LANE DETECTION USING COMPUTER VISION AND MACHINE LEARNING FOR SELF-DRIVING CARS

An Undergraduate Research Scholars Thesis

by

RADHIKA SONI

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Dezhen Song

May 2022

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Radhika Soni, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Lane Detection using Computer Vision and Machine Learning for Self-Driving Cars

Radhika Soni
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. Dezhen Song
Department of Computer Science and Engineering
Texas A&M University

Self-Driving Cars are not only a reality today but a glimpse into how advanced and complex technology is going to be in the next century. The best and the brightest have been brought together by industry and academia around the world in this race to develop the best Autonomous vehicles possible. There's one important question which remains a topic of debate; which techniques should one use? Some researchers believe that traditional computer vision approaches are the answer, while several others have been utilizing deep learning-based approaches.

With this research, we compare which of these methods proves to be better in the case of lane detection, one of the most fundamental aspects of self-driving cars. The paper builds on previous research done in comparing the two approaches for various fields, and listing out the pros and cons of both approaches. The readers would have an in-depth understanding of the state-of-the-art techniques utilized for lane detection, giving them the ability to make an unbiased choice for their specific use case.

DEDICATION

To my family, instructors, peers, the Aggieland community, as well as organizers of the SAE/GM AutoDrive Challenge who supported and provided resources throughout the research process.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. Dezhen Song, and my PhD mentor Aaron Kingery for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my family for their encouragement, patience, and love.

The data used for this research was provided by The 12th Unmanned Team, with the support of the SAE/GM AutoDrive Challenge.

This work was also made possible in part by NetBot Laboratory at Texas A&M University.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Undergraduate research was supported by the Undergraduate Research Scholars Program & the Engineering Honors Program at Texas A&M University.

NOMENCLATURE

ROS	Robotic Operating System
ML	Machine Learning
CNN	Convolution Neural Network
MCity	Test Facility at University of Michigan in Ann Harbor
CV	Computer Vision
RGB	Red Green Blue
2D	Two-Dimensional
ROI	Region of Interest
GUI	Graphical User Interface
IoU	Intersection over Union
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative

1. INTRODUCTION

1.1 Motivation

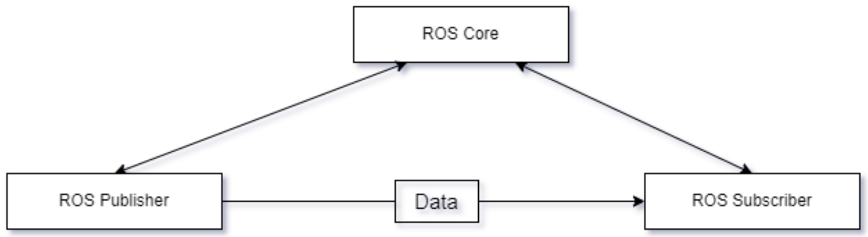
As humans we make hundreds of decisions every day, these decisions are typically made based on percepts we receive from the environment around us. Most of this perception is visual when it comes to driving. Self-Driving cars are autonomous vehicles that should be capable of detecting objects around them and take timely decisions to react to these impulses. Thus, there are several Computer Vision based problems when it comes to autonomous vehicles. Some examples include identifying objects like road signs, traffic signals, pedestrians, other cars, and lanes. The focus of this paper is Lane Detection, one of the most fundamental requirements for vehicle motion planning.

Motion planning is the core technology and an extremely challenging problem for autonomous cars. As defined by The Robotics Institute at the University of Michigan, “Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.” [1]. Initial steps of the motion planning process in self driving cars involve the detection of lanes, thereafter, following the trajectory of the lines, detecting the signs on the lanes, and lastly, obstacle detection to identify closed lanes. The main theme of the research is to identify which methods of detection should be best suited for the task of lane line detection, lane color detection as well as lane type detection. These could be classified into 2 broad types - traditional vision-based and deep learning-based.

1.2 ROS Pipeline

The Robot Operation System (ROS) is a set of software libraries and tools that help one develop and build robot applications. We used ROS for this project to streamline the process of input and output for all subsystems. Fig. 1.1 below displays the elements of the ROS pipeline:

- Publisher – Inputs the images into out ROS pipeline.
- Subscriber – Subscribes to the publisher to obtain information(image) from the node.
- Core – All subscribers and publishers connect using this master node(roscore)



```
def main()
   Create Object for Image Converter Class
   Create ROS Node
   try:
      Spin ROS Node
   except:
      Shut OpenCV Windows
class image_converter:
   Create CV Bridge
   Create Image Subscriber
   Create Image Publisher

def callback():
   try:
     Convert Image to OpenCV format
     Curved_lane_detection.find_street_lanes(Image)
   except CvBridgeError:
     break
   try:
     Publish Image to Topic
   except CvBridgeError as e:
     break
```

Fig. 1.1 Pseudocode for the ROS pipeline for Lane Detection

1.3 Traditional Computer Vision

Long before the development of deep learning algorithms traditional computer vision (CV) techniques were used to process visual information. The field dates back almost 60 years, when extracting information from visual data became prevalent. Lawrence Gilman Roberts, often attributed as the father of CV, first tried experiments on trying to construct and display a three-dimensional array of solid objects from a two-dimensional photograph. He mentions in his paper from 1965, “These assumptions enable a computer to obtain reasonable, three-dimensional description from the edge formation in a photograph by mean of topological mathematical process.” [2]. With better quality images, academia was then able to divide CV into more fields, out of which image segmentation and edge detection are the most popular ones utilized for lane detection. We chose a combination of edge detection and image segmentation for our research.

1.3.1 Edge Detection

Edge Detection is the essential technique used by most CV-based algorithms to detect lanes; it usually involves noting differences in intensity of the pixels to draw edges. If we consider the image as a function, a gradient would essentially be the slope of the image function. Edges are thus typically found with significant changes to the magnitude of these gradients between various regions of the image [3].

The most prevalent edge detectors currently used in the industry for lane detection are the Canny edge detector and the Sobel Edge Detector [4,5].

1.3.2 Canny Edge Detector

John Canny first introduced the Canny edge detection technique in his paper "A computational approach to Edge detection" [4]. The process for the Canny Algorithm can be shown with the following flow chart:

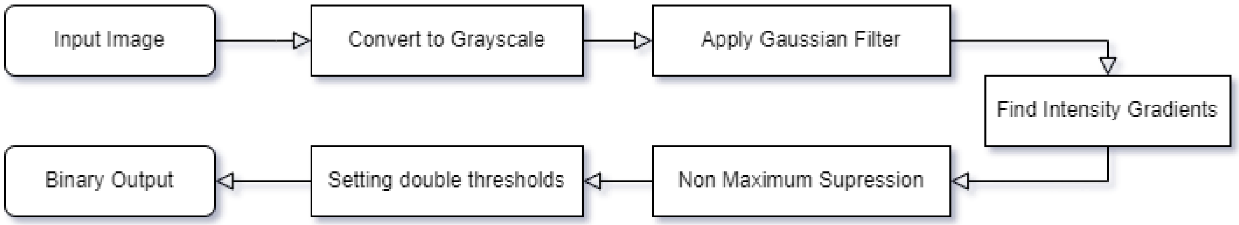


Fig. 1.2 Process for Canny Edge Detector [4]

Fig. 1.2 above is a simplified form of the process used by John Canny to develop the edge detection technique. The first step of the technique is to identify an input image and convert it to grayscale. Gray scaling is a technique which involves converting an RGB image with 3 channels, to each pixel being assigned just 1 value from 0-255. The next step in the process is applying a gaussian filter. The 2-D gaussian function is shown in Eqn. 1.1. This Gaussian filter applies a blur to each pixel of the image which helps reduce bright spots thus reducing noise.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad \textit{(Eqn. 1.1)}$$

The last necessary step in the process is to find gradient differences in the intensity of the pixels. Since our image is 2D, we get an x-gradient and a y-gradient. After the gradient thresholding, we receive our output binary image, which is in the desired form for post-processing.

