection
Power (FPD-Link)	5V or Vbatt (jumper selectable)
JTAG	60 pin and 14 pin (adapter not included)
BSP	D3 software framework/TI BIOS Vision SDK
Expansion	I2C, UART, SPI i 10th inch header
Access Panel	14 pin JTAG, UART, uSD card, HDMI
Lab Power Supply	12VDC, 2A
Accessory	Micro SD card

Figure 85:TDA3x specifications

Hardware components:

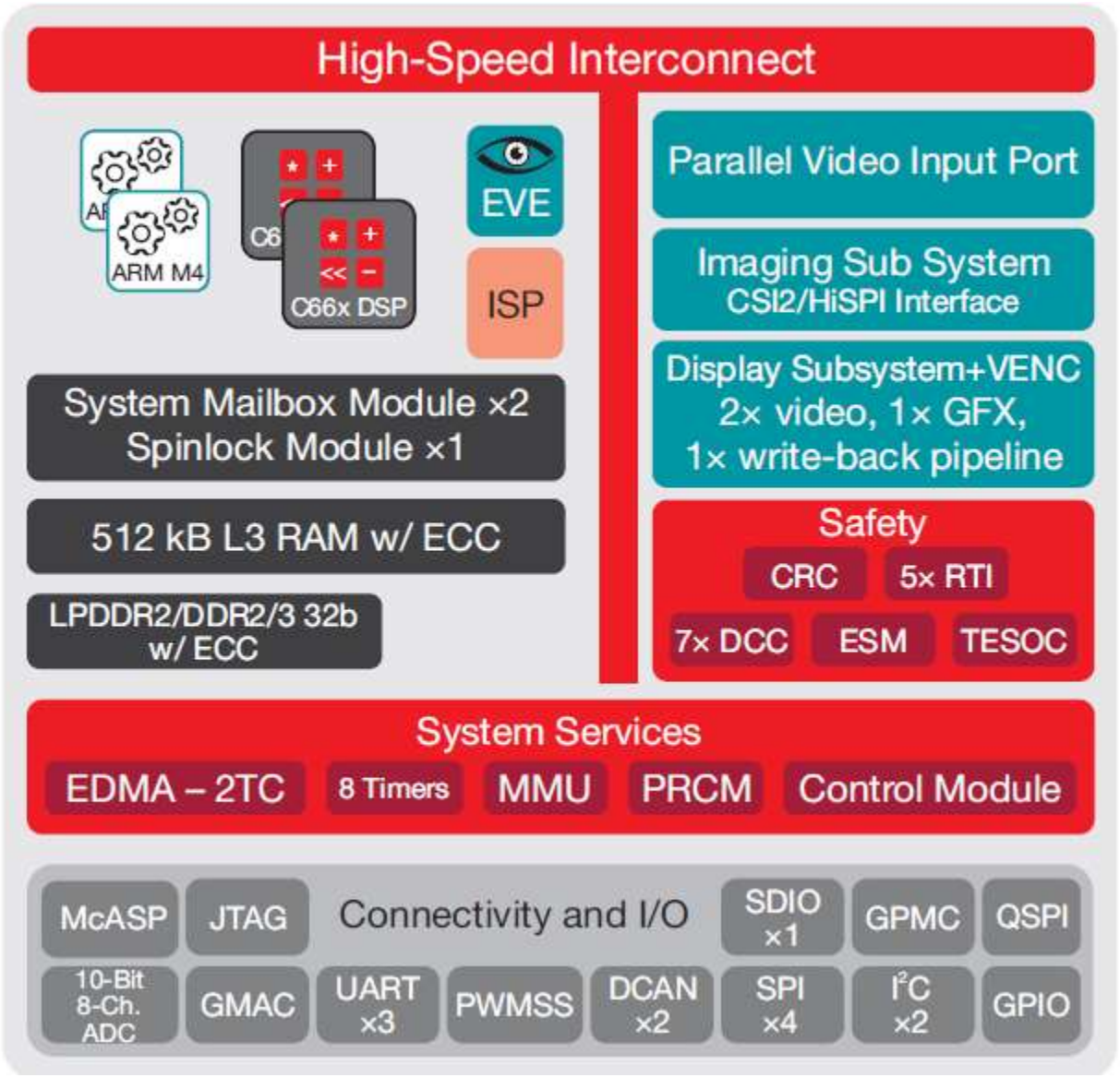


Figure 86:TDA3x Hardware components

Main advantages:

- The Soc is produced for ADAS system and Embedded vision projects especially.
- The Soc has different and powerful hardware components such as: EVE core, ISP core which especially added for computer vision projects.

Disadvantages:

- Doesn’t support OpenCV library which the core of the project.
- Only OpenGl library for computer vision projects
- Only support C & C++ programming languages, so we can’t run our Deep Learning approach which written in python language.
- Doesn’t support Embedded Linux but only support attached Software Development Kit – SDK which difficult to edit, compile and execute our code.
- Doesn’t support AI computing, so we can’t apply our Deep Learning approach.
- Some Hardware, Technical Support from TI needed aren’t available or unreachable.
- Doesn’t support CUDA programming so we can’t fast our execution.

Nvidia Jetson TX1 Performance

Block	Timing performance
IPM & ROI	8.326 ms
LSD	33.5048 ms
Filtration	1.7541 ms
Curve fitting	0.452 ms
Inverse IPM	17.6404 ms

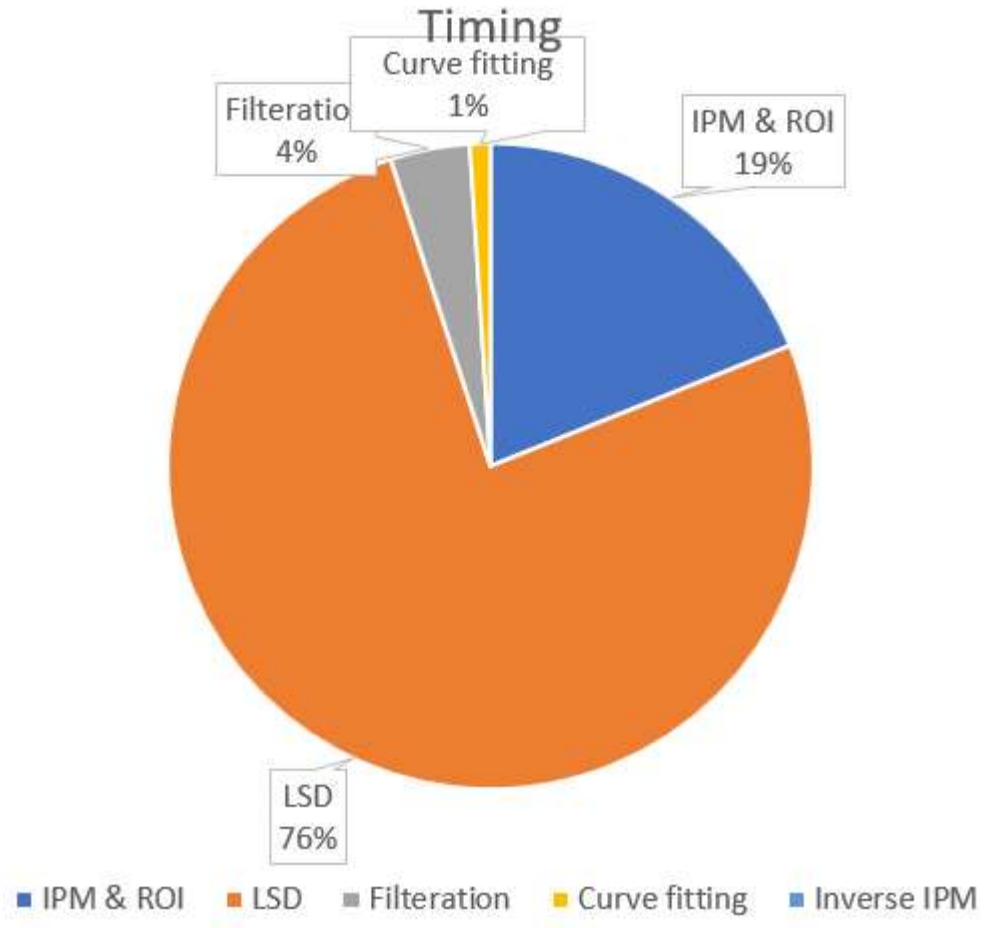


Figure 87: Kit performance

The frame takes 61.67 ms processing . The performance arrives on the kit more than 16 frame per second.

Comparison Between Hardware Performance

The following table shows the performance of the algorithm on different CPUs recording the execution time and the avg. frame per second on each**:**

