



People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Abdelhamid Ibn Badis University of Mostaganem
Faculty of Exact Sciences and Computer Science
Department of Mathematics and Computer Science



Speciality : Artificial Intelligence for Internet of Things

MASTER'S DEGREE PROJECT REPORT

Theme:

Road lanes detection in challenging conditions

Presented by the student : Benslimane Mustapha

Supervised by: Mr Moumene Mohammed El Amine

June 2025

Abstract

This thesis is the final report of the graduation project conducted as part of the Master 2 program in Artificial Intelligence for the Internet of Things (AI4IoT). The project focuses on the development of a classical lane detection system for autonomous driving applications, with the main objective of identifying lane lines in road scenes using traditional computer vision techniques.

The system relies on a sequence of image processing steps, including grayscale conversion, edge detection, region masking, and Hough Transform, to detect lane boundaries in each video frame. The TuSimple dataset was used to evaluate the method's effectiveness in real-world highway scenarios. A basic averaging technique was also applied to improve temporal stability across consecutive frames.

This project combines knowledge in computer vision, image processing, and real-time system design. It demonstrates how classical techniques, despite the rise of deep learning, can still offer efficient and interpretable solutions for specific tasks such as lane detection in structured driving environments. The work contributes to the broader field of intelligent transportation systems by providing a lightweight and explainable alternative to data-driven approaches.

Dedication and Acknowledgements

First and foremost, I thank Allah for granting me the strength, patience, and means to complete this work under the best conditions. I would like to express my sincere gratitude to all those who contributed to the realization of this final year project and I extend a special thanks to my academic supervisor, **Mr Moumene Mohammed El Amine**, for his attentive guidance, availability, and valuable advice throughout the project. His supervision was a great help in advancing my work. I also want to thank my family and friends for their moral support, patience, and constant encouragement. Their presence by my side was essential during the demanding moments of this journey. This thesis is dedicated to them, as a sign of gratitude and respect.

List of figures:

Figure N°	Figure title	Page
Figure 1	Before we apply the grayscale on the image	39
Figure 2	After applying the grayscale on the image	39
Figure 3	Before we apply CLAHE on the image	39
Figure 4	After applying CLAHE on the image	39
Figure 5	This shows on of the frames that will be combined together	40
Figure 6	Final result after combining all the consecutive	40
Figure 7	Before applying BilateralFilter on the image	40
Figure 8	After applying BilateralFilter on the image	40
Figure 9	The effect of the BilateralFilter in the Canny Edge Detector	41
Figure 10	the result of the Canny Detector on both a single image and on an averaged one	42
Figure 11	the result of the region_of_interest function	43
Figure 12	Displaying the result when applying the pipeline on one image	48
Figure 13	Displaying the results when applying the pipeline on a folder containing different road scene scenarios	49

Figure 14	Displaying the results when applying the pipeline on a structured dataset	49
-----------	---	-----------

List of Tables:

Table N°	Table title	Page
Table 1	Accuracy of the Model in Different Road Scenarios.	51

List of Abbreviations:

Abbreviation	The Full Expression
AI	Artificial Intelligence
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
RAW	Raw Image Format
MRI	Magnetic Resonance Imaging
CT	Computed Tomography
LoG	Laplacian of Gaussian
OCR	Optical Character Recognition
RGB	Red Green Blue
BGR	Blue Green Red
HSV	Hue Saturation Value
HOG	Histogram of Oriented Gradients

ConvNet,CNN	Convolutional Neural Network
ResNet	Residual Network
Mask R-CNN	Mask Region-Based Convolutional Neural Network
YOLO	You Only Look Once
API	Application Programming Interface
CV	Computer Vision
AGI	Artificial General Intelligence
IoT	Internet of Things
3D,4D	Three-Dimensional, Four-Dimensional
AR	Augmented reality
WHO	World Health Organization
ADAS	Advanced Driver Assistance System
ROI	Region of Interest
IPM	Inverse Perspective Mapping
SLT	Symmetrical Local Threshold
FNRF	Fast Neighborhood Reduction Filter
IPDA	Interacting Multiple Model Data Association
LSD	Line Segment Detector
DBSCAN	Density-Based Spatial Clustering
BLWS	Bresenham Line Writing Space
HSA	Harmony Search Algorithm

CLAHE	Contrast Limited Adaptive Histogram Equalization
RANSAC	Random Sample Consensus
KLT	Kanade-Lucas-Tomasi
MPC	Model Predictive Control
SUPRL	Neural Network-Based Real-Time Lane Detection
SVM	Support Vector Machines
ELM	Extreme Learning Machines
ANN	Artificial Neural Networks
RBF	Radial Basis Function
FCN	Fully Convolutional Networks
SBBOA	S-Shaped Binary Butterfly Optimization Algorithm
RNN	Recurrent Neural Network
JSON	JavaScript Object Notation
ROC	Receiver operating characteristic
TPR	True positive rate
NLP	Natural Language Processing
GPU	Graphical processing unit
CPU	Central Processing Unit
RAM	Random Access Memory
NMS	Non-Maximum Suppression
FPS	Frames per second

Table of Contents

General Introduction	10
Chapter 1: Introduction to Computer Vision	10
1.1 Introduction	
1.1.1 What is computer vision?	10
1.1.2 Applications and importance	11
1.2 Fundamentals of Digital Images	
1.2.1 What is an image?	11
1.2.2 Definition of a pixel	12
1.2.3 Color models (RGB, Grayscale and HSV)	12
1.2.4 Resolution and dimensions	13
1.3 Image Processing	
1.3.1 Basic operations (conversion, resizing, filtering)	13
1.3.2 Edge detection	14
1.3.3 Thresholding and binarization	14
1.3.4 Hough Transform	16
1.4 Introduction to Machine Learning in Vision	
1.4.1 Difference between traditional image processing and Machine Learning	17
1.4.2 Feature extraction	17
1.4.3 Classification and object recognition	17
1.5 Deep Learning and Computer Vision	
1.5.1 Convolutional Neural Networks (CNNs)	18

1.5.2 Object detection and segmentation	18
1.5.3 Example architectures (ResNet, YOLO, Mask R-CNN)	19
1.6 Applications and Use Cases	
1.6.1 Self-driving cars (lane detection, traffic signs)	20
1.6.2 Facial recognition	20
1.6.3 Industrial inspection	20
1.6.4 Medical imaging analysis	20
1.7 Tools and Libraries for Computer Vision	
1.7.1 OpenCV	20
1.7.2 TensorFlow/Keras	21
1.7.3 PyTorch	21
1.8 Conclusion and Future Perspectives	
1.8.1 Current limitations and challenges	21
1.8.2 The future of computer vision and AI	22

Chapter 2: State of the Art

2.1 Introduction

2.1.1 Overview of lane detection in autonomous vehicles.	23
2.1.2 Importance of accurate lane detection for autonomous vehicles.	24

2.2 Traditional Lane Detection Methods

2.2.1 Geometric and classical approaches	24
2.2.2 Limitations of traditional methods	28

2.3 AI-Based Lane Detection Approaches

2.3.1 Machine Learning-Based Methods	28
2.3.2 Deep Learning-Based Methods	29

2.4 Benchmark datasets for lane detection

2.5 Evaluation Metrics in Lane Detection

2.6 Challenges and Open Issues in Lane Detection	32
2.7 Conclusion	34
Chapter 3: methodology	
3.1 Introduction	34
3.2 Introduction to chosen method and dataset	
3.2.1 Overview of the chosen method	34
3.2.2 Justification of the chosen method	35
3.2.3 Justification of the chosen dataset	35
3.3 The implemented processing steps	35
3.4 Model evaluation	49
3.5 The libraries applied in this project	53
3.6 Conclusion	53
General Conclusion	54
List of References	56

General Introduction:

Road lane detection is an important challenge in the development of autonomous vehicles and advanced driver assistance systems and it plays a crucial role when it comes to helping drivers avoid unfortunate incidents. Even with the significant progress made in recent years, lane detection remains problematic because it still struggles when facing road challenges such as slandered or faded lane markings, obstruction from overtaking vehicles or some other objects and even adverse weather including fog, heavy rain and snow. The main objective of this work is to implement a deep learning based approach to significantly improve lane detection and tracking in complex situations by providing a generalized model that would give satisfactory results in real-time, which will be tested using one of the most common public benchmarks.

Chapter 1: Introduction to Computer Vision

1.1 Introduction:

This first chapter presents the foundations necessary for understanding the field of computer vision. We begin with a general definition of computer vision. Next, we introduce essential concepts related to digital images processing methods. Finally, we discuss the basic principles of machine learning and deep learning applied to computer vision, key elements to understand relevant approaches of road lanes detection.

1.1.1 What is computer vision? :

Computer vision is a field of artificial intelligence (AI) that enables computers to extract useful information from visual inputs like images and videos. Automated actions are then taken using the computer vision insights. Computer vision enables computers to "see" just as artificial intelligence (AI) enables them to "think." [1]

1.1.2 Applications and importance:

Computer vision is widely used in a variety of fields. It facilitates decision-making and automation across various industries. Here are a few important applications:

1. **Agriculture:** Advanced models for planting and harvesting are being developed by companies that specialize in agricultural technology. These technologies are also helpful for advanced weather analysis, weeding, and plant health detection. [1]
2. **Autonomous vehicles:** Self-driving cars, such as Teslas, use multi-camera setups, computer vision and 5G to evaluate their environment. This makes options such as 360-degree object identification and autopilot possible. These vehicles use computer vision for navigation, scene observation, and decision-making, and they can run manually or autonomously. [1]
3. **Healthcare:** Computer vision assists in medical imaging by identifying diseases and abnormalities. It analyses X-rays, MRIs, and other scans to enhance diagnostic accuracy. [2]
4. **Retail:** Computer vision is utilized in retail for customer behaviour analysis, inventory control, and theft prevention. It can track products on shelves and monitor customer movements. [2]
5. **Sports:** Computer vision is used in sports to generate highlights, analyse games, and track players. It allows for player and ball movements to produce useful statistics. [2]
6. **Manufacturing:** Computer vision plays a crucial role in modern manufacturing by improving quality control, preventing equipment failures, and enhancing packaging. It detects defects early, reducing waste and improving efficiency. [2]

1.2 Fundamentals of Digital Images

1.2.1 What is an image? :

An image is a visual representation of an object, scene, or pattern captured using cameras, sensors, or digital devices. It consists of tiny elements called pixels, each holding color and brightness information.

Images can be:

- **Digital:** Stored as pixel grids (e.g., JPEG, PNG).
- **Analog:** Captured on film or paper (e.g., printed photos).

In computer vision, images serve as input data for AI models to analyse, recognize objects, and extract useful information.