1.3.3 Sobel Edge Detector

The Sobel operator is named after creators Irwin Sobel and Gary Feldman, who introduced the operator in a conference in 1968 & titled it as "An Isotropic 3 x 3 Image Gradient Operator" [5,6].

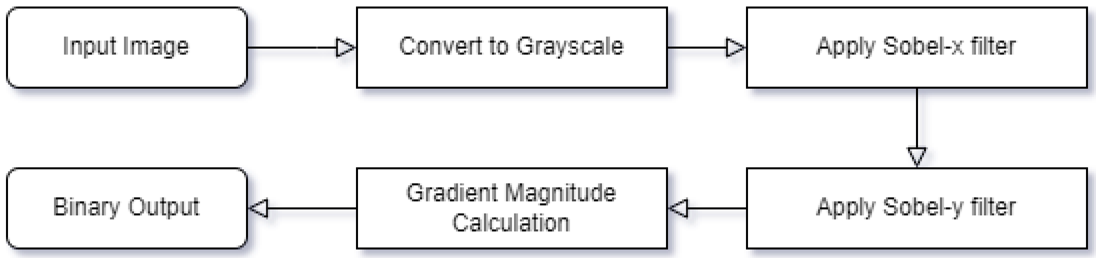


Fig. 1.3 Process for Sobel Edge Detector [6]

Fig. 1.3 above shows the steps used in the Sobel Edge Detector. The initial steps of the Sobel method are the same, i.e., conversion to grayscale and applying a Gaussian Blur. The next step is to apply the 3x3 kernels identified as the Sobel operator to the 2-D Gaussian filters, to receive gradients in the x and y directions respectively (Eqn. 1.2).

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \mathrm{A} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \mathrm{A} \qquad \textit{(Eqn. 1.2)}$$

1.4 Deep Learning

Deep Learning is the process of acquiring data, learn from it and perform a given task based on that learning. Deep Learning models were theoretically made around the 1960's but the compute power wasn't powerful enough to store and run these models. As compute power improved over the next few decades, deep learning models also became prevalent.

Training a deep learning model to perform a given task, large amounts of data are required. This training data must be appropriately labelled for your use case and must include enough edge cases e.g., shadows on roads can affect computer vision algorithms seeking edge detection as the method, but a deep learning model with a training set that includes images with road shadows would perform well.

Labelling is often a tedious and long process, so we decided to use pre-built datasets available as open source to train our Neural Networks. We also utilized some state-of-the-art models known for Lane Detection and tested them on our data directly.

Wang et al. [7] proposed LaneNet, a lane detection-based method consisting of two deep neural networks. Fig. 1.4 demonstrates the structure of the network based on LaneNet, Li et al [8]. We utilized this model to see results on our dataset without any additional training.

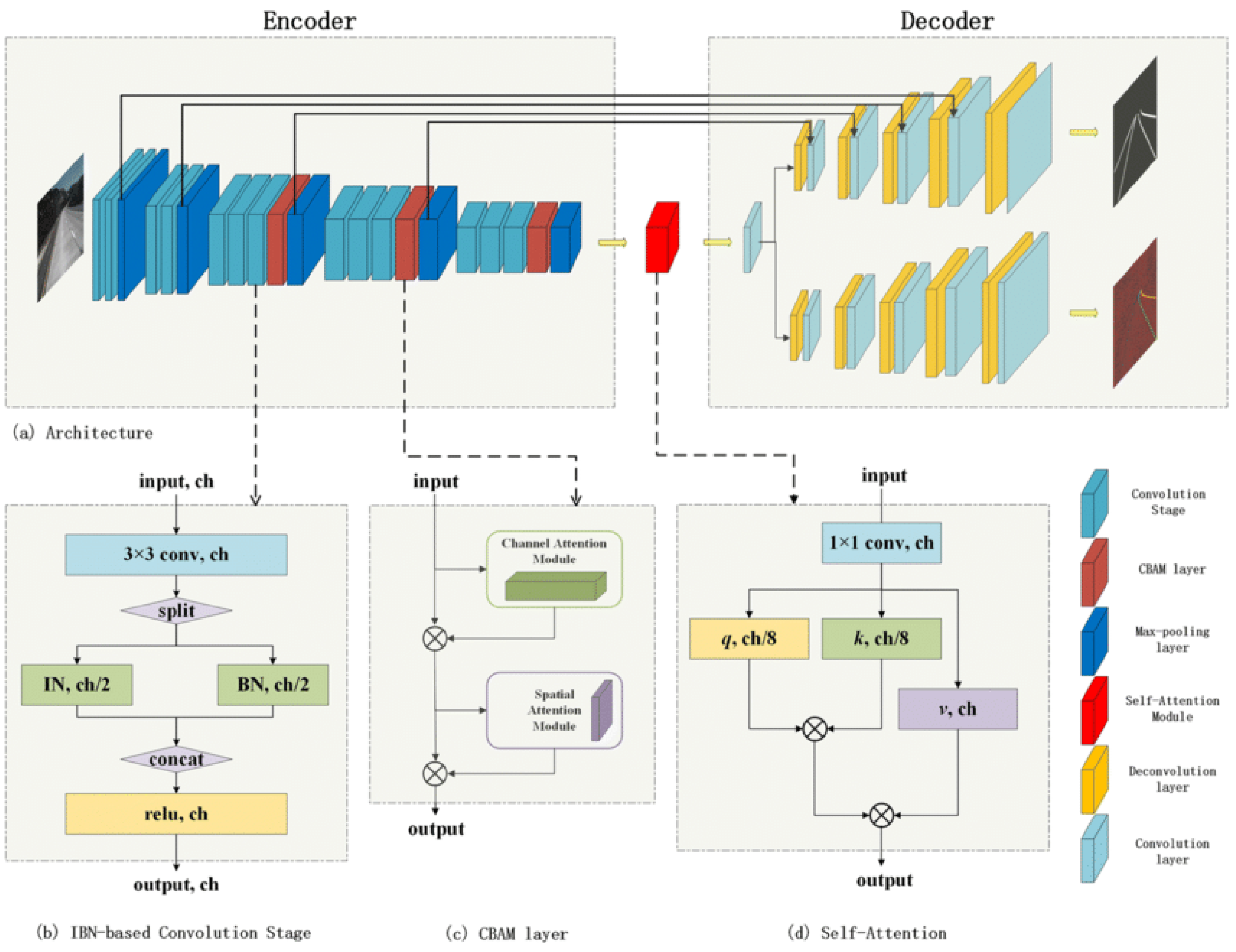


Fig. 1.4 Deep Learning Architecture based on LaneNet [8]

1.5 Research Objectives

The main objectives of the research are as follows:

1. Build a ROS pipeline, to communicate with the pre-existing system developed by the 12th man team. Identifying and fixing any bottlenecks for processing time. Testing the pipeline for sample data and deploying on the final data from MCity.
2. Using methods described above performing ***Straight Lane Detection*** and determining precision and accuracy.
3. Using methods described above performing ***Curved Lane Detection*** and determining precision and accuracy.
4. Identify and classify the type and color of the lanes detected using Computer Vision and Deep Learning.
5. Compare state-of-the-art ML models based on LaneNet [10] to compare accuracies for Traditional CV vs ML.

2. METHODS

2.1 Straight Lane Detection

2.1.1 Image Pre-processing

The steps we follow for pre-processing the image are depicted by the Fig. 2.1 below. The final output after the pre-processing is an image with the applied thresholds.