	CPU: i7 2.6 GHz **RAM: 16 GB** **OS: Ubuntu Linux**	Jetson TX1 **CPU: octa-core cortex-A15 2GHz & cortex-A7 1.7GHz** **RAM: 2 GB** **OS: Ubuntu Linux**
Execution time per frame (mile seconds)	**13.7491**	**60**
Avg. frame per second	**72.72**	**16.66**

Final Conclusion:

It is concluded that there are different approaches that can be used to tackle the lane detection problem either by computer vision or deep learning.

Computer vision approaches like Hough transform , IPM and line segment detection algorithm and they vary from their performance and speed.

Computer vision results:

PC: The frame took 13.95 ms on PC. So, the performance arrives to 71 frame per second.

Nvidia Jetson TX1: The frame takes 61.67 ms on kit . The performance arrives on the kit more than 16 frame per second.

While deep learning approaches uses CNN for classification and detection and it can work in real time as this is the advantage it has over the computer vision approaches while computer vision approaches have the advantage of reliable performance and robustness over deep learning approaches and it needs parallelism in software and pipelining in hardware to have a real time performance.

Graphical User Interface – GUI

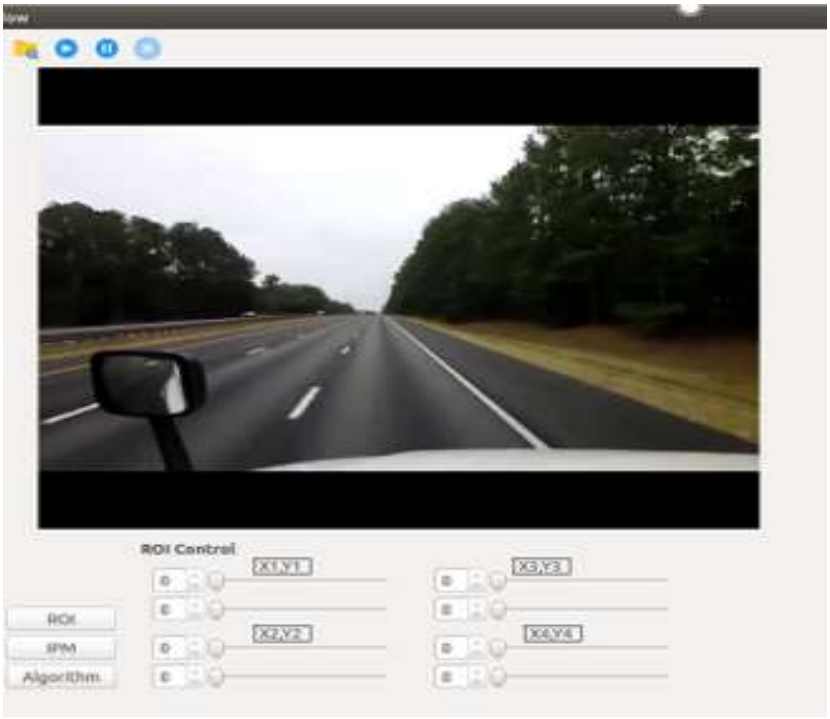


Figure 88: gui

Using Qt5 library based on C++ programming language, we designed a simple GUI to present the project.