1.2.2 Definition of a pixel:

The unit of measurement for the dimensions and quality of a digital image or display is called a pixel, which is short for "picture element". It is used to describe the smallest digital image unit that a camera's image sensor can collect. Each pixel represents a specific color and brightness value, and together they make the photograph. When a cross-section of an image is magnified, the individual pixel becomes visible. The number of pixels in an image sensor determines the resolution and detail captured by the camera. Resolution refers to the number of pixels in a digital image or display and is typically expressed as the total number of pixels horizontally and vertically. [3]

1.2.3 Color models (RGB, Grayscale and HSV):

1. RGB: The RGB color model is the most often used color model in digital image processing and OpenCV. There are three channels in the color image. Each color has its own channel. The main color components of this model are red, green, and blue. All other colors are created using the proportional ratio of these three colors. Black is represented by a value of 0, and the intensity of the color increases as the value rises. [4]

2. HSV: There are three channels: hue, saturation, and value. Primary colors are not used directly in this color model. It uses color in the way humans perceive them. [4]

3. Grayscale: A grayscale image, as the name indicates, contains only shades of grey ranging from black to white. It is the simplest type of digital image format. In a grayscale image, each pixel is represented by a single number indicating its brightness or intensity. [5]

1.2.4 Resolution and dimensions

Pixel dimensions refer to the measurement of the width and height of an image in terms of pixels. Each pixel acts as a single point in that image, combining to create the whole image like tiny building blocks. The greater the number of pixels, the higher the resolution, meaning more detail and definition in the image. [6]

1.3 Image Processing

1.3.1 Basic operations (conversion, resizing, filtering):

- **Image Filtering:**

Image filtering is one of the fascinating uses of image processing. Image filtering is a technique for altering the size, shape, color, depth, smoothness, and other image properties. It alters the pixels of the image to transform it into the desired form using different types of graphical editing methods through graphic design and editing software. [7]

- **Image Conversion:**

Image conversion becomes essential when dealing with the diverse requirements of digital platforms, data analysis, and personal preferences. Whether it's optimizing images for faster web page loading with WebP, ensuring transparent backgrounds with PNGs, or preserving every detail in RAW files for post-processing, the need to convert images is universal in managing digital content. [8]

- **Image resizing:**

Image resizing (also called image scaling or resampling) is the process of changing the dimensions of a digital image by either increasing (upsampling) or decreasing (downsampling) its size. This is a fundamental operation in image processing and computer vision. In the image resizing process the dimension of an image is altered to meet specific requirements in image processing and computer vision tasks. Image resizing involves changing the number of pixels in the image, which affects both the visual size, and the amount of data needed to represent it.[9]

1.3.2 Edge Detection:

Definition:

Edge detection is one of the most important tasks in the field of computer vision and image processing, and edge provides important information for many subsequent visual tasks, such as image recognition, image segmentation, image retrieval, face recognition, corner detection, road detection, medical imaging, and target tracking. Each of these visual tasks requires extracting object boundaries or perceiving obvious edges from the original image, Therefore, ensuring the stability of detected image edges is crucial for efficiently performing these tasks. [10]

Existing Methods:

- **Sobel Operator:** Uses convolution with a pair of 3×3 kernels to detect edges in both horizontal and vertical directions by approximating the gradient of the image intensity.
- **Prewitt Operator:** Similar to Sobel but with a simpler kernel, used for detecting horizontal and vertical edges by computing intensity changes.
- **Roberts Cross Operator:** Uses a small 2×2 kernel to compute diagonal gradients, making it effective for detecting fine edges but sensitive to noise.
- **Canny Edge Detector:** A multi-stage algorithm that includes Gaussian smoothing, gradient calculation, non-maximum suppression, and hysteresis thresholding to detect strong and weak edges effectively.
- **Laplacian of Gaussian (LoG):** Uses a second-order derivative to detect edges based on zero-crossings after applying Gaussian smoothing to reduce noise

1.3.3 Thresholding and binarization

Binarization definition:

Image binarization is a crucial part of image analysis, including scene text detection and medical image analysis. Especially in the field of document image processing, binarization has a wide range of applications as a basic method of digital image processing, including text recognition, document image segmentation, image morphological processing, and feature extraction. It commonly serves as the primary stage in document analysis and recognition systems, as well as Optical Character Recognition (OCR), exerting a substantial impact on the efficacy of subsequent character segmentation and recognition. Binarization is a technique

used to separate the region of interest, such as text, from the background in an image, and it represents one of the fundamental methods of image segmentation. This process involves converting a grayscale image into a binary black and white image. Thresholding is a widely used tool in image segmentation, including global, local, and various automatic thresholding methods. [11]

Thresholding definition:

Image thresholding works on grayscale images, where each pixel has an intensity value between 0 (black) and 255 (white). The thresholding process involves converting this grayscale image into a binary image, where pixels are classified as either foreground (object of interest) or background based on their intensity values and a predetermined threshold. Pixels with intensities above the threshold are assigned to the foreground, while those below are assigned to the background.

Certain Thresholding Techniques:

- **Simple Thresholding:** Simple thresholding uses a single threshold value to classify pixel intensities. If a pixel's intensity is bigger than the threshold, it is set to 255 (white); otherwise, it is set to 0 (black).
- **Adaptive Thresholding:** Adaptive thresholding is used for images with non-uniform illumination. Instead of a single global threshold value, it calculates the threshold for small regions of the image, which allows for better handling of different lighting conditions.
- **Otsu's Thresholding:** Otsu's method is an automatic thresholding technique that finds the optimal threshold value by minimizing the -intra-class variance (the variance within the foreground and background classes).
- **Multilevel Thresholding:** Multilevel thresholding extends simple thresholding by using multiple threshold values to segment the image into more than two regions. This is useful for images with complex structures and different intensities.

- **Color Thresholding:** Color Thresholding In color images, thresholding can be applied to each color channel (e.g., RGB, HSV) separately. This method Takes advantage of color information to segment objects.
- **Local Thresholding:** Local thresholding calculates a various threshold for each pixel based on its local neighborhood. This method is effective for images with non-uniform illumination or varying textures.
- **Global Thresholding:** Global thresholding uses a single threshold value for the whole image. This technique is suitable for images with uniform lighting and clear contrast between the foreground and background.
- **Iterative Thresholding:** Iterative thresholding begins with an initial guess for the threshold value and iteratively refines it based on the mean intensity of the pixels above and below the threshold. The process continues until the threshold value converges.[12]

1.3.4 Hough transform:

Hough Transform is a computer vision technique that detects shapes like lines and circles in an image. It converts these shapes into mathematical representations in parameter space, making it easier to identify them even if they're broken or covered. This method is valuable for image analysis, pattern recognition, and object detection.

The 9 Transform algorithm in line detection is a feature extraction method in image analysis, computer vision, and digital image processing. It uses a voting mechanism to identify bad examples of objects inside a given class of forms. [13]

1.4 Introduction to Machine Learning in Vision

1.4.1 Difference between traditional image processing and Machine Learning:

"Traditional computer vision techniques and deep learning-based approaches differ fundamentally in their methodologies and the way they handle visual data. Traditional computer vision relies heavily on manual feature extraction and domain-specific knowledge. Techniques such as edge detection, texture analysis, and HOG (Histogram of Oriented Gradients) descriptors are examples where predefined algorithms are used to identify patterns and features in images. These methods require significant human intervention to design and fine-tune

features that are relevant to the task at hand. On the other hand, deep learning-based approaches, particularly those utilizing convolutional neural networks (CNNs), automate feature extraction. In deep learning, the model learns to identify features directly from the data through multiple layers of processing. Each layer extracts increasingly complex features from the raw pixel data, allowing the model to learn hierarchical representations of the input images. This process reduces the need for manual feature engineering and enables the model to adapt to a wide range of visual tasks with minimal human intervention. Another key difference lies in their performance and scalability. Traditional methods often struggle with large datasets and complex tasks because they rely on handcrafted features that may not generalize well across different scenarios. In contrast, deep learning models excel in handling large volumes of data and can achieve superior performance on tasks such as image classification, object detection, and segmentation, given sufficient data and computational resources. The ability to leverage vast amounts of labeled data for training allows deep learning models to improve accuracy and robustness significantly." [14]

1.4.2 Feature extraction:

Feature extraction in image processing is the process of identifying and isolating relevant information or attributes from an image that are useful for tasks such as object recognition, image classification, and tracking. These features can be edges, textures, corners, or any other distinct patterns that help in identifying important parts of an image. The purpose of feature extraction is to reduce the complexity of an image while retaining the important information needed for further analysis. For example, in edge detection, techniques like Canny edge detection or Sobel filters are applied to identify boundaries or transitions between different regions of an image. Once features are extracted, they can be used for classification, matching, or even for further analysis like pattern recognition. Feature extraction reduces the dimensionality of image data, making it more manageable for algorithms and improving the speed of subsequent processes, such as machine learning classification. In applications like medical image analysis, feature extraction plays a vital role in identifying tumors, abnormalities, or other conditions based on specific features in the image. [15]

1.4.3 Classification and object recognition:

Image classification assigns a label to an entire image, while object localization identifies and marks the position of one or more objects by drawing bounding boxes. Object detection is a more complex task that combines both classification and localization, labelling each detected object and outlining its position within the image. Collectively, these processes fall under the general field of object recognition. [16]

1.5 Deep Learning and Computer Vision

1.5.1 Convolutional Neural Networks (CNNs):

Neural networks are a subset of machine learning and form the foundation of deep learning algorithms. They consist of multiple layers, including an input layer, hidden layers, and an output layer, where each node processes and transmits data based on weighted connections and threshold values. Among these, convolutional neural networks (ConvNets or CNNs) are specifically designed for computer vision tasks. Unlike traditional neural networks that process data in a fully connected manner, CNNs utilize convolutional layers to automatically extract spatial features from images. This advancement replaces manual, time-consuming feature extraction methods with a more scalable approach, leveraging matrix multiplication to recognize patterns in images. However, this efficiency comes at a cost, as CNNs require significant computational resources, often relying on graphical processing units (GPUs) to train models effectively. [17]

1.5.2 Object detection and segmentation:

Image segmentation plays a critical role in various applications, from medical diagnostics to autonomous driving. It is a fundamental yet complex task in computer vision, as it requires detailed low-level spatial information. In recent years, the success of deep convolutional neural networks (CNNs) has significantly influenced the field of segmentation, leading to the development of highly effective models. These models enhance accuracy by learning intricate patterns and structures within images, making segmentation more reliable and efficient than traditional methods. [18]

1.5.3 Example architectures (ResNet, YOLO, Mask R-CNN):

Several deep learning architectures have been developed to enhance image classification, object detection, and segmentation. Among them, ResNet, YOLO, and Mask R-CNN stand out due to their efficiency and accuracy in computer vision tasks.

- **ResNet (Residual Networks):**

Introduced by Kaiming He and colleagues in 2015, ResNet is a convolutional neural network designed to help train very deep networks. Its key feature is "skip connections" that allow gradients to flow through the network more effectively, making it possible to train much deeper networks than before. These connections help address the vanishing gradient problem by allowing the network to learn residual functions, stabilizing training and improving performance.