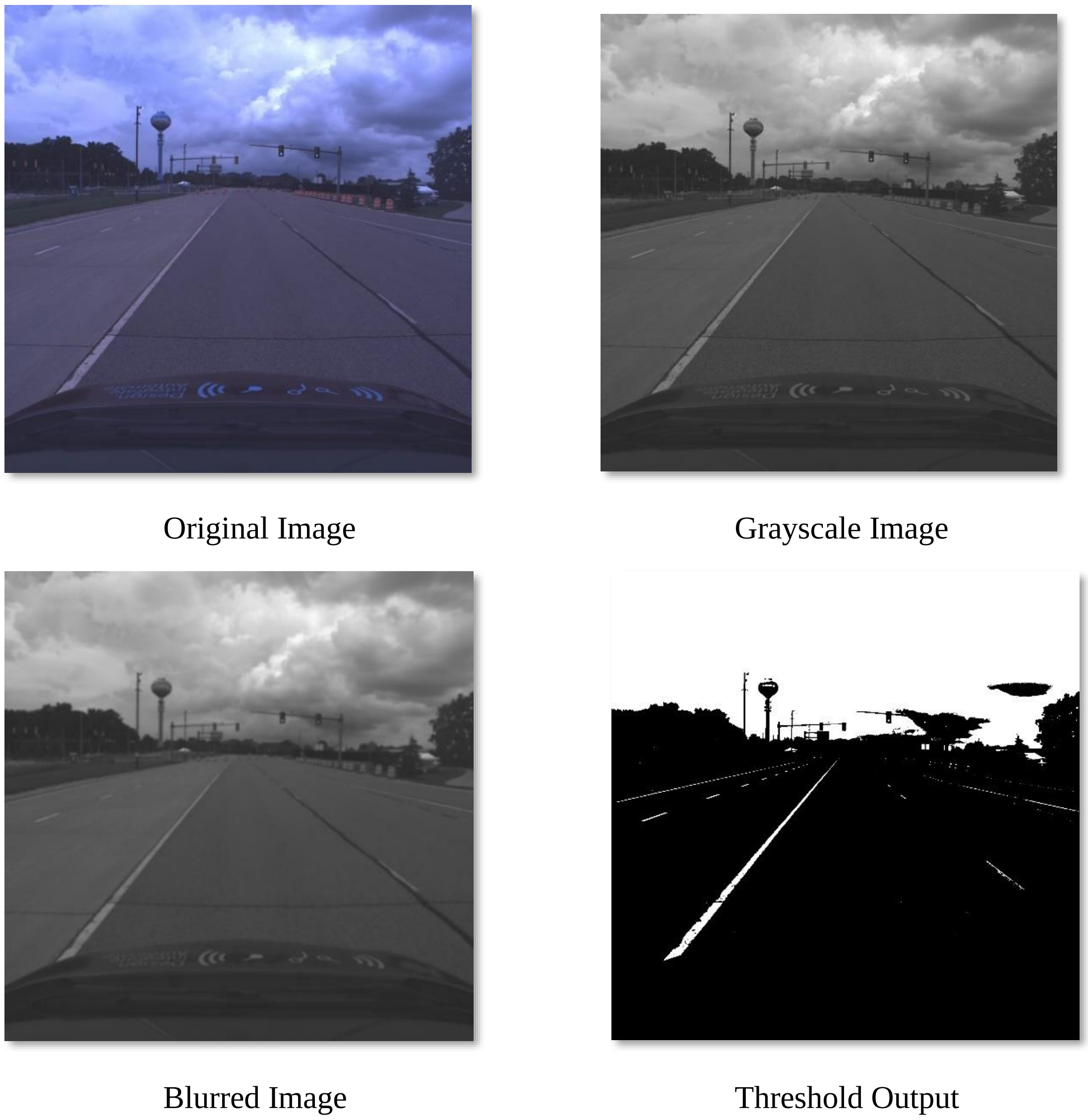


Fig. 2.1 Intermediate outputs of Image pre-processing Top Left is showing the Original Image, Top Right is showing grayscale image. Bottom Left shows the output of Gaussian Blur. Bottom Right shows final thresholded output for Straight Lane Detection

2.1.2 Application of Canny and Sobel Methods

We then used the threshold output image and applied the Canny filter method discussed under heading 1.3.2. Fig. 2.2 below shows the output for the Canny Filter on the thresholded image.

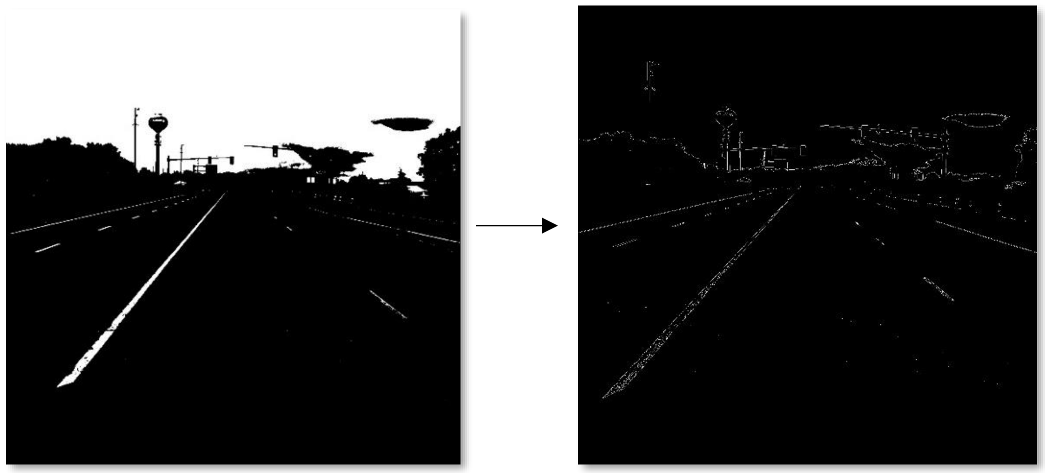


Fig. 2.2 Output for Canny Filter. Left image shows threshold output from pre-processing. Right image shows output for Canny Edge Filter.

2.1.3 Vanishing Point and Region of Interest (ROI)

The image obtained after applying the Canny and Sobel filters shows all edges needed in the picture for lane detection. The next important step is to identify our region of interest to avoid unnecessary edges from being detected. We thus, figure out a smaller area of interest. We perform this by calculating the vanishing point of the image and then forming a trapezoid with lines parallel to the location of the vanishing point. This concept of vanishing point was first introduced by Filippo Brunelleschi in the early 15th century and is one of the predominant laws of image spaces [9]. For our purposes, we manually calculate the vanishing point in our scenario since this is a challenging computer vision problem on its own.

For calculating the vanishing point, you first select the eligible edges for your detection. These edges meet at a certain line which is referred to as the vanishing point. The diagram below shows the vanishing point line for our image.