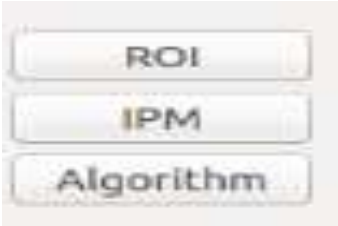
The main window contents:

- Original video window
- IPM , ROI and Algorithm push buttons
- ROI control group box
- Task Bar.

1- Original video window: show the video which was chosen by the user.

Figure 89: Original video window

2- IPM , ROI and Algorithm push buttons:



ROI push button: show the output from ROI block.

IPM push button: show the output from IPM block.

Algorithm push button: show the output frame.

4- Task bar: contain browse push button to choose the input video to the algorithm ,

Task bar contains:

- Browse push button to choose the input video to the algorithm.
- Play push button to play or resume the video
- Pause push button
- Stop push button

4- ROI control to choose the 4 corner points in the ROI

Future Work

1. Deep Learning - CNN

Model can be improved by using more data for training and by adding more layers and of course following this with regularization and data augmentation in order to prevent overfitting and it can also be improved by increasing the variations in training data to be able to work robustly under various circumstances.

2. Detect the road lanes using one line

The road lanes may be erased in some parts of the road and this may cause wrong detection A solution proposed is to use only one line to predict the location of the other line by mapping the points detected from the correctly detected line to the expected location of the other line using distance between the lines obtained from the previous clear lines.

3. Simulate virtual lines to the driver in case if there is no lane marks at all using techniques of deep learning and vehicle to vehicle communication

4. Lane switching assistance Assist the driver when switching lanes by recognizing the surrounding environment

References:

1. OpenCV Documentation. https://opencv.org
2. Qt Documentation. http://doc.qt.io/qt-5/
3. Harald Fernengel. Qt on Real Time OSs. https://www.slideshare.net/qtbynokia/qt-on-real-time-operating-systems/
4. Mohamed Aly: Real Time Detection of Lane Markers in Urban Streets; Proceedings of the IEEE Intelligent Vehicles Symposium; Eindhoven, The Netherlands. 4–6 June 2008.
5. Dajun Ding, Chanho Lee, Kwang Lee: An Adaptive Road ROI Determination Algorithm for Lane Detection; Proceedings of the TENCON 2013–2013 IEEE Region 10 Conference; Xi'an, China. 22–25 October 2013.
6. Toan Minh Hoang and Kang Ryoung Park: Road Lane Detection by Discriminating Dashed and Solid Road Lanes Using a Visible Light Camera Sensor.
7. Aaron Bobik, Irfan Essa and Arpan Chakraborty. Georgia Tech. Introduction to computer vision. **Udaciy**. https://www.udacity.com/course/introduction-to-computer-vision--ud810
8. N.N. Ahmed Salim, X. Cheng and X. Degui, 2014. A Robust Approach for Road Detection with Shadow Detection Removal Technique. Information Technology Journal, 13: 782-788.
9. Krishna Kant Singh, Kirat Pal, M.J. Nigam: Shadow Detection and Removal from Remote Sensing Images Using NDI and Morphological Operators, International Journal of Computer Applications, March 2012.
10. Ammu M Kumar and Philomina Simon,2015. Review of lane detection and tracking algorithms in advanced driver assistance system.
11. Pallavi V. Ingale and Prof. K. S. Bhagat. Comparative Study of Lane Detection Techniques.
12. Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, Xiaoou Tang,2017.Spatial As Deep: Spatial CNN for Traffic Scene Understanding.
13. Ian Goodfellow Deep Learning book