ResNet has been widely applied in various computer vision tasks and comes in several versions, like ResNet-18, ResNet-50, and ResNet-152, indicating the number of network layers. Its innovative approach has significantly influenced deep learning research and applications. [18]

- **YOLO (You Only Look Once):**

Was published in CVPR 2016 by Joseph Redmon et al. the YOLO framework has stood out for its remarkable balance of speed and accuracy, enabling the rapid and reliable identification of objects in images. Since its inception, the YOLO family has evolved through multiple iterations, each building upon the previous versions to address limitations and enhance performance. It presented for the first time a real-time end-to-end approach for object detection. The name YOLO stands for "You Only Look Once," referring to the fact that it was able to accomplish the detection task with a single pass of the network, as opposed to previous approaches that either used sliding windows followed by a classifier that needed to run hundreds or thousands of times per image or the more advanced methods that divided the task into two-steps. Also, YOLO used a more straightforward output based only on regression to predict the detection outputs. [19]

- **Mask R-CNN:**

Extending Faster R-CNN, Mask R-CNN performs both object detection and instance segmentation. It adds a branch for predicting segmentation masks alongside bounding boxes and class labels, making it ideal for tasks that require detailed object segmentation, such as medical image analysis and video understanding. [20]

1.6 Applications and Use Cases

1.6.1 Self-driving cars (lane detection, traffic signs) :

Computer vision helps autonomous vehicles detect lanes, recognize traffic signs, and avoid obstacles using cameras and deep learning models. Algorithms like CNNs process real-time video feeds to identify road markings and interpret signals.

1.6.2 Facial recognition:

Facial recognition systems use computer vision to detect and verify faces in images or videos. This is applied in security systems, smartphone unlocking, and biometric authentication by analysing facial features and comparing them with stored data.

1.6.3 Industrial inspection:

In manufacturing, computer vision detects defects in products, ensuring quality control. Cameras and AI models analyse images of items on assembly lines, identifying cracks, misalignments, or other defects automatically.

1.6.4 Medical imaging analysis:

In healthcare, computer vision is used to analyse medical images like X-rays, MRIs, and CT scans. AI models help doctors detect diseases, such as tumors or fractures, by highlighting abnormalities in scans.

1.7 Tools and Libraries for Computer Vision

1.7.1 OpenCV:

OpenCV (Open Source Computer Vision Library) is a popular open-source library for real-time computer vision and image processing. It provides tools for image manipulation, object detection, feature extraction, and machine learning, making it widely used in both research and industry applications.

1.7.2 TensorFlow/Keras:

TensorFlow is an open-source deep learning framework developed by Google, commonly used for building and training machine learning models. Keras, which is integrated into TensorFlow, provides a high-level API for creating deep learning models with ease, making it a preferred choice for tasks like image classification and object detection.

1.7.3 PyTorch:

PyTorch is an open-source deep learning framework developed by Facebook, designed for flexibility and ease of use. It supports dynamic computation graphs, making it popular for research and production. PyTorch is widely used in computer vision applications, including image classification, object detection, and segmentation, with frameworks like Mask R-CNN and YOLO built on top of it.

1.8 Conclusion and Future Perspectives

1.8.1 Current limitations and challenges:

- **Poor data quality:**

High-quality labelled and annotated datasets are the foundation of a successful computer vision system. In industries such as healthcare, where computer vision technology is being significantly used, it is crucial to have high-quality data annotation, and labelling since the consequences of inaccurate computer vision systems can be significantly damaging. For example, many tools built to catch Covid-19 failed due to poor data quality.

- **Lack of training data:**

Collecting relevant and sufficient data can have various challenges. These challenges can lead to a lack of training data for computer vision systems. For example, gathering medical data is a challenge for data annotators. This is mainly due to the sensitivity and privacy aspects of healthcare data. Most medical images are either of a sensitive nature or are strictly private and are not shared by healthcare professionals and hospitals. Additionally, it is possible that the developers do not have the resources to collect sufficient data.

- **Inadequate hardware:**

Computer vision technology is implemented with a combination of software and hardware. To ensure the system's effectiveness, a business needs to install high-resolution cameras, sensors, and bots. This hardware can be costly and, if it's not optimal or improperly installed, can lead to blind spots and ineffective CV systems. [21]

1.8.2 The future of computer vision and AI:

The future of computer vision holds enormous potential across various industries, including healthcare, automotive, retail, and security. The integration and development of computer vision with other technologies, such as machine learning and deep learning, is paving the way for innovative applications and solutions that can significantly enhance operational efficiency and drive business growth. The future of computer vision and AI is expected to be shaped by several key advancements:

1. **Computer Vision and Artificial General Intelligence (AGI):** The development of AGI aims to create AI systems that can generalize learning across different tasks, making computer vision more adaptable and capable of human-like reasoning. Instead of performing narrowly defined tasks, future AI models may understand and interpret visual information in a more flexible and autonomous way.
2. **Enhanced AI-Powered Perception:** Computer vision is advancing beyond basic image recognition to more complex perceptual tasks, such as understanding context, predicting actions, and processing multi-modal data (integrating vision with audio, text, and other inputs). This will enable AI to make more sophisticated decisions.
3. **Real-Time and Edge Processing:** A major shift is occurring toward processing visual data in real-time on edge devices like smartphones, autonomous vehicles, and IoT sensors. This reduces dependency on cloud computing, lowers latency, and enables faster decision-making, making AI-powered systems more efficient.

4. **Advancements in 3D Vision and Spatial Understanding:** The future of computer vision includes improvements in 3D scene reconstruction, depth perception, and spatial awareness, which are crucial for applications like augmented reality (AR), robotics, and autonomous navigation.
5. **Ethical Considerations and Bias Reduction:** As AI systems become more integrated into society, there is a growing need to address issues like bias in training data, privacy concerns, and the ethical implications of surveillance and facial recognition technologies. Efforts to develop fair, transparent, and accountable AI models will be critical.[22]

Chapter 2: State of the Art

2.1 Introduction:

This chapter presents a detailed review of the state of the art in road lane detection. We first discuss classical approaches, before analysing methods based on artificial intelligence. The objective is to identify the strengths and weaknesses of each approach in order to justify the methodology chosen in our project.

2.1.1 Overview of lane detection in autonomous vehicles:

Lane detection is a technique for automatically detecting road markers to ensure that cars stay in their assigned lane and do not collide with vehicles in other lanes. Lane detection plays a big role in autonomous driving as it displays information specific to the geometrical features of lane line structures to the vehicle's intelligent system to show the position of lane markings. This allows the autonomous vehicle to make multiple decisions and judgments about its location and state and to ensure safe driving.

Researchers in this field have looked into different methods, with the input dataset being one of two types: self-collected or acquired from an online public dataset. The most common methodologies used were geometric modelling,

traditional methods and AI-based approaches, including deep learning and machine learning. Over the past four years, there has been an increase in research on the use of deep learning. While some studies have used stand-alone deep learning implementations, others have concentrated on combining deep learning with other machine learning approaches and traditional methodology.

Lane detection is a challenging task due to the variety of lane marking styles, varying road conditions, and the fact that lane lines are relatively thin. Detection accuracy is about 96% in stable conditions and greatly drops to an all-time low in conditions of ongoing heavy rainfall. The rainfall or external issues such as rain, shadows, and brightness, and internal issues such as worn down or uneven lane markings prevent reliable detection. And what makes lane detection even more complex is the variations in lane patterns, road curvature, and lighting conditions, especially at night or during rain. [23]

2.1.2 Importance of accurate lane detection for autonomous vehicles:

According to a World Health Organization (WHO) report published in June 2022, approximately 1.3 million people die yearly from road traffic accidents. As a human driver, it is hard to remain in the correct lane. Therefore, the costs of road traffic accidents to society are expensive in terms of human injury and economic loss. As a result, The essential goal of the development system in an autonomous vehicle is to assist drivers in identifying driving risks and ensuring extra safety and comfort for the driver and passengers in the car. One of the crucial systems in autonomous vehicles is The Advanced Driver Assistance System (ADAS) which aims to reduce driver error by helping to avoid vehicle collisions, increase traffic efficiency, and enhance transportation development. [23]

2.2 Traditional Lane Detection Methods

2.2.1 Geometric and classical approaches:

Lane detection systems in traditional methods depend on established rules and structured processing stages to detect lane markings. These approaches use image pre-processing to reduce noise from the image followed by feature extraction to spot lane-like areas and then model fitting to estimate lane boundaries. Once the

lanes are identified, Line tracking techniques enable consistent lane detection across frames. The most frequent techniques employed include:

1. Image pre-processing:

Before starting any processing stages it is common to first filter the image using different image pre-processing methods to improve clarity and remove unnecessary details. The most pre-processing used tools are the **Region of Interest (ROI) Extraction, vanishing point detection, Image Augmentation, grayscale conversion and thresholding**. First, **Region of Interest (ROI) Extraction** aims to delete the extra details from the image by only looking in the lower part of the image to reduce noise although it can still be affected by shadows and vehicle obstructions. **Vanishing point detection** has also been studied to establish the ROI. **Image Augmentation** also aims to provide the lane line information with filtering out the non-lane information. The **grayscale conversion** main goal is to reduce computational demands by concentrating on intensity shifts over color data. Finally, the **thresholding** techniques with both **Adaptive thresholding** and **Otsu's thresholding** manages to optimize segmentation by separating lane markings off the image background. By using these methods it would assist lane detection systems to provide a more precise and accurate outcome in different scenarios.

2. Feature extraction:

Feature extraction is the most important task in the lane detection process focusing on identifying and isolating lane-specific patterns from an image. One of the fundamental techniques used in the process is **Inverse Perspective Mapping (IPM) / Perspective Transform**, which eliminates the perspective distortion and gives a better view of the road with lane structures appearing more clearly. One of these common methods applied in this regard is **Thresholding**, which tries to distinguish lane markings from the background based on pixel intensity differences. Some of these methods include **Symmetrical Local Threshold (SLT), Segmentation Threshold, Adaptive Threshold**, and **Otsu's Threshold**, all of which help to enhance the visibility of lanes by effectively segmenting lane lines. **Quadric Methods** are also used to obtain the precise modelling of the detected lanes shape in a way that it is represented with greater precision. Numerous **Filtering Techniques**, including **Gaussian Filter, Mean Filter, Median Filter,**

Anisotropic Diffusion Filter, Fast Neighborhood Reduction Filter (FNRF), Wavelet Transform, Bilateral Filter, and Interacting Multiple Model Data Association (IPDA), are used to remove noise and smooth the lane features extracted. In addition, **Color and Edge-Based Detection** like **Canny Edge Detector, Sobel Edge Detector, and Laplacian Edge Detector** are applied to identify lane edges and enhance edge contrast. To maximize the precision of lane detection, **clustering algorithms** like **Density-Based Spatial Clustering (DBSCAN)** and **K-Means Clustering** are applied to group similar lane pixels. Methods such as **Sliding Window and Neighborhood-Based Feature Points** facilitate lane marking tracking between consecutive frames to provide stable detection. Finally, **Morphological and Heterogeneous Operations** are applied to post-process lane boundaries by eliminating unnecessary noise and improving lane continuity. Such feature extraction techniques allow lane detection systems to better analyse and detect road markings in various driving environments.