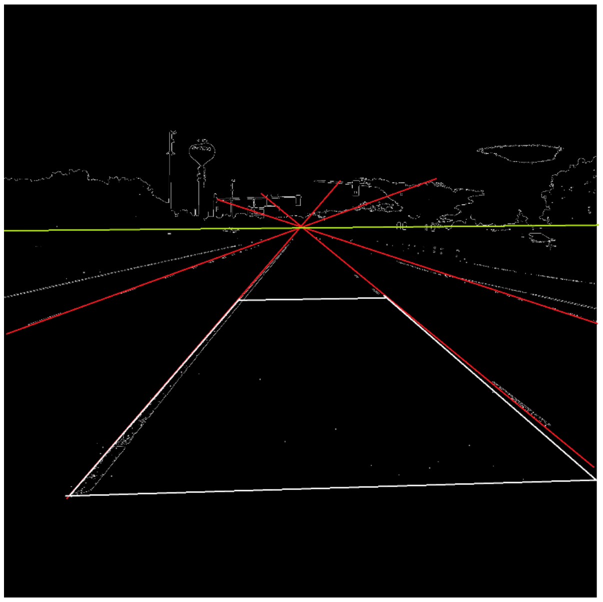


Fig. 2.3 Lines meeting at vanishing point. Green line is the Vanishing Point Line, Red lines are all lane edges present in the environment, White Box show the Region of Interest

Straight Lane detection is from a single perspective which means that the Vanishing point line matches the viewer's eye level, in our case the camera. The term Vanishing Point was first explained by Brooke Taylor in her book on Perspective in the 1800s and later compiled by Kristi Andersen in her book "The Geometry of an Art: The History of the Mathematical Theory" [10]. Since, the camera position stays uniform during detection, this calculation only needs to be done once. Fig. 2.3 shows the lanes on the road meeting up at the Vanishing point, thus helping us from our ROI.

2.1.4 Hough Line Transform

After identifying the ROI, we use Hough Line Transform as a feature extraction technique, to extract the lines we are interested in which are the lanes on the road in our case [11]. We utilized Probabilistic Hough Transform for our use case to obtain the pixels where lanes are located inside the ROI [12]. Fig. 2.4 below shows the final output of our ROS pipeline for Straight Lane Detection.

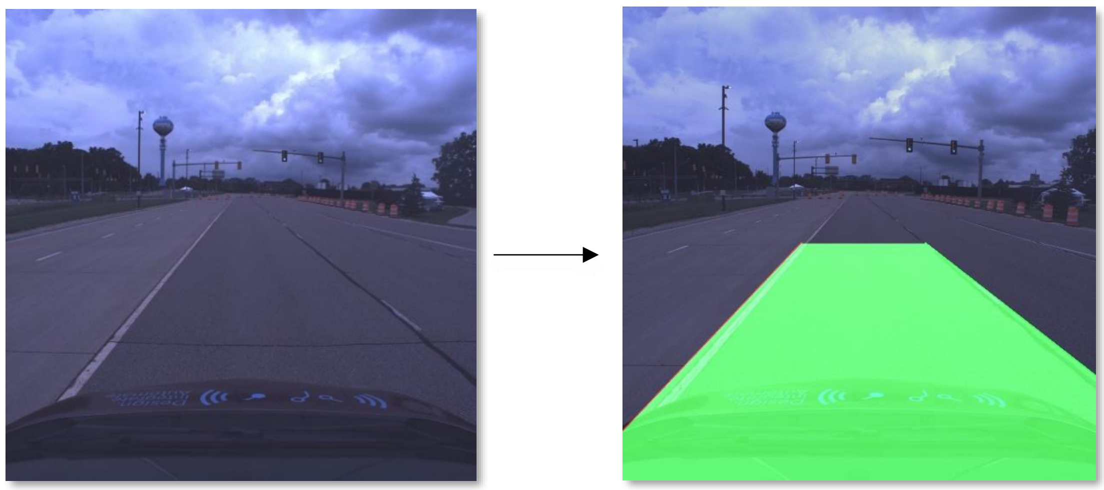


Fig. 2.4 Output for Straight Lane Detection, left image shows our original input image, Right side shows the final output for the detected lane. Green highlighted box represents area between detected lanes, red and green contours are used to differentiate between Left and Right lanes.

2.2 Curved Lane Detection

The Curved Lane detection problem is more difficult given the complexity of detecting curved lines. Several methods have been used for curved lane detection using computer vision. The method we choose for the thesis is called a Sliding Histogram Method for Lane Detection, He et al. visualized in Fig. 2.5, is the process we used for Curved Lane Detection [13].

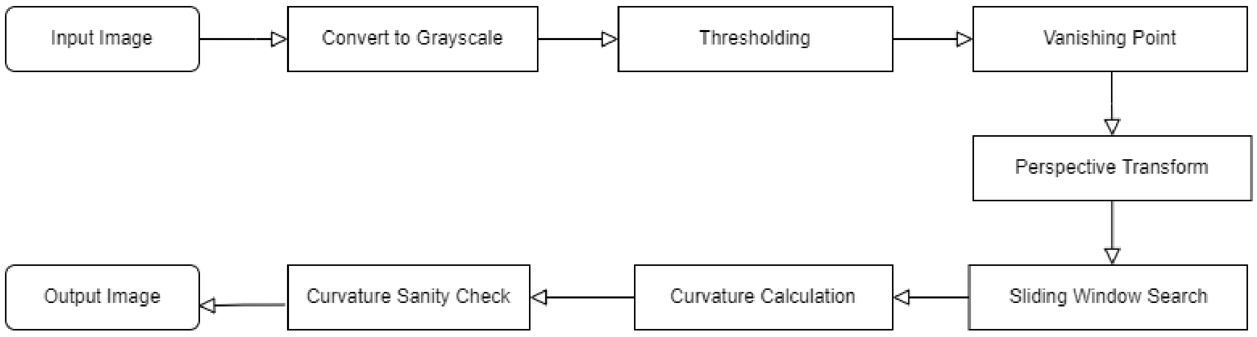
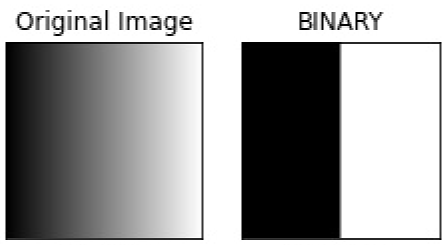


Fig. 2.5 Pipeline for Curved Lane Detection

2.2.1 Thresholding

Thresholding is a common technique utilized in most computer vision systems, wherein a given threshold is set for pixels, and a pixel's RGB values above the threshold are converted to white and the rest to black. This method of thresholding is often referred to as Binary thresholding and helps reduce noise in the image. An advanced method of thresholding called Adaptive thresholding was utilized in our process. Adaptive thresholding uses thresholding based on small regions around the pixel being threshold thus ensuring more uniform thresholding even with non-uniform lighting conditions [14]. We used OpenCV's open-source implementation for this calculation. Fig. 2.6 below shows the conversion scheme as well as the outputs on our sample image [15].



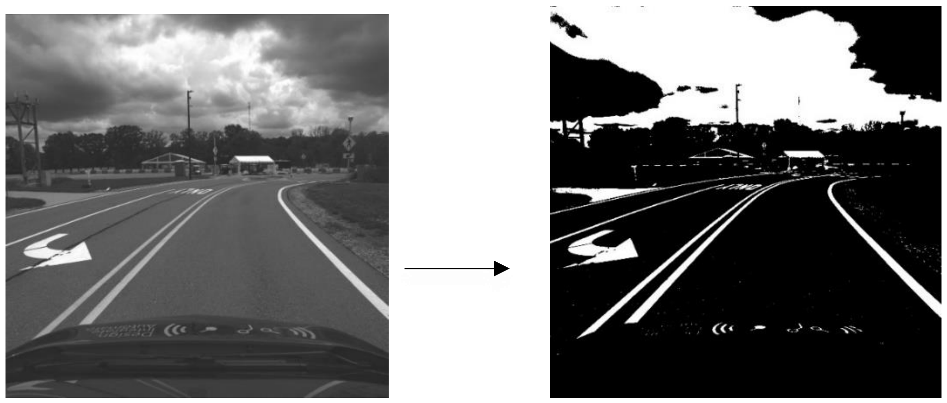


Fig. 2.6 Results for Adaptive thresholding, left image shows the grayscale image used as input, right image shows the output post-thresholding

2.2.2 Vanishing Point

Fig 2.7 below shows the next step in the process, i.e., detecting the vanishing point and identifying the ROI as we did in Straight Lane Detection (Section. 2.1.3).

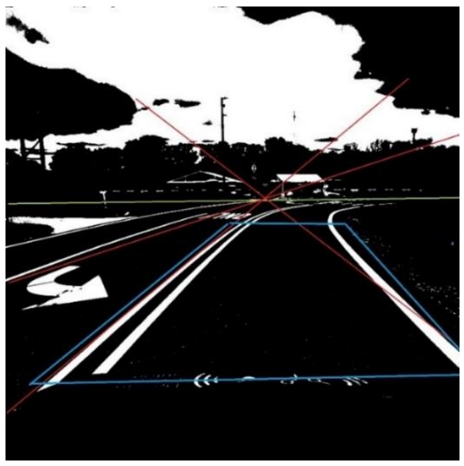


Fig. 2.7 Vanishing Point for Curved Lane Detection Green line is the Vanishing Point Line, Red lines are all lane edges present in the environment, Blue Box show the Region of Interest

2.2.3 Perspective Transform

Filippo Brunelleschi is also attributed the credit for writing the mathematical laws of perspective at the beginning of the Italian Renaissance, early in the 15th century. These laws played as the basis for the math behind 3D transform into 2D plane. One aspect of these perspective transforms can enable one to see the top view of an image. This is also known as Image Rectification and is performed using simple matrix calculations applied to every pixel. For our purpose we defined a set of source pixels, which is our ROI. Output of the transform gives a new image shown in Fig 2.8.

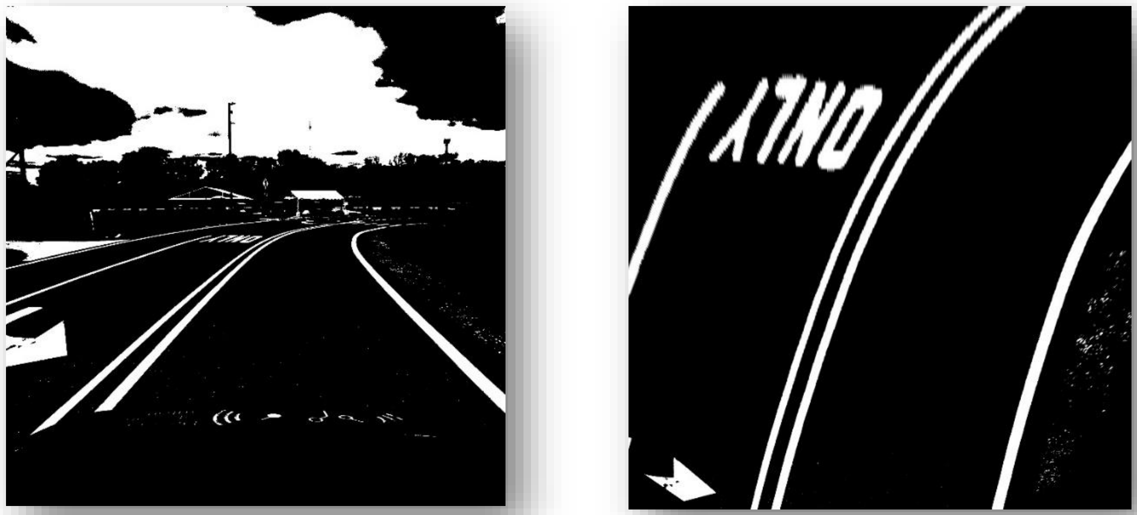


Fig. 2.8 Output for the perspective transform, left image is the output image from thresholding (Section 2.1.1), right image is the output for Perspective Transform

2.2.4 Sliding Window Search

The next step in the process is developing a histogram of the pixels found in our ROI and looking at the peaks of the histogram. Fig. 2.9 shows 3 lane lines, and the line histogram shows two peaks on the left and the right respectively. This gives us two arrays with pixels and their locations.

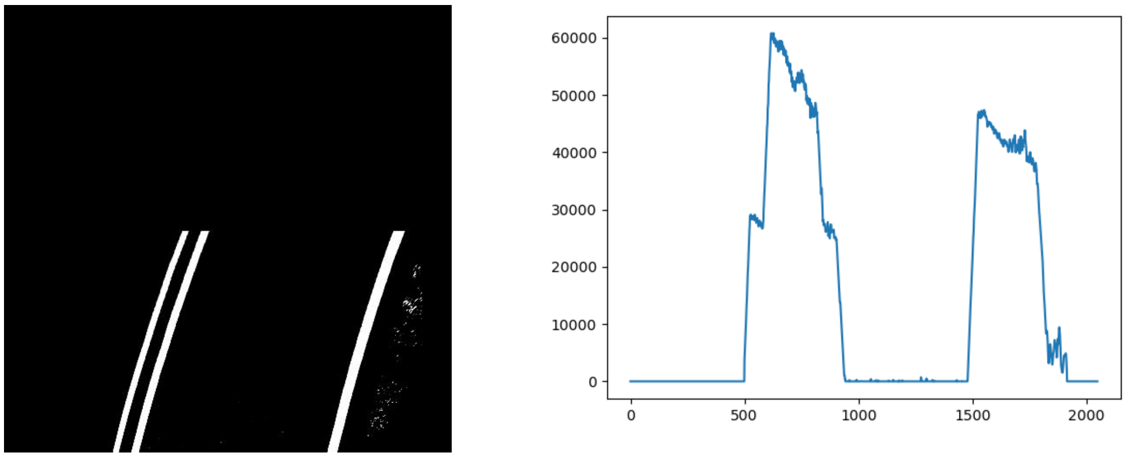


Fig. 2.9 ROI image on the left and histogram of pixel density on the right

2.2.5 Curvature Calculation

The final step in the detection process is to create windows of dx width and slide them across these arrays to determine the shift in curvature. We then fit a second order polynomial curve on these pixels and the curvature value is calculated using the formula for radius of curvature from a curve Eqn. 2.1.

$$R = \frac{\left(1 + \left(\frac{dy}{dx}\right)^2\right)^{3/2}}{\left|\frac{d^2y}{dx^2}\right|} \qquad \text{(Eqn. 2.1)}$$

The equation of curvature for our sample image (Fig 2.9) and diagram for radius of curvature generated using Desmos is shown in the Fig. 2.10 below [16].

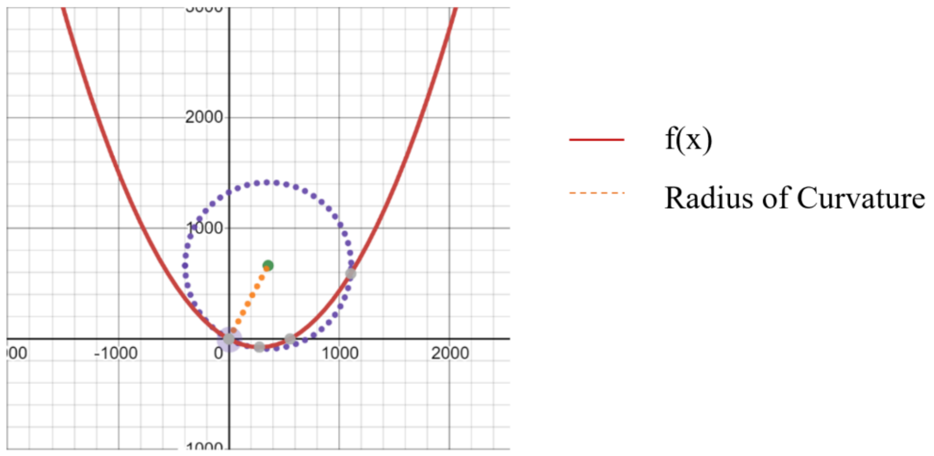


Fig. 2.10 Radius of curvature(R) of a sample second order curve f(x) plotted on Desmos [16]

Thus, we obtain the curves that fit these pixels, and these curves give the direction the road is turning towards. Fig 2.11 shows the results for the curved lane detection pipeline.