3. Lane model fitting:

Lane model fitting has a big role in lane detection as it approximate the shape and structure of lane lines based on extracted features. The common model fitting methods include **Line Segment Detector (LSD)** which is used for straight lane segments and then gives a structured representation of lane boundaries. Numerous **Fitting Techniques B-spline, Quadratic, Polynomial, Parabolic, Dynamic Programming, Lost Square and Bresenham Line Writing Space (BLWS)** that were employed to model the lane shapes and are convenient for both straight and curved road scenarios. **Vanishing Point Detection** is also used for determining the perspective and orientation of the road. At the same time **Waveforms, Contour Analysis and the Harmony Search Algorithm (HSA)** refine lane shape approximation by analysing the structural patterns of lane lines. Furthermore, **Contrast Limited Adaptive Histogram Equalization (CLAHE)** is applied to enrich the lane contrast and help the lane boundaries become more visible in different lighting conditions. In addition, to keep the stability of the model robustness under noisy environment the **Random Sample Consensus (RANSAC)** is employed, as it eliminate outliers, thereby improving the precision of lane model estimation. Another technique is the **Graph-Based Approach** is chosen to transform the lane features to a structured framework, -enhancing the reliability of

lane detection. Moreover, the **Seed Fill Algorithm** is implemented to pick out and fill the lane areas, a measure that significantly improves segmentation and lane boundary identification. What's more is the **Kanade-Lucas-Tomasi (KLT) Tracker**, **Lucas-Kanade Optical Flow** and **Model Predictive Control (MPC)** which are a real-time lane tracking techniques capable of tracing lane points from one frame to another. Whereas the **Model Predictive Control (MPC)** is responsible for the prediction of the position of a vehicle and then modifies the path of the lane accordingly. Also, **Neural Network-Based Real-Time Lane Detection (SUPRL)** exploits the power of deep learning to track the lane continuously. And **Linear Regression** is applied to fit lane features to a straight-line model when necessary. By leveraging these lane model fitting techniques, lane detection systems can accurately estimate lane shapes, track lane boundaries in real-time, and ensure robust lane detection even under challenging driving conditions.

4. Line tracking:

The line tracking process is important in maintaining lane continuity from one frame to the subsequent one to ensure the long-term stability and correctness of lane detection. A very common practice in this direction is the application of the **Kalman Filter** that allows lane positions to be predicted and adjusted through lane line movement estimation from previous frames. This approach allows consistent tracking to take place despite lane markings being occluded or lost temporarily through changes in environmental conditions. Another important element to this approach is **Lane Classification** that classifies lane markings into solid and dashed lanes on the basis of qualitative characteristics. The classification into lane types increases the accuracy of the approach and allows the system to respond effectively to ever-changing driving environments on the roads. Another high-quality line tracking technique employed is the **Parabola Equation-Based Tracking** that effectively models and detects curved lanes by curve fitting lane points through a parabolic equation. As a result of this approach, there is high accuracy in lane tracking with curved lane segments and the system becomes more adapted to real driving conditions. Thus, these line tracking techniques work in tandem to maintain stability and correctness to lane detection through continuity of lanes, error rejection, and uninterrupted tracking in dynamic environments. [23]

2.2.2 Limitations of traditional methods:

- Geometric modelling/conventional approaches need to manually adjust the parameters, although this procedure is efficient and uncomplicated. Furthermore, previous methods based on handcrafted features to detect lanes are limited in scenarios using edge, texture, or color information, which requires complicated post-processing modules to perform. Likewise, in many complex scenarios, these approaches function inadequately.
- Traditional computer vision (CV) techniques are time-consuming and computationally demanding and rely on complicated algorithms to analyse the delicate aspects of lane images.
- Unfavourable weather, such as rain, will impact lane detection. Likewise, insufficient lighting and a night setting will produce poor results. However, there are yet no practical solutions for dealing with such issues. As a result, conventional approaches are ineffective in detecting lanes in complex traffic situations.
- The number of lanes is frequently not fixed, and techniques for detecting lanes are sometimes incorrect. Straight lines, for example, are treated as lanes by Hough transform-based algorithms, which may cause street lamps to be mistaken for lanes.
- Traditional methods need a consistent framework for detecting various scenes and the inefficiency of using images due to label noise.
- Traditional approaches work in a controlled environment and face numerous problems regarding robustness difficulties caused by road scene fluctuations. Furthermore, occlusion from overtaking vehicles or other objects and excessive illumination make lane identification and tracking difficult. [23]

2.3 AI-Based Lane Detection Approaches

2.3.1 Machine Learning-Based Methods:

Unlike traditional techniques that rely on predefined rules and handcrafted features, Machine learning based methods use statistical models to learn pattern from available data. Several lane detection experiments have implemented **Bayesian Classifiers, Haar Cascades, Extreme Learning Machines (ELM),**

Support Vector Machines (SVM), and Artificial Neural Networks (ANN). One strategy consists of **Bayesian learning**, where a **Cascaded Dempster-Shafer Combination Rule** is applied to process data and improve lane estimation. To improve precision, morphological procedures such as **erosion and dilation** are used for removing outliers. Additionally, **SVM models with Radial Basis Function (RBF) kernels** have been tested to build a robust framework for detecting lane markings. ML-based methods have also been applied to address challenging road conditions. For example, researchers have developed a **hierarchical Bayesian model** for estimating multi-hyperbola parameters to identify curved lanes under rough conditions. Furthermore, ML techniques have been combined with deep learning models, such as **SegNet**, to refine lane segmentation, though segmentation uncertainties remain a challenge. [23]

2.3.2 Deep Learning-Based Methods:

Deep learning solutions have enhanced lane detection through **Convolutional Neural Networks (CNNs) and Fully Convolutional Networks (FCNs)** used for feature extraction and segmentation. Unlike traditional or machine learning-based techniques, deep learning models **autonomously learn lane structure** via large datasets, thereby making such detections less sensitive to light, weather, and road variability. The most prominent architecture used is **CNN**, capable of learning various lane pixel features and predicting lane markings accordingly. Researchers have optimized CNN models using the **S-Shaped Binary Butterfly Optimization Algorithm (SBBOA)** which aids hyper parameter tuning and lane detection efficacy in the end. In addition, Fully Convolutional Networks such as **SegNet, U-Net, ResNet, and ResUNet++** have been benchmarked for lane detection and found that **ResUNet++ has the best performance** in challenging environments. Furthermore, where uncertainty exists for segmentation, **statistical techniques in deep learning** have been proposed, applying **Bayesian Theorem** to increase reliability for segmentation. In addition, **Recurrent Neural Networks (RNNs)** have been integrated into CNN-based models to capture **temporal dependencies**, improving lane detection consistency in continuous driving scenes. An additional improvement stepping into deep learning-based lane detection comes from the introduction of **attention mechanisms**, which allow systems to pay attention to lane-specific information with less focus on irrelevant information.

Therefore, this has led to better segment lane markings when utilized in complex driving situations. Finally, Hybrid models that fuse **deep learning techniques with geometric modelling** have also been found to be more robust in complex situations. These approaches apply **deep feature extraction and geometric constraints** for more effective lane tracking despite difficult road conditions. [23]

2.4 Benchmark datasets for lane detection:

- **TuSimple Dataset:**

- The TuSimple dataset is a publicly available traffic-detection data set with light traffic and clear lane markings and the dataset is unique because it annotates complete lane boundaries instead of lane markings.
- The dataset contains almost 7000 video segments. Each video clip comprises twenty frames in total and the last frame of each series is labelled with lane annotations.
- Seventy percent of the videos are used for learning in the network, twenty percent for validation, and 105 for testing.
- The annotations are JSON format, showing the lanes' x-position at different discretized y-positions.
- It provides a variety of weather scenarios and collected during the daylight in highways and is a massive dataset for measuring the lane detection performance.
- It collects data from roads with two lanes/three lanes/or more lanes... clear lane lines with excellent image quality, no blur, and relatively simple identification challenges.

- **KITTI dataset:**

- The KITTI benchmark is also popular data for road scenes. It contains various information regarding the road scene, including color pictures, stereo images, and laser point data.
- KITTI dataset is divided into 50% training and 50% testing. Where 289 for training and 290 test images in the road and lane estimate benchmark.
- Ground truth was created in the KITTI dataset by manually annotating the images.

- The KITTI road has two sorts of annotations: road segmentation, which covers all lanes, and ego-lane, which designates the lane in which the car is presently moving.
- **CALTECH dataset:**
 - The Caltech Lanes dataset is the second most used dataset for lane detection.
 - It has four video sequences (or sub-datasets) in urban streets, both straight and curved, totalling 1225 images.
 - The resolution of each video clip is in 640 x 480 pixels and includes varying lighting and illumination situations, lane markings, sun glint, pavement types, shadows, crosswalks, and congested environments.
- **CITYSCAPES dataset:**
 - The focus of this literature is on the semantic comprehension of urban street sceneries from the perspective of a car.
 - CITYSCAPES is well-known for high-resolution images and finely labelled training data.
 - The data was made for semantic segmentation tasks but not specified for lane detection.
 - The collection contains 5000 photos with high-quality pixel-level annotations with 2975 training datasets, 500 validation datasets, and 1525 test datasets.
- **APOLLOSCAPE dataset:**
 - It's a vast data collection with about 19040 photos with training sets are 12400, validation sets are 3320, and test sets are 3320, respectively.
 - APOLLOSCAPE has six separating markings, four guiding markings, two stopping lines, 12 turning markings, and other pixel-level lane markings and lane characteristics.
 - It also has some additional markings like a stopping line, a zebra line, a single solid/dashed line, a double solid line, and other semantic segmentation information that can be found on the road.
 - This dataset is challenging to work with because it includes high-quality pixel-level ground truth of over 110,000 frames.
- **CULANE dataset:**

- It is a simple dataset collected during the daytime along highways in excellent or moderate weather.
- CULane features various challenging driving circumstances, including congested roads or highways with low lighting and frames that lack lane markers. As a result, it is rarely preferred by researchers for detecting the lane.
- The dataset is distributed in different ways: to 60% training, 10% validation and 30% testing / 65% training, 10% validation and 25% testing / 75% training and 25% testing set.
- Limitations: For basic lane detecting applications, it is less frequently utilized because it is more difficult than TuSimple. [23]

2.5 Evaluation Metrics in Lane Detection:

Various performance indicators are available, but the most frequent are accuracy, precision, F-score, and receiver operating characteristic (ROC) curves. The accuracy rate should indicate the algorithms global output if the dataset is balanced. The accuracy demonstrates the accuracy of optimistic predictions. The lesser the amount of ‘false alarms,’ the higher the accuracy. The recall, also known as the true positive rate (TPR), is the proportion of positive cases that the algorithm accurately detects. As a result, the better the recall, the more accurate the algorithm finds positive instances. The F1 score is the harmonic mean of Precision and Recall, and because they are merged into a single metric, it may be used to compare algorithms. The harmonic mean is employed instead of arithmetic since it is more sensitive to low values. As a result, if an algorithm is accurate and has a high recall, it has a decent F1 score. These parameters can be calculated as individual metrics for each class or overall metrics for the algorithm [23]

2.6 Challenges and Open Issues in Lane Detection:

- **Imbalanced Dataset Problem:**

Extremely imbalanced data set problem because the backdrop class contains most of the lane pixels in the image. The amount of backdrop pixels is significantly more than the number of lane pixels due to the lane’s slenderness. It may take time to pick up on such characteristics. Aside from unbalanced data, the quality of acquired data and annotations also restricts the capacity of various methods. To

address these issues, advanced techniques such as transfer learning and attention mechanisms can be implemented. Also, a more generic dataset that replicates real-world road conditions can be investigated.