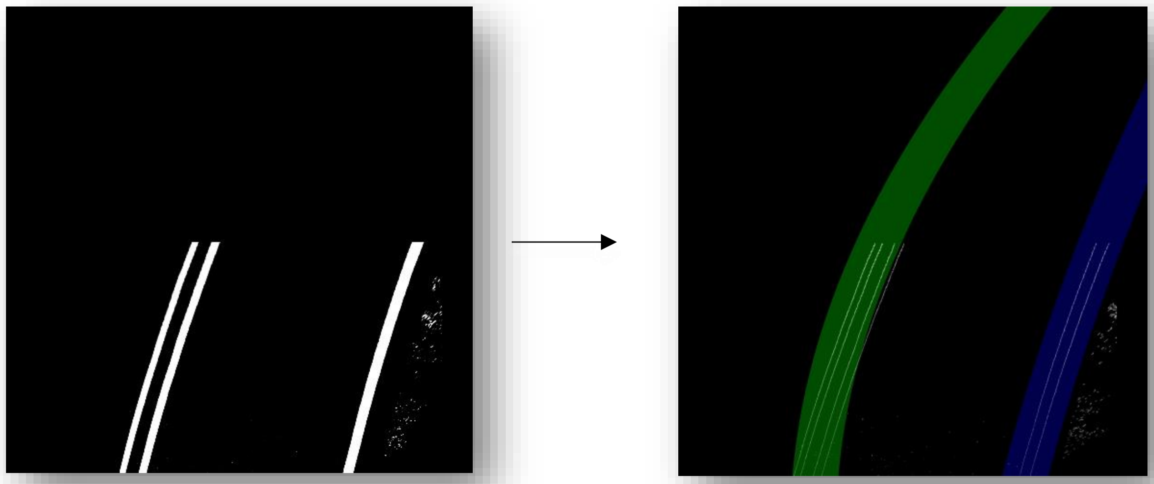


Fig. 2.11 Highlighted lanes for Curved Lane Detection, green and blue contour highlights are drawn to differentiate between Left and Right Lane.

2.3 Lane Type Detection

The lane type detection can be complex to solve using Deep learning since it would require accurate labeling to ensure correct detection, and as explained earlier this can be an expensive process. To our knowledge, there are no existing deep learning models which classify the lane marking type, making computer vision an essential method for this process.

We utilized the output for Straight Lane detection pipeline as input for this subsystem. The model density as shown in the histogram (Fig. 2.12) below shows a vast difference in the number of pixels between Solid and Dashed Lines.

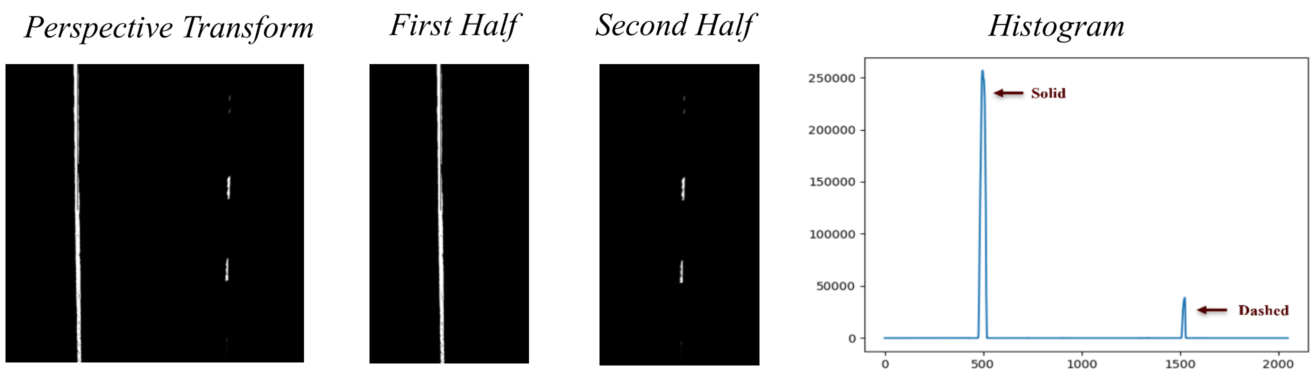


Fig. 2.12 Histogram showing detection of Solid and Dashed lanes, left binary image shows output image from perspective transform, split into two halves, the right image shows the histogram of pixel densities, Histogram generated using Python [17]

2.4 Lane Color Detection

Lane color detection is not typically solved using computer vision as there are other methods to determine where traffic is flowing in your surroundings using mapping and localization. We wanted to try to solve this problem with computer vision itself. The main challenge in identifying the lane color was caused by constantly changing light conditions. To account for this, we used a range of colors for both the yellow and the white lanes to ensure accuracy.

The steps of the process for Lane Color Detection are shown in Fig 2.13 below. We first apply gaussian blur to reduce pixelation after threshold masking. The next step in the process is to apply threshold masks in the yellow and white ranges.

Mask values can typically be represented as RGB values for the respective pixels, mapped from 0 to 255.The range we selected for the yellow mask was from [130, 60, 200] to [240, 240, 255]. Values for the White Mask ranged from [0, 0, 185] to [255, 70, 255].

As seen in Fig. 2.13, the yellow and white masks still have pixelation, which gets removed once we combine both masks, thus giving us access to nothing but the yellow and white pixels in the image. We then apply an ROI mask and obtain the output image showing yellow and white lanes. These can be filtered later to distinguish between the two colors.

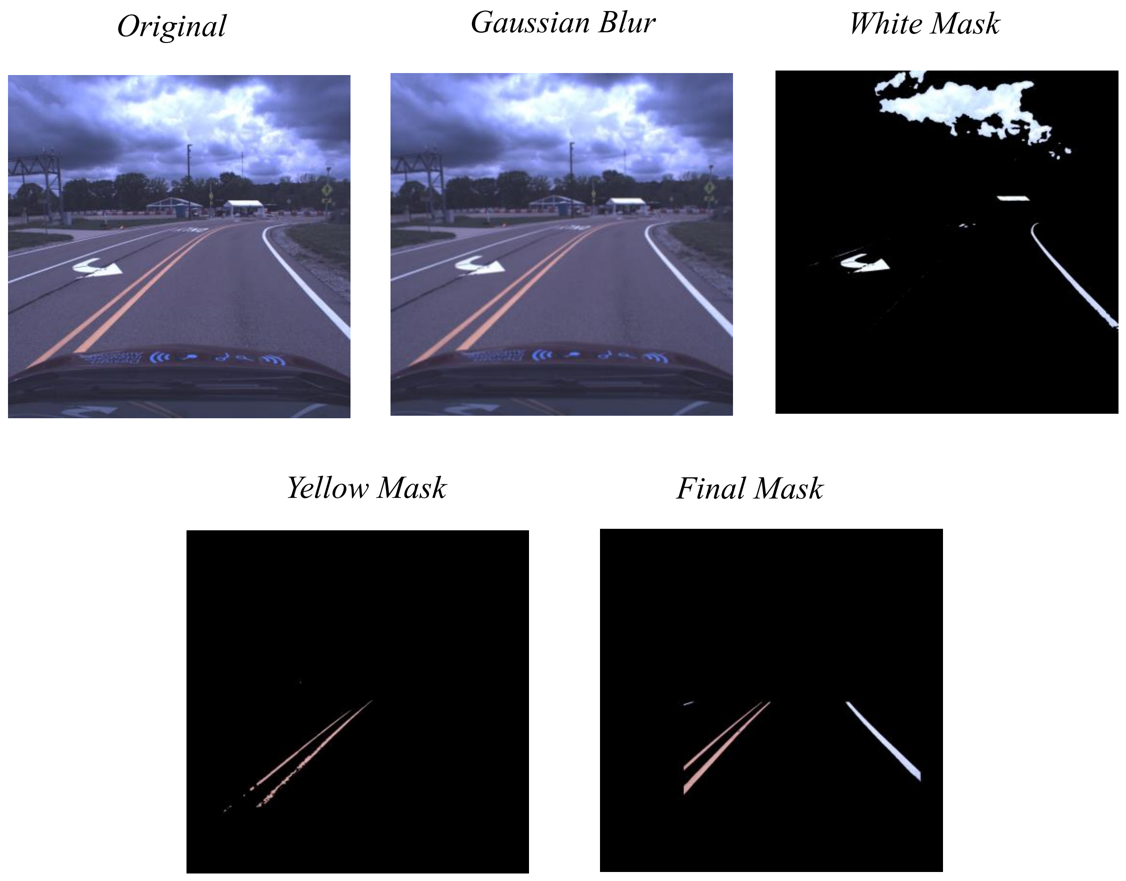


Fig. 2.13 Detection output for Yellow and White Lanes, top left is the sample image, top middle is the gaussian blur output, top right is output for white mask, bottom left is output for yellow mask, bottom right is output of yellow AND white mask.

2.5 Deep Learning Model

For the deep learning side, we utilized pre-existing models as mentioned in Section 1.4. We utilized the same ROS pipeline as established in Section 1.2 and passed images to the Deep Learning model for predictions. Fig 2.14 below shows the outputs for the Deep Learning Model:

```
def road_lines(recent_fit, avg_fit, model, image)
   Resize Image for Deep Learning Model
   Make prediction using Model
   try:
      Update Recent Fit
      Get Avg. Fit of last 5 fits
      Display lanes from Avg. fit
   except:
      Shut OpenCV Windows
```

Fig. 2.14 Pseudocode for Deep Learning detection pipeline [18] [19]

3. ANALYSIS

3.1 Dataset Overview

We collected samples from a few different categories to make a test set shown in Fig. 3.1. These samples differ in terms of environment lighting, direction of turn, fade on lanes etc. We also added some curved samples in the Straight Lane test and vice versa.

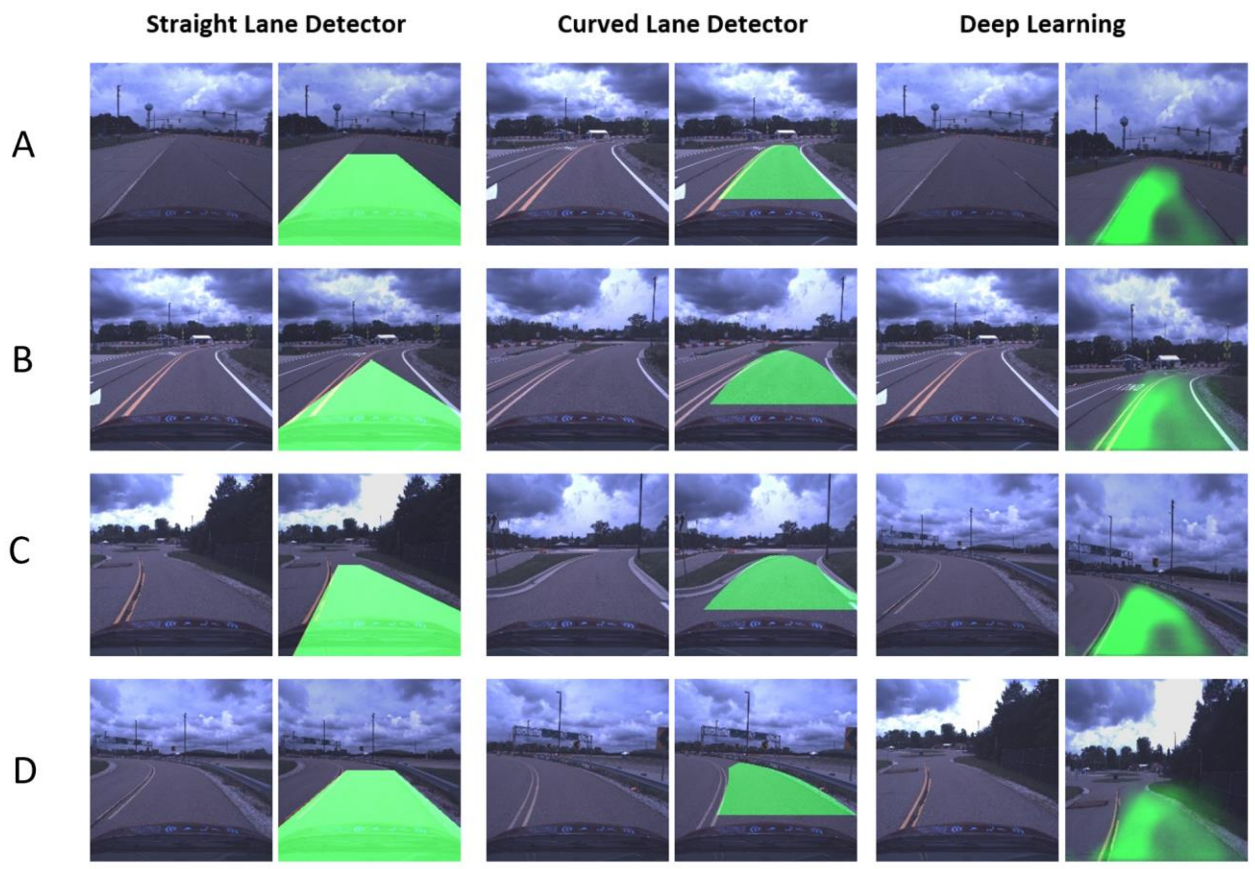


Fig. 3.1 Test samples for analysis, there are 3 columns, first one representing samples for straight lane, second for curved lane, and third for Deep Learning. Each column has two sub-columns, left one being input, right one as output.

3.2 Important Parameters

3.2.1 Test Statistics

True Positive (TP): Detects ***True*** when the actual value is ***True***

True Negative (TN): Detects ***False*** when the actual value is ***False***

False Positive (FN): Detects ***True*** when the actual value is ***False***

False Negative (FN): Detects ***False*** when the actual value is ***True***.

Sensitivity: Proportion of true positives that are correctly identified (Eqn. 3.1) [20].

$$Sensitivity = \frac{TP}{TP + TN} \qquad \textit{(Eqn. 3.1)}$$

Specificity: Proportion of true negatives that are correctly identified (Eqn. 3.2) [20].

$$Specificity = \frac{TN}{TN + FP} \qquad \textit{(Eqn. 3.2)}$$

3.2.2 Intersection Over Union (IoU)

IoU is commonly used as a similarity index for object detection problems in Computer Vision. As the name suggests we take intersection and unions of the expected output and the actual output and divide to get the score (Eqn. 3.3). This score ensures that the model is attributed for correct identification, while penalizing for incorrect identification. We chose test samples in each category to test IoU.

$$IoU = \frac{Detected\ Image\ \cup\ Mask\ Image}{Detected\ Image\ \cap\ Mask\ Image} \qquad \textit{(Eqn. 3.3)}$$

3.3 Results

The results for the detectors for all samples (Fig 3.1) are shown in Table 3.1 below:

Table 3.1 Accuracy Measures for Straight Lane Detection, Curved Lane Detection, and Deep Learning