- **Variation and Changeable Lane Markings:**

The wide diversity of lane markers, the complex and changing road circumstances such as no line, shadows, occlusion and adverse weather conditions such as rain, fog, and snow provide few or no visible lane lines, the inconsistency of the lanes, the curvature of the lane, and the varied lane pattern make detection much more difficult. Since traditional methods work in a controlled environment only, they face robustness difficulties caused by road scene fluctuations. Furthermore, occlusion from overtaking vehicles or other objects and excessive illumination make lane identification and tracking difficult. Additionally, the performance of lane detection and tracking algorithms suffers due to these issues in lane recognition and tracking.

- **Limitations of Deep Learning Models:**

Advanced detecting algorithms such as deep learning have outperformed traditional methods in complex scenarios, but they have limitations. For example, despite the importance of multilane detection, only a limited number of lanes can be detected, and the cost of detection time is frequently prohibitive. Therefore, various factors influence lane detection tasks, including specific complex traffic scenarios. Attention mechanisms have tremendously improved feature extraction in NLP and CV. The employment of an attention mechanism improves feature localization in the feature map and eliminates the need for post-processing. Therefore, as lanes are long and thin for lane detection, there are considerably fewer annotated lane pixels than background pixels, which is challenging for a model to learn.

- **Issues Related to Real-World Driving Conditions:**

The biggest concern for ADAS is that lane detection must work in a diverse driving situations, from city streets to rural roads to day to night to rain to snow. Generalization has become a concern, for it was found that models trained on one dataset fail when used in another, since they're not able to detect what's going on in unexpected typical driving situations. [23]

2.7 Conclusion:

In conclusion, traditional approaches have significant limitations in difficult conditions (bad weather or lighting variations). On the other hand, methods based on deep learning, offer promising performance and robustness facing complex conditions although however, they demand high-performance hardware, such as a powerful GPU, a multi-core CPU, and ample RAM, which may not be accessible in all environments to train the module. These results support our choice to go for a traditional based approach, detailed in the next chapter.

Chapter 3: Methodology

3.1 Introduction:

This chapter presents the methodology adopted in our project which aim to improve road lane detection in difficult conditions. We describe the chosen approach and dataset, data pre-processing steps, the choice and configuration of the method used, as well as the experimental protocol implemented. Finally, we present the metrics selected to objectively evaluate the performance of the proposed model.

3.2 Introduction to chosen method and dataset

3.2.1 Overview of the chosen method:

For my project I decided to go with the Hough Transform alongside some other image processing and feature extraction techniques. Hough Transform is a computer vision technique used for feature extraction to detect shapes like lines and circles in an image. It converts these shapes into mathematical representations in parameter space, making it easier to identify them even if they're broken or obscured. This method is valuable for image analysis, pattern recognition, and object detection. It uses a voting mechanism to identify bad examples of objects inside a given class of forms. This voting mechanism is carried out in parameter space. First, the HT algorithm produces object candidates as local maxima in an accumulator space. [13]

3.2.2 Justification of the chosen method:

The reason of choosing the Hough Transform method is mainly because of these four points:

- **Robustness:** It can detect shapes in images even if they are partially obscured or degraded.
- **Simplicity:** Conceptually straightforward, making it relatively easy to implement and understand. Unlike deep learning approaches which would consume a lot of time for training the model.
- **Flexibility:** Can be adapted to detect various geometric shapes, not just lines but also circles and ellipses.
- **Noisy Global Detection:** Able to identify lines and shapes across the entire image rather than just in localized regions. [13]
- **Speed:** The method is computationally efficient and can deliver quick results, making it well-suited for real-time or near-real-time applications.

3.2.3 Justification of the chosen dataset:

The dataset implemented for this project is the **TUSimple dataset**, as it contains real-world highway driving scenarios which offer a good foundation for academic research on traffic lane detection. It is collected by camera mounted on a car driving in US highways. It has 6408 one-second-long video clips of 20 frames each divided into 3626 clips for training and 2782 dedicated to testing purposes. Furthermore, the dataset also provides an evaluation script with an annotated JSON file to evaluate the module but it is more designated for models that returns 4 lane markings.

3.3 The implemented processing steps

Before presenting the data processing steps, it is important to provide a general idea how the model is structured and used. Our model provides four different operations to detect lanes on: single image, single documents with different scenarios, clips from a structured dataset with an at least a dozen of frames each and finally on a video.

The following procedures were implemented on the data to ensure it was ready to be utilized by the model:

1. Image loading and defining output locations:

We first start by opening the files or documents which contain images:

For testing the model on a single image, we upload a random image from the **TuSimple dataset** (which is locally stored) by using the **mpimg.imread ()** function that comes from the **matplotlib.image** module. This function is used for reading an image from a file and converting it to a NumPy array in RGB format. The variable **one_image** will then contain the uploaded image as a 3D NumPy array with the shape (Height, Width, 3), where 3 represents the RGB color channels. The image is then ready to be used for subsequent pre-processing and lane detection steps.

```
#uploading a single image  
one_image = mpimg.imread("C:/Users/mosta/Desktop/TUSimple/clips/0601/1494452389590241875/4.jpg")
```

For testing the model on multiple different images stored in a single folder, we define an input path using the variable **input_one_folder**, which stores the location of the folder containing images from the TuSimple dataset and we also define an output path using the variable **output_one_folder**, which provide the location of where to store the results, if the output folder does not exist we will use the **os.makedirs()** function to create one on the same spot where it was supposed to be located to ensure that the output path is ready to be used before any files are written to it, and we would put the parameter **exist_ok** to true to prevent showing an error if it finds the folder in that location. After preparing the paths we will go through a **for** loop to extract each file as **filename** from the input directory using **os.listdir ()** which returns a list of all the files names present in the **input_one_folder**, then we will check if the file is an image by using **filename.endswith ()** function where it checks if the file type is either in jpg or png format. If it is we will extract that image full path by merging the input folder path with the file name to **filepath** using the **os.path.join ()** function then we will read the image with an OpenCV tool called **cv2.imread ()**, and since it reads the image by default in a BGR format we will need to convert it using **cv2.cvtColor ()** to a RGB format so it can be ready for further processing.

```
#uploading input of a single folder and the results output location
input_one_folder = 'C:/Users/mosta/Desktop/Hough-transform/InputForOneFolder'
output_one_folder = 'C:/Users/mosta/Desktop/Hough-transform/OutputForOneFolder'
# Create the output folder if it doesn't exist
os.makedirs(output_one_folder, exist_ok=True)

for filename in os.listdir(input_one_folder):
    if filename.endswith((".jpg", ".png")):
        filepath = os.path.join(input_one_folder, filename)
        image = cv2.imread(filepath)
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

For testing the model on a structured dataset or on a folder containing multiple consecutive frames from a video clip, we define an input path using the variable **input_root**, which stores the location of the folder containing images from the TuSimple test set and we also define an output path using the variable **output_root**, which provide the location of where to store the results and as we did before we will check the output folder if it exists or if we need to create one using the **os.makedirs()** function. We will also define the **batch_size** where it indicates the number of images we will take from each folder to combine them together. Next step is to extract the images from the input root which is more complex in structure so we need to use the **os.walk ()** function that will allow us to access the whole directory tree to find the images we are looking for. This function returns a generator that gives back one result at a time containing the current **root**, a list of subdirectories (**dirs**), and a list of **files** in that folder. The condition **if not dirs:** ensures that only the folders without subdirectories are processed, assuming these contain the actual image files. Within each valid folder, the code constructs a list called **image_files**, which contains the full paths to all .jpg and .png files inside the root. The filenames are filtered using **.endswith ()** and normalized to lowercase using **.lower ()** to ensure case-insensitive matching. Finally, the list is sorted to maintain consistent processing order. Then we check if the images that are in the folder sufficient for a batch size if not we show a warning that we will skip this folder. To load the images we loop through the **image_files** list from the first image until the **batch_size** that we need from the folder and read each image and check if it exists then convert it to RGB format and add it to **images** variable for further processing.

```

# Parameters
#uploading input of the TUSimple clips folder and the results output location
input_root = 'C:/Users/mosta/Desktop/Hough-transform/Input'
output_root = 'C:/Users/mosta/Desktop/Hough-transform/Output-batch10-v1'
batch_size = 10
os.makedirs(output_root, exist_ok=True)

# Traverse all subfolders
for root, dirs, files in os.walk(input_root):
    if not dirs: # Only process actual image folders (not the root itself)
        image_files = sorted([
            os.path.join(root, f) for f in files
            if f.lower().endswith(('.jpg', '.png'))
        ])

        if len(image_files) < batch_size:
            print(f"Skipping {root}: found only {len(image_files)} images.")
            continue

        # Load and average the batch
        images = []
        for img_path in image_files[:batch_size]:
            img = cv2.imread(img_path)
            if img is not None:
                img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                images.append(img_rgb)

```

For testing the model on a video clip we need to get the video using **VideoFileClip** function from the **moviepy.editor** module providing it with the path to the input video. We also define an output path using the variable **white_output**, which provide the location of where to store and name the result video.

```

#Uploading the video
clip1 = VideoFileClip('C:/Users/mosta/Downloads/solidWhiteRight.mp4')
white_output = 'C:/Users/mosta/Desktop/solidWhiteRight_output.mp4'

```

2. Image enhancement:

As for this image pre-processing part we apply different image enhancement techniques first we convert the images to grayscale using the OpenCV convertor **cv2.cvtColor()** as shown in (figure 1, figure 2), then we improve the edges visibility by applying the **CLAHE (Contrast Limited Adaptive Histogram Equalization)** technique by **createCLAHE ()** then **clahe.apply ()** which improves local contrast and helps highlight lane lines more clearly, especially under poor or uneven lighting conditions as shown in (figure 3, figure 4). Concerning the parameters **cliplimit=2.0** is used to define the degree of contrast used, a higher value means more details which may cause us unwanted noise. Values between 2.0 and 4.0 are common for lane detection tasks, and **tileGridSize= (8, 8)** divides the image into small blocks so the function can apply contrast on each block individually.

```
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
enhanced_gray = clahe.apply(gray_image)
```

Before applying grayscale



After applying grayscale



Figure 1: Before we apply the grayscale on the image

Figure 2: After applying the grayscale on the image

Before applying CLAHE



After applying CLAHE



Figure 3: Before we apply CLAHE on the image

Figure 4: After applying CLAHE on the image

To apply the model on a structured dataset we take the average of multiple frames from the same folder so we can feed it to the model instead of passing each image at a time this helps finding patterns in successive images such as lane lines and also helps removing unimportant noise which can be really helpful against obstacles that would prevent us from obtaining lane lines such as faded lanes, poor lighting and bad weather (as shown in figure 5, figure 6). This process is done by first using **np.stack ()** to put all the images we want to combine in a single 4D array and we convert the values of it to float using **.astype (np.float32)** to help us calculate the mean correctly when using **np.mean ()** to give us the average pixel-wise value across all images and then the **np.uint8 ()** converts the final result back into an 8-bit unsigned integer format (0–255), suitable for saving or displaying as a regular image.