Measures	**IoU**	**TP**	**FP**	**FN**	**TN**
Straight Lane Detector A	0.973	0.980	0.011	0.019	0.988
Straight Lane Detector B	0.872	0.916	0.459	0.085	0.542
Straight Lane Detector C	0.957	0.972	0.132	0.027	0.869
Straight Lane Detector D	0.932	0.989	0.073	0.011	0.928
Average	**0.933**	**0.964**	**0.168**	**0.035**	**0.831**

Measures	**IoU**	**TP**	**FP**	**FN**	**TN**
Curved Lane Detector A	0.979	0.980	0.124	0.019	0.876
Curved Lane Detector B	0.964	0.993	0.040	0.006	0.961
Curved Lane Detector C	0.955	0.961	0.217	0.041	0.783
Curved Lane Detector D	0.955	0.962	0.198	0.039	0.802
Average	**0.963**	**0.974**	**0.145**	**0.026**	**0.856**

Measures	**IoU**	**TP**	**FP**	**FN**	**TN**
Deep Learning Model A	0.891	0.933	0.296	0.682	0.704
Deep Learning Model B	0.789	0.779	0.511	0.284	0.489
Deep Learning Model C	0.795	0.782	0.549	0.279	0.452
Deep Learning Model D	0.8514	0.842	0.367	0.187	0.633
Average	**0.832**	**0.834**	**0.431**	**0.358**	**0.570**

3.3.1 Discussion

Computer Vision detectors, i.e., the Straight Lane Detector and the Curved Lane Detector performed surprisingly well in most test cases in our dataset. The performance of these methods however is dependent on various aspects of driving conditions. Some edge cases that we encountered where Computer Vision didn't perform well were as follows:

1. Presence of Shadows: Encountering shadows tend to mess up the CV algorithms we used since, shadows can be mistaken for gradient changes, and thus get detected as edges. This causes discrepancies in the model, thus giving unexpected results. There are however some algorithms for detection and removal of shadows that could be used for preprocessing [21].
2. Presence of Road Sign markings or Crosswalks: We also noticed that the Sliding Histogram method tends to mess up when crosswalks are encountered, since these crosswalks are the same color as the lane lines.
3. Double Yellow/White Lines: The histogram method can falsely detect the lane as being present between the two line of the left rather than the entire lane. We intend to resolve this issue by specifying a minimum distance between the detected lane lines, which should be wider than the car itself.
4. Unclear Lane Markings: Some edge cases included lane marking that were either worn out or covered by tar thus confusing the Computer Vision models into not detecting those as lines.

The Deep Learning model we utilized also performed surprisingly well given the fact that it was not specifically trained on our dataset. We intend to perform some transfer learning on these models to make them more specific for our use case. The drawbacks discussed above for the Computer Vision methods have been solved using Deep Leaning methods, given that you can label objects, essentially reducing the noise the model looks at to begin with.

4. CONCLUSION

4.1 Summary

The comparison of Deep Learning and Traditional CV was performed by the research, and there are assumptions, drawbacks as well as benefits of each of these methods. The key takeaways from the Computer Vision section were that CV is quite helpful in scenarios that needed quick processing, for instance tasks like Lane Color and Type detection. Deep Learning on the other hand comes quite handy when computer vision underperforms in the presence of varied lighting or shadows. However, one of the major drawbacks of Deep Learning is the quantity as well as quality of labeled data needed for the process.

The research is of course still ongoing, and we expect more rigorous results on the deep learning side with the help of some transfer learning, given that the model that we used was not specifically trained on our dataset. We hope to overcome these drawbacks of DL by pooling resources amongst research groups and put a step forward towards making self-driving cars safer.

REFERENCES

[1] “Motion planning,” *Michigan Robotics*, 17-Sep-2019. [Online]. Available: https://robotics.umich.edu/research/focus-areas/motion-planning/#:~:text=Motion%20planning%20is%20a%20term,building%20to%20a%20distant%20waypoint. [Accessed: 23-Mar-2022].

[2] L. G. Roberts, “Machine perception of three-dimensional solids,” thesis, Institute of technology, Cambridge Mass., 1963.

[3] “Canny edge detector,” *Wikipedia*, 28-Dec-2021. [Online]. Available: https://en.wikipedia.org/wiki/Canny_edge_detector. [Accessed: 21-Jan-2022].

[4] “A computational approach to edge detection,” *IEEE Xplore*, Nov-1986. [Online]. Available: https://ieeexplore.ieee.org/document/4767851. [Accessed: 21-Jan-2022].

[5] Sobel, Irwin & Feldman, Gary. (1973). "*A 3×3 isotropic gradient operator for image processing."* Pattern Classification and Scene Analysis. 271-272.

[6] “Sobel operator,” *Wikipedia*, 22-Dec-2021. [Online]. Available: https://en.wikipedia.org/wiki/Sobel_operator. [Accessed: 22-Jan-2022].

[7] Z. Wang, W. Ren, and Q. Qiu, “LaneNet: Real-time Lane Detection Networks for autonomous driving,” *arXiv.org*, 04-Jul-2018. [Online]. Available: https://arxiv.org/abs/1807.01726. [Accessed: 23-Jan-2022].

[8] Li, Wenhui & Qu, Feng & Liu, Jialun & Sun, Fengdong & Wang, Ying. (2020). A lane detection network based on IBN and attention. Multimedia Tools and Applications. 79. 10.1007/s11042-019-7475-x.

[9] “Vanishing point,” *Wikipedia*, 16-Jan-2022. [Online]. Available: https://en.wikipedia.org/wiki/Vanishing_point#:~:text=different%20vanishing%20points.-,Theorem,%CF%80%20and%20its%20vanishing%20point. [Accessed: 22-Jan-2022].

[10] Gray, Jeremy. (2009). Kirsti Andersen, The Geometry of an Art: The History of the Mathematical Theory of Perspective from Alberti to Monge , Springer, New York (2007) ISBN 10:0-387-25961-9. ISBN 13:978-0387-25961-1. xviii+802 pp. $199.00. Historia

Mathematica - HIST MATH. 36. 182-183. 10.1016/j.hm.2008.08.007.

[11] “Hough Line transform,” *OpenCV*. [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. [Accessed: 23-Jan-2022].

[12] “Feature detection,” OpenCV. [Online]. Available: https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga8618180a5948286384e3b7ca02f6feeb. [Accessed: 24-Mar-2022].

[13] J. He, S. Sun, D. Zhang, G. Wang and C. Zhang, "Lane Detection for Track-following Based on Histogram Statistics," 2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), 2019, pp. 1-2, doi: 10.1109/EDSSC.2019.8754094.

[14] F. Bergholm, *"Edge focusing,"* IEEE Trans. Putt. Anal. Mach. Intellig., vol. PAMI-9, pp. 726-741, Nov. 1987.

[15] “Image thresholding,” *OpenCV*. [Online]. Available: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. [Accessed: 01-Apr-2022].

[16] Desmos, Desmos, Inc, San Diego, California.

[17] Python , Python Software Foundation, Wilmington, Delaware.

[18] M. Virgo, “Mvirgo/MLND-capstone: Lane detection with deep learning,” GitHub. [Online]. Available: https://github.com/mvirgo/MLND-Capstone. [Accessed: 31-Mar-2022].

[19] Klintan, “Pytorch-lanenet: LaneNet implementation in Pytorch,” GitHub. [Online]. Available: https://github.com/klintan/pytorch-lanenet. [Accessed: 31-Mar-2022].

[20] Altman, D G, and J M Bland. “Diagnostic tests. 1: Sensitivity and specificity.” BMJ (Clinical research ed.) vol. 308,6943 (1994): 1552. doi:10.1136/bmj.308.6943.1552.

[21] H. Le and D. Samaras, “From shadow segmentation to shadow removal,” *arXiv.org*, 01-Aug-2020. [Online]. Available: https://arxiv.org/abs/2008.00267. [Accessed: 01-Apr-2022].