```
averaged = np.uint8(np.mean(np.stack(images, axis=0).astype(np.float32), axis=0))
result = pipeline(averaged, True, 100, 200)
```



Figure 5: This shows one of the frames that will be Combined together Figure 6: Final result after combining all the consecutive frames

The last image enhancement step is to the images that are used in the model but not averaged (single image or images from a single document with different road scenes scenarios) that needs to be smoothed and for that we apply the **Bilateral Filtering** to reduce noise while preserving edge details, **d=9** means the pixel neighborhood used during filtering, **sigmaColor= 75** controls how much influence pixels with different intensities have, **sigmaSpace= 75** defines how far apart pixels can be and still influence each other. The visual difference between the non-filtered image and the filtered one may not be noticeable by the naked eye (figure 7, figure 8). However, it still makes a big effect when later applied in the Canny detector (as shown in figure 9).

```
if averaged:
    cannyed_image = cv2.Canny(enhanced_gray, min_thrs, max_thrs)
else:
    filtered_image = cv2.bilateralFilter(enhanced_gray, d=9, sigmaColor=75, sigmaSpace=75)
    cannyed_image = cv2.Canny(filtered_image, min_thrs, max_thrs)
```



Figure 7: Before applying BilateralFilter on the image Figure 8: After applying BilateralFilter on the image

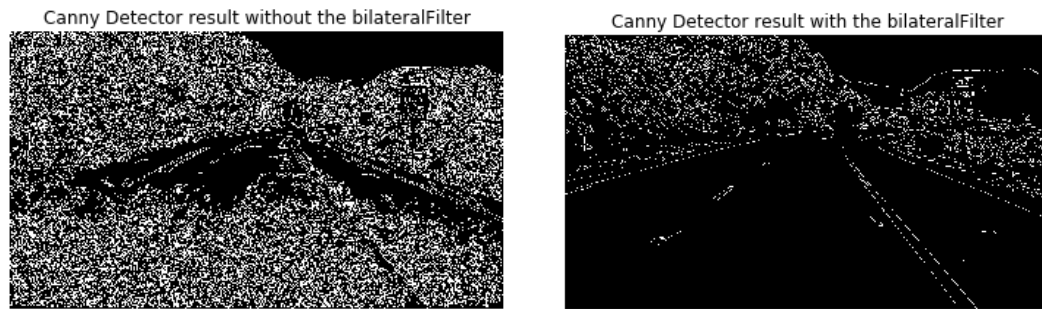


Figure 9: The effect of the BilateralFilter in the Canny Edge Detector

3. Edge detection:

After the images have been pre-processed using image enhancement techniques such as CLAHE and bilateral filtering to reduce noise and improve contrast now they are ready to be processed. We are going to extract edges using the **Canny Edge Detector** this OpenCV technique works in multiple stages as it first applies Gaussian blur to reduce the even more unwanted noise then it calculates the intensity gradients of the image which indicate how strong an edge is and in which direction it points, then the Non-Maximum Suppression (NMS) is used to thin out the edges, leaving behind only the most significant edge pixels by using the thresholding values to identify them where pixels with gradient magnitude greater than **max_thrs** are considered strong edges and are kept in the final output, and pixels with gradient magnitude lower than **min_thrs** are removed entirely. The values between **min_thrs** and **max_thrs** are considered weak edges so it then uses edge tracking by hysteresis where it checks if the weak pixel is connected (in 8-connected neighbourhood) to a strong edge and it will be kept otherwise the detector will consider it as noise and remove it. (Figure 10)

```
if averaged:
    cannyed_image = cv2.Canny(enhanced_gray, min_thrs, max_thrs)
else:
    filtered_image = cv2.bilateralFilter(enhanced_gray, d=9, sigmaColor=75, sigmaSpace=75)
    cannyed_image = cv2.Canny(filtered_image, min_thrs, max_thrs)
```

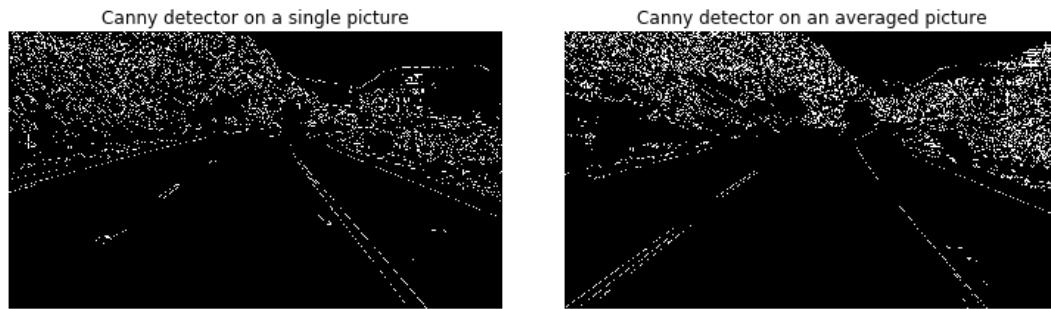


Figure 10: the result of the Canny Detector on both a single image and on an averaged one

4. Region of Interest (ROI):

Before going to extract the lane lines from the detected edges we are going to crop the **cannyed_image** to only focus on the lane lines and remove irrelevant parts of the image where we can find the sky or other vehicles on the road or just an unnecessary background. To apply it first we define a triangle that begins at the bottom left corner of the image, proceeds to the centre of the image at the horizon, and then follows another edge to the bottom right corner of the image and store it in **region_of_interest_vertices** variable.

```
region_of_interest_vertices = [
    (0, height),
    (width / 2, height / 4),
    (width, height),
]
```

Now we can crop the image using those vertices by calling the **region_of_interest** function, where this function first creates a mask — a black image with the same size as the original picture we are going to crop, which in this case is the **cannyed_image**. Then, we fill the area we want to keep in white on the mask using **cv2.fillPoly ()**. After that, we apply the mask to the image using **cv2.bitwise_and ()**, which provides us with a final result where the white part of the image is kept and the rest is blacked out. (Figure 11)

```

cropped_image = region_of_interest(
    cannyed_image,
    np.array(
        [region_of_interest_vertices],
        np.int32
    ),
)

```

```

def region_of_interest(img, vertices):
    mask = np.zeros_like(img)
    match_mask_color = 255
    cv2.fillPoly(mask, vertices, match_mask_color)
    masked_image = cv2.bitwise_and(img, mask)
    return masked_image

```

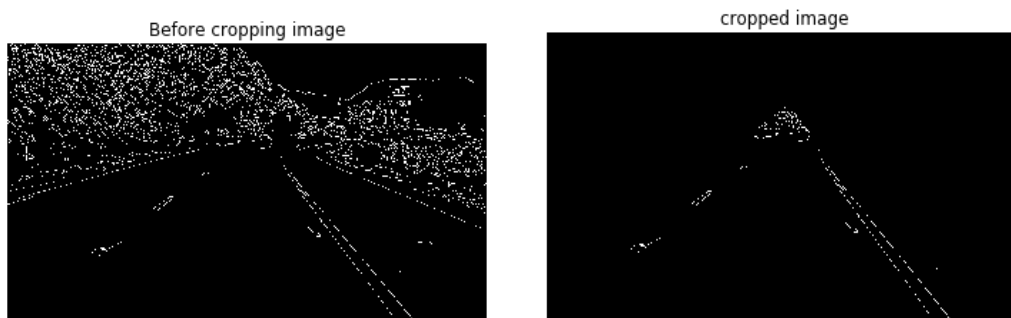


Figure 11: the result of the `region_of_interest` function

5. Lane detection:

To detect lane lines we will use Probabilistic Hough Transform. This technique is implemented through the OpenCV function `cv2.HoughLinesP()` which scans the edge image for points that form straight lines. We will give the function a **cropped_image** with a parameters like **rho=6** which means the algorithm will consider distances in steps of 6 pixels when voting for lines, **theta=np.pi / 60** sets the angle step to 3° ($\pi/60$), **threshold=160** is the minimum number of votes needed to consider a line, **lines** is where we will be storing the detected lines, **minLineLength=40** is the minimum length of a line segment to be detected. **MaxLineGap=25** is the maximum gap between points on a line to still be considered a single line.

```
lines = cv2.HoughLinesP(  
    cropped_image,  
    rho=6,  
    theta=np.pi / 60,  
    threshold=160,  
    lines=np.array([]),  
    minLineLength=40,  
    maxLineGap=25  
)
```

6. Visualize the detected lane lines:

For this last step of the lane detection pipeline we first need to point the start and the end of the left and right detected lanes. We begin by returning the original image if the Hough Transform did not find lines on the image, then if there are lines we check if one of them is completely vertical in **if $x_1 - x_2 = 0$** and we skip it so the model does not mistake lane lines with poles or edges of buildings as the lanes usually appear as diagonal lines. After calculating the **slope** it will allow us to know if the line is for a left or a right lane but first we also want to skip the lines that has the absolute value of **slope** less than 0.5 as they represent a too horizontal line which are not useful for lane detection. If the value of the **slope** calculated for each line is negative then it's a left line otherwise we put the line in the right line list.

```

lines = cv2.HoughLinesP(
    cropped_image,
    rho=6,
    theta=np.pi / 60,
    threshold=160,
    lines=np.array([]),
    minLineLength=40,
    maxLineGap=25
)

left_line_x = []
left_line_y = []
right_line_x = []
right_line_y = []
if lines is None:
    print("No lines detected by Hough Transform.")
    return image
for line in lines:
    for x1, y1, x2, y2 in line:
        if x2 - x1 == 0:
            continue
        slope = (y2 - y1) / (x2 - x1)
        if math.fabs(slope) < 0.5:
            continue
        if slope <= 0:
            left_line_x.extend([x1, x2])
            left_line_y.extend([y1, y2])
        else:
            right_line_x.extend([x1, x2])
            right_line_y.extend([y1, y2])

```

Now that we separated all the lines that could represent the left and right lanes, if there were no lines valid detected we will return the original image otherwise we will define where we want the start and the end of the drawn lane on the Y axis where **max_y** is the beginning of the line starting from the bottom of the picture and **min_y** is the end of the lane where we will give it the length of 40% of the height of the original image. Next we will put the separated lines all together to get as a result one left line and one right line, to do that we are going to use **np.polyfit ()** to calculate the coefficients of the line using the x and y coordinates ($x = m y + b$) and then we will make it a function so we can use it to calculate the x values for the start and the end of the line with the **min_y** and **max_y** we defined using **np.poly1d ()**.

```

if left_line_x and left_line_y and right_line_x and right_line_y:
    min_y = int(image.shape[0] * (3 / 5))
    max_y = int(image.shape[0])
    poly_left = np.poly1d(np.polyfit(
        left_line_y,
        left_line_x,
        deg=1
    ))

    left_x_start = int(poly_left(max_y))
    left_x_end = int(poly_left(min_y))

    poly_right = np.poly1d(np.polyfit(
        right_line_y,
        right_line_x,
        deg=1
    ))

    right_x_start = int(poly_right(max_y))
    right_x_end = int(poly_right(min_y))

```

And now that we have the start and the end of both the left and right lines we are going to check if the distance between them is at least 300 pixels so we can consider it a valid lane then we can print the two lines on the picture using **draw_lines ()** function that needs the original image and the coordinates of the start and the end of both lines as parameters.

And finally we will return the image with lines drawn on it as the result of the pipeline.

```

lane_width = abs(right_x_start - left_x_start)
if lane_width < 300:
    print(f"No valid lines detected.")
    return image
line_image = draw_lines(
    image,
    [[
        [left_x_start, max_y, left_x_end, min_y],
        [right_x_start, max_y, right_x_end, min_y],
    ]]
)

else:
    # Fallback: just return the original image if no lines were detected
    print("No valid lines detected.")
    line_image = image
    return image
return line_image

```

The **draw_lines ()** function in addition to the two parameters we explained earlier also takes a color and thickness parameters that we defined where we will give a red color and a thickness of 7 to the line we are going to draw. First we create a full black picture called **line_img** with the dimensions of the original image then

we copy the original image to preserve it and only work on a copy of it, additionally we will draw the lines on the **line_img** we created using the OpenCV function called **cv2.line ()** where it takes the coordinates of the line with the color and thickness provided. The final step is to use **cv2.addWeighted ()** function to overlay the detected lane lines onto the original image with transparency/blending. Where we will take 80% of brightness from the original image and 100% from the **line_img** which appear the lines to be more dominant on the original image.

```
def draw_lines(img, lines, color=[255, 0, 0], thickness=7):
    line_img = np.zeros(
        (
            img.shape[0],
            img.shape[1],
            3
        ),
        dtype=np.uint8
    )
    img = np.copy(img)
    if lines is None:
        return
    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(line_img, (x1, y1), (x2, y2), color, thickness)
    img = cv2.addWeighted(img, 0.8, line_img, 1.0, 0.0)
    return img
```

And that is it! All we have left to do now is to show the final result and save them to the output folder. Starting by showing the result of one image after applying the pipeline by using the **matplotlib.pyplot** module where we prepare the image using **plt.imshow ()** then we use **plt.show ()** to display it. (Figure 12)

```
#Apply the pipeline on one image
result_image = pipeline(one_image, False, 50, 150)
plt.title("final result on one image")
plt.imshow(result_image)
plt.axis("off")
plt.show()
```

final result on one image



Figure 12: Displaying the result when applying the pipeline on one image

Next, we show the results for the folder containing different road scene scenarios. Each image result will be saved in the output folder by combining the output folder path with the file name using **os.path.join ()**. We then save the image to that path in a BGR format to meet the OpenCV requirements using **cv2.imwrite ()**. After that, we display the result in the same way as before, except this time each image is titled with its file name. (Figure 13)

```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
result = pipeline(image_rgb, False, 50, 150)
out_path = os.path.join(output_one_folder, filename)
cv2.imwrite(out_path, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
```

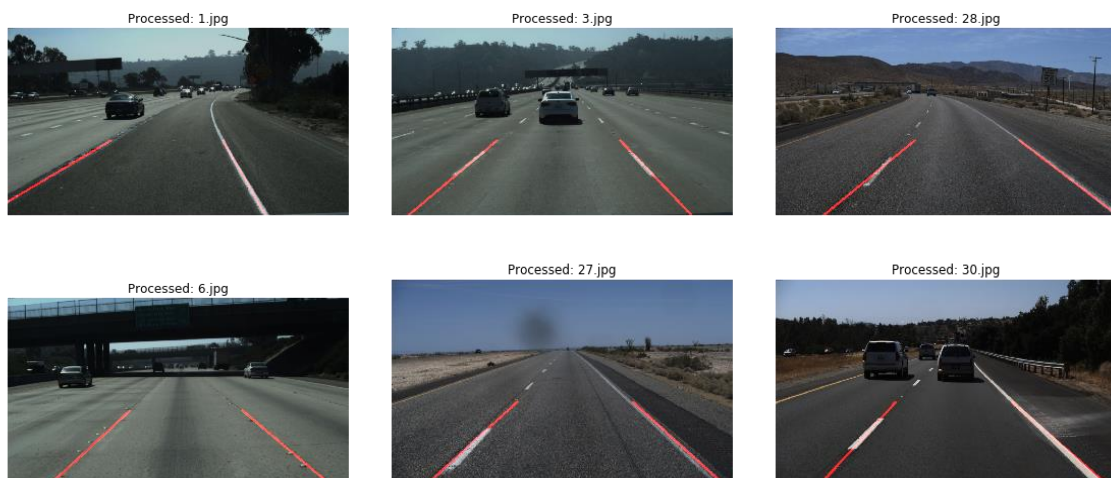


Figure 13: Displaying the results when applying the pipeline on a folder containing different road scene scenarios

Next, to show the result of a set from the TUSimple, we first create a `folder_name` variable that has the name of the input folder we applied the pipeline on using `os.path.basename(root)`. Then using the `os.path.join()` we create the result file name path and after that we write the result on that file created in BRG format to meet the OpenCV requirements and finally we print the result in the same way we did before and titling each image by its folder name. (Figure 14)

```
result = pipeline(averaged, True, 100, 200)

# Create output filename based on subfolder name
folder_name = os.path.basename(root)
output_path = os.path.join(output_root, f"{folder_name}_result.png")

# Save and show
cv2.imwrite(output_path, cv2.cvtColor(result, cv2.COLOR_RGB2BGR))
plt.imshow(result)
plt.title(f"Lane detection: {folder_name}")
plt.axis("off")
plt.show()
```



Figure 14: Displaying the results when applying the pipeline on a structured dataset

3.4 Model evaluation:

Evaluating the model is an essential step in assessing its overall performance and understanding how well it performs the target application (detecting lane lines). In this section, we will evaluate the accuracy and speed of the model when applied to images from the TUSimple dataset, which includes real-world challenges such as driving on straight or

curved lanes, and dealing with shadows and faded lane markings. This will help us identify the strengths and weaknesses of the method and determine whether it is suitable for real-time applications.

1. Accuracy of detection:

We measured our model's accuracy by counting in how many frames lane lines were detected and dividing that by the total number of frames in the test set of TuSimple. When this was applied to the test set of 2782 clips it got us an accuracy of **94.28%** where we consider a successful detection if the model produces an output with lane lines drawn on the frame so we had 2623/2782 success rate meaning that the method can provide very promising results. This sets an easy-to-understand frame-wise detection accuracy metric that tells us how often the algorithm was able to deliver any meaningful output at all. The method worked well in scenes of highways with clear marked lanes. The method then faltered when the markings were faded, when vehicles blocked them, or when shadows acted upon them. This does not measure the geometric accuracy or the real positioning of the lines which have been detected but it provides a very useful indication of how practically reliable the model is. More future work, could provide more detailed evaluation metrics, and quality measurement for lane detection besides just its presence.

2. Robustness:

In order to evaluate the model's adaptability to different scenarios of road and environment conditions, testing was carried out on 600 video clips that were equally divided into 4 specific scenarios: straight lane lines, curved lane lines, faded or missing lane markings, and scenes with shadows which may affect the detection. Each category contained 150 clips mapping out much variation in reality that the model may encounter during driving. Results are summarized in **Table 1**. The model performed best on straight lane lines, achieving an accuracy of **92.67%**. This implies strong reliability in the standard highway scenario. It also handled curved lanes reasonably well, with an accuracy of **88%**. This shows that the method can handle moderate curvature available in the road.

However, the model might underperform when facing some tougher challenges like faded lanes where the accuracy drops to **72%** which shows that the algorithm could find it hard when lane lines are not particularly clear. The last scenario is handling the clips with shadows that could interfere with the detection, we find that our model shows moderate sensitivity to lighting changes and occlusions, with an accuracy of **84.93%** which we consider good enough since it's a classical approach that does not predict the lane positioning.

From these results we understand that our model perform well in ideal and moderate challenging conditions but can still struggle a bit where visual lane features are degraded or obstructed. This will help make future improvements to the model to enhance its performance even more in such scenarios.

Scenarios	straight lane	curved lane	Faded or missing lane	Shadows influence
Accuracy	92.67%	88%	72%	84.93%

Table 1: Accuracy of the Model in Different Road Scenarios.

3. Speed and efficiency:

The method was evaluated in how fast it can run as a real-time application. Average processing time per frame is about 0.10 seconds (100 milliseconds), meaning it runs at approximately 10 frames per second (FPS). It is not real-time by strict industry standards (typically 25–30 FPS), but it is fairly quick for offline analysis or semi-real-time systems, and this should be considered in view of the fact that the classical approach used has simplicity and interpretability. All evaluations were done on a [Intel Core i5-6200U CPU with 8 GB RAM] without GPU acceleration. As the method does not require large memory or computational resources, it remains a lightweight solution compared to deep learning based alternatives which often require powerful hardware and more processing time.

4. Parameter Dependence:

The performance of our model is dependent on manually tuning the parameters. Where in the Canny edge detector we have got parameters like the minimum and maximum threshold, and in the Probabilistic Hough Transform we have got parameters like minimum line length, and maximum line gap. These values are manually adjusted by trial-and-error and observing the output on sample frames to find the perfect configuration. It is a very important step as small changes in these variables can make a very big difference to the success of the model. For example, lower threshold values can make the detector catch more noise which makes it harder to detect the lane yet a higher values of the threshold might also not detect the edges of the lane lines and therefore not being able to detect them later on the Hough Transform. This manual parameter tuning makes it somewhat non-robust and non-scalable to adapt to varied real-world environments.

5. Limitations:

- The algorithm depends heavily on clear, continuous lane markings. In cases where the markings are worn or absent, the detection performance might drop.
- The method is sensitive to varying lighting conditions such as shadows or strong sunlight, which can interfere with edge detection and lead to inaccurate lane line extraction.
- The method requires careful tuning of parameters, and these may need to be adjusted manually for different environments or camera perspectives.
- Classical methods lack semantic understanding. They cannot distinguish between valid lane lines and similar visual patterns like road edges, shadows, or markings unrelated to lanes (e.g., arrows or text on the road).
- Being a purely classical approach, the method lacks the ability to learn or adapt from data, and cannot infer missing lanes or reason about the scene context.
- The algorithm often fails to detect lanes when parts of them are occluded by nearby vehicles.

3.5 The libraries applied in this project:

In this project, we used many Python libraries that helped us implement our model effectively:

1. **OpenCV:** This was the core library used for image processing tasks. It provided essential functions for operations such as reading and saving an image, creating a mask and then cropping an image with it, drawing a line and showing it on an image. It also provided us with image pre-processing techniques such as converting color formats, and applying filters on the image to reduce the noise and highlight the edges. It even provided us with the Canny Detector and the Probabilistic Hough transform we used in our model.
2. **Numpy:** This library Enabled efficient numerical operations and array handling.
Creating a black mask, copying an image and creating a polynomial function.
3. **Matplotlib:** Used for visualizing the detection results and displaying intermediate image processing steps.
4. **os:** Used to navigate through the dataset directories and load batches of images for evaluation.
5. **math:** Used to calculate the absolute value.
6. **time:** Used to calculate the average processing time of each batch.
7. **moviepy.editor:** This library was used to read a video clip and making changes to it.

3.6 Conclusion:

In this chapter, we provided a full understanding of the pipeline that was followed for lane detection when utilizing a classical computer vision approach. In particular, for all of the reasons we presented, the Hough Transform was chosen for the lane detection because it is robust, easy to use, and most importantly, explicitly designed to utilize and extract geometrically defined features such as lane-lines.

The TuSimple dataset was chosen because it gives high-quality labelled highway lane images which are ideal for developing evaluation metrics to quantify the detection accuracy of traditional lane detection methods.

In describing the steps to process the images, we included all the important components of edge detection, selecting the region of interest, and line detection. We evaluated the model based on frame-to-frame detection accuracy. The lane line detection model seemed to work well especially when there is a clearly defined road without any obstacles. However, some limitations we noted included that the detection was very sensitive to environmental conditions, and also the detection lacked a level of understanding about the surrounding scene.

Finally, we opted to describe the tools and libraries we used to put this method together and test it extensively. This should illustrate how possible a solution like this is to use with cheap and efficient technologies. We have established a strong basis for classical lane detection with this methodology, as well as a foundation from which future improvements can be made.

General Conclusion:

This work provided an in-depth review of computer vision with special consideration given to lane detection systems in autonomous vehicles. In the first chapter, we developed a theoretical basis with a general understanding of computer vision as it relates to image processing techniques, machine learning, and deep learning. We analysed how machines can perceive and understand visual information using computer vision and we also provided many examples of computer vision applications in various industries, such as autonomous vehicles, medical imaging, and industrial inspections.

In the second chapter, we described many works of lane detection systems. We distinguished between classical geometric and deep learning based approaches. This survey also discussed the evaluation benchmarks and datasets, quantitative and qualitative evaluation measures, and the current challenges of lane detection systems, including varying illumination conditions, occlusions, and lane variations. The review demonstrated that deep learning based approaches have the highest accuracies, while classical methods will continue to be useful for their lack of complexity, ability to be interpreted, and efficiency in areas of constraints.

In the third chapter, we implemented a standard lane detection system which was performed with image processing techniques such as edge detection, region masking, and the Hough Transform. The method was performed using the TuSimple dataset, and we recorded performance as frame-wise accuracy. Although the technique was simple, the system performed acceptably given good visual conditions. The system did show the following limitations: lighting and occlusion conditions, limited potential for flexibility, and manual parameter tuning.

Overall, this project saw both the potential and limitations of traditional image processing in modern computer vision tasks. While the classical methods are effective in defined scenarios, looking forward a possible integration of the methods presented here with machine learning models or temporal tracking methods could be beneficial. If successfully developed, this hybrid direction may bridge the gap between simplicity and intelligence, making it suitable for real-world applications in autonomous driving.

List of References:

- [1] Spiceworks. (2025, Jan. 7). "What Is Computer Vision? Meaning, Examples, and Applications in 2023." Spiceworks. [Online]. Available: 0_ [Accessed: Feb. 10, 2025]
- [2] GeeksforGeeks. (2025, January 30). "Computer Vision." GeeksforGeeks. [Online]. Available: <https://www.geeksforgeeks.org/computer-vision/> [Accessed: Feb. 10, 2025]
- [3] Sony Electronics Inc. (2024, May 1). "What is a pixel and how does it form a picture?" Sony Electronics Inc. [Online]. Available: <https://www.sony.com/electronics/support/articles/00342545> [Accessed: Feb. 10, 2025]
- [4] GeeksforGeeks. (2023, May 09). "Difference Between RGB, CMYK, HSV, and YIQ Color Models" GeeksforGeeks [Online]. Available: <https://www.geeksforgeeks.org/difference-between-rgb-cmyk-hsv-and-yiq-color-models/> [Accessed: Feb. 14, 2025]
- [5] 33rd Square. (2024, Dec 3). "Grayscale and RGB Format for Storing Images" 33rd Square [Online]. Available: <https://www.33rdsquare.com/grayscale-and-rgb-format-for-storing-images/> [Accessed: Feb. 14, 2025]
- [6] Lightroom-Photoshop-Tutorials.com. (2024, Aug. 29). "Pixel Dimensions" Lightroom-Photoshop-Tutorials.com [Online]. Available: https://lightroom-photoshop-tutorials.com/pixel-dimensions/#elementor-toc_heading-anchor-5 [Accessed: Feb. 15, 2025]
- [7] B. Desai, M. Paliwal, and K. K. Nagwanshi. (2022, Jun 4). "Study on Image Filtering Techniques, Algorithm and Applications" arXiv: 2207.06481 [Online]. Available: <https://arxiv.org/pdf/2207.06481> [Accessed: Feb. 21, 2025]
- [8] Conveditor. (2023). "Mastering the Art of Image File Conversion: A Comprehensive Guide." Conveditor [Online]. Available: <https://www.conveditor.com/blog/image> [Accessed: Feb. 24, 2025]
- [9] Timothy Malche. (2024, Oct. 14). "Image Resizing." Roboflow Blog [Online]. Available: <https://blog.roboflow.com/image-resizing/> [Accessed: Feb. 24, 2025]
- [10] V. Bogdan, C. Bonchis, and C. Orhei. (2019, Oct. 2). "Custom Extended Sobel Filters," arXiv: 1910.00138 [Online]. Available: <https://arxiv.org/pdf/1910.00138> [Accessed: Feb. 26, 2025]
- [11] Zhengxian Yang, Shikai Zuo, Yanxi Zhou, Jinlong He, and Jianwen Shi. (2024, Apr. 7). "A Review of Document Binarization: Main Techniques, New Challenges, and

Trends." Electronics [Online]. Available: <https://www.mdpi.com/2079-9292/13/7/1394> [Accessed: Feb. 28, 2025]

- [12] GeeksforGeeks. (2024, Jun 13). "Image Thresholding Techniques in Computer Vision" GeeksforGeeks [Online]. Available: <https://www.geeksforgeeks.org/image-thresholding-techniques-in-computer-vision/> [Accessed: Feb. 28, 2025]
- [13] Dr. Dulari Bhatt. (2024, Oct 14). "A Complete Guide on Hough Transform." Analytics Vidhya [Online]. Available: <https://www.analyticsvidhya.com/blog/2022/06/a-complete-guide-on-hough-transform/> [Accessed: Mar. 5, 2025]
- [14] GeeksforGeeks. (2024, Jun 07). "Difference Between Traditional Computer Vision Techniques and Deep Learning Based Approaches." GeeksforGeeks [Online]. Available: <https://www.geeksforgeeks.org/difference-between-traditional-computer-vision-techniques-and-deep-learning-based-approaches/> [Accessed: Mar. 7, 2025]
- [15] Zilliz. (n.d.). "What is Feature Extraction in Image Processing?" Zilliz [Online]. Available: <https://zilliz.com/ai-faq/what-is-feature-extraction-in-image-processing> [Accessed: Mar. 7, 2025]
- [16] Jason Brownlee. (2021, Jan. 27). "Object Recognition with Deep Learning." Machine Learning Mastery [Online]. Available: <https://machinelearningmastery.com/object-recognition-with-deep-learning/> [Accessed: Mar. 7, 2025]
- [17] IBM. (n.d.). "What Are Convolutional Neural Networks?" IBM [Online]. Available: <https://www.ibm.com/topics/convolutional-neural-networks> [Accessed: Mar. 14, 2025]
- [18] Allan Koudri. (2023, Dec. 8). "Mastering ResNet: Deep Learning for Image Recognition." IKOMIA [Online]. Available: <https://www.ikomia.ai/blog/mastering-resnet-deep-learning-image-recognition> [Accessed: Mar. 14, 2025]
- [19] Juan R. Terven and Diana M. Cordova-Esparza. (2024, Jan. 07). "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS." arXiv [Online]. Available: <https://arxiv.org/html/2304.00501v6/#bib.bib5> [Accessed: Mar. 24, 2025]
- [20] K. He, G. Gkioxari, P. Dollár, and R. Girshick. (2017). "Mask R-CNN." arXiv [Online]. Available: <https://arxiv.org/abs/1703.06870> [Accessed: Mar. 24, 2025]
- [21] Cem Dilmegani. (2024, Sep. 23). "Computer Vision Challenges." AIMultiple [Online]. Available: <https://research.aimultiple.com/computer-vision-challenges/> [Accessed: Mar. 27, 2025]

- [22] Jesse Anglen. (n.d.). "Future of Computer Vision: 2024 & Beyond." Rapid Innovation [Online]. Available: <https://www.rapidinnovation.io/post/future-of-computer-vision#5-edge-and-embedded-computer-vision> [Accessed: Mar. 27, 2025]
- [23] N. J. Zakaria, M. I. Shapiai, R. A. Ghani, M. N. M. Yassin, M. Z. Ibrahim, and N. Wahid, "Lane Detection in Autonomous Vehicles: A Systematic Review," *IEEE Access*, vol. 11, pp. 3729-3749, Jan.5, 2023, doi: 10.1109/ACCESS.2023.3234442. [Accessed: Mar. 28, 2025]
- [24] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "*Spatial As Deep: Spatial CNN for Traffic Scene Understanding*," arXiv preprint arXiv: 1712.06080, Dec. 2017. [Online]. Available: <https://arxiv.org/pdf/1712.06080> [Accessed: Apr. 22, 2025].
- [25] D. C.-E. Lin, "*Tutorial: Build a Lane Detector*," TDS Archive, Dec. 17, 2018. [Online]. Available: <https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132> [Accessed: Apr. 22, 2025].

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

جامعة عبد الحميد بن باديس-مستغانم-
كلية العلوم الدقيقة والإعلام الآلي

تصريح شرفي خاص بالالتزام بقواعد النزاهة العلمية
لإنجاز البحث

أنا الممضي أدناه،
الطالب(ة): بن سليمان مصطفى
رقم التسجيل الجامعي: 202037028667
الحامل لبطاقة التعريف الوطنية رقم: 100030963012720000 والصادرة بتاريخ: 2024.07.12
عن مزغران مستغانم،
المسجل ب: كلية العلوم الدقيقة والإعلام الآلي / قسم: علوم الحاسوب
شعبة: الذكاء الاصطناعي / التخصص: الذكاء الاصطناعي لإنترنت الأشياء
والمكلف بإنجاز مذكرة ماستر بعنوان:

اكتشاف ممرات الطريق في الظروف الصعبة

أصرح بشرفي أنني ألتزم بمراعاة المعايير العلمية والمنهجية ومعايير الأخلاقيات العلمية والنزاهة
الأكاديمية المطلوبة في إنجاز البحث ، وأتحمل المسؤولية الشخصية عن كل المحتوى المتضمن
في البحث المذكور أعلاه .

التاريخ: 23 جوان 2025

إمضاء المعني



*ملحق القرار الوزاري رقم 933 المؤرخ في 28 جويلية 2016 الذي يحدد القواعد المتعلقة بالوقاية من السرقة
العلمية ومكافحتها.