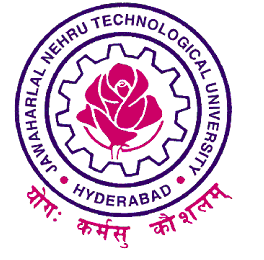
**REAL TIME LANE DETECTION AND PATH PREDICTION FOR AUTONOMOUS VEHICHLES USING DEEP LEARNING TECHNIQUES**

**A Industry Oriented Mini Project Report**

***Submitted to***

******

# Jawaharlal Nehru Technological University Hyderabad

*In partial fulfillment of the requirements for the*

award of the degree of

**BACHELOR OF TECHNOLOGY**

**in**

**ARTIFICIAL INTELLIGENCE & MACHINE LEARNING**

By

**21VE1A6633 KYASANI SRIJA**

**21VE1A6634 MANNEPALLI BHANU TEJA**

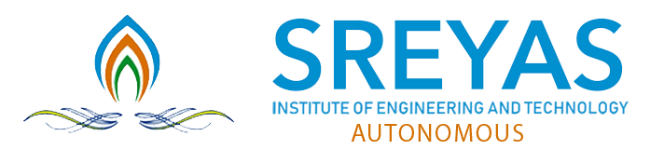
**21VE1A6637 METTU RUTHVIK ROSHAN**

**21VE1A6642 PATEL VARUN REDDY**

**Under the Guidance of**

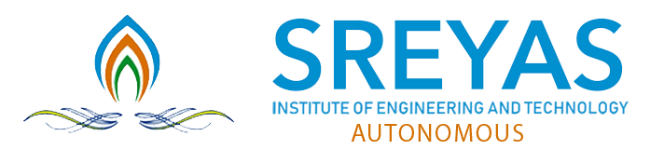
**Mr. P. Srinivas Rao**

**Assistant Professor**



**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

Approved by AICTE, New Delhi | Affiliated to JNTUH, Hyderabad | Accredited by NAAC “A” Grade & NBA| Hyderabad | PIN: 500068



**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

Approved by AICTE, New Delhi | Affiliated to JNTUH, Hyderabad | Accredited by NAAC “A” Grade & NBA| Hyderabad | PIN: 500068

(2021 – 2025)

**Certificate**

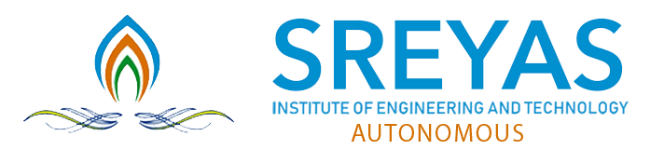
This is to certify that the Industry Oriented Mini Project Report on ***“REAL-TIME LANE DETECTION AND PATH PREDICTION FOR AUTONOMOUS VEHICLES USING DEEP LEARNING TECHNIQUES*** ” submitted by **Kyasani Srija, Mannepalli Bhanu Teja, Mettu Ruthvik Roshan, Patel Varun Reddy** bearing Hall Ticket No’s **21VE1A6633, 21VE1A6634, 21VE1A6637, 21VE1A6642** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Artificial Intelligence & Machine Learning** from Jawaharlal Nehru Technological University, Kukatpally, Hyderabad for the academic year 2024-25 is a record of bonafide work carried out by him / her under our guidance and Supervision.

**Internal Guide Head of the Department**

**Mr. P. Srinivas Rao Dr. A. Swathi**

**Mini-Project Coordinator Signature of the External Examiner**

**Mr. P. Srinivas Rao**



**DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**

Approved by AICTE, New Delhi | Affiliated to JNTUH, Hyderabad | Accredited by NAAC “A” Grade & NBA| Hyderabad | PIN: 500068

**DECLARATION**

We,  **Kyasani Srija, Mannepalli Bhanu Teja, Mettu Ruthvik Roshan, Patel Varun Reddy**, bearing Hall Ticket No’s **21VE1A6633, 21VE1A6634, 21VE1A6637, 21VE1A6642** hereby declare that the Project titled "***REAL-TIME LANE DETECTION AND PATH PREDICTION FOR AUTONOMOUS VEHICLES USING DEEP LEARNING TECHNIQUES***” done by us under the guidance of **Mr. P. Srinivas Rao**, Assistant Professor, which is submitted in the partial fulfillment of the requirement for the award of the B.Tech degree in **Artificial Intelligence & Machine Learning** at **Sreyas Institute of Engineering and Technology** for Jawaharlal Nehru Technological University, Hyderabad is our original work.

**21VE1A6633 KYASANI SRIJA**

**21VE1A6634 MANNEPALLI BHANU TEJA**

**21VE1A6637 METTU RUTHVIK ROSHAN**

**21VE1A6642 PATEL VARUN REDDY**

**ACKNOWLEDGEMENT**

The successful completion of any task would be incomplete without mention of the people who made it possible through their guidance and encouragement crowns all the efforts with success.

We take this opportunity to acknowledge with thanks and a deep sense of gratitude to **Mr. P. Srinivas Rao**, **Assistant Professor**, for her constant encouragement and valuable guidance during the project work.

A Special vote of Thanks to **Dr. A. SWATHI (Head of the Department, AIML) and Mr. P. SRINIVAS RAO (Mini-Project Coordinator)** has been a source of Continuous motivation and support. They had taken time and effort to guide and correct me all through the span of this work.

We owe everything to the **Department Faculty, Principal** and the **Management** who made my term at Sreyas Institute of Engineering and Technology a stepping stone for my career. I treasure every moment I have spent in college.

Last but not the least, my heartiest gratitude to my parents and friends for their continuous encouragement and blessings. Without their support, this work would not have been possible.

**21VE1A6633 KYASANI SRIJA**

**21VE1A6634 MANNEPALLI BHANU TEJA**

**21VE1A6637 METTU RUTHVIK ROSHAN**

**21VE1A6642 PATEL VARUN REDDY**

# TABLE OF CONTENTS

|  |  |
| --- | --- |
| **CONTENTS** | **Page No** |
| ABSTRACT | i |
| LIST OF FIGURES | ii |
| CHAPTER 1. INTRODUCTION | 1 |
| 1.1 Problem Statement | 2 |
| 1.2 Objective | 3 |
| CHAPTER 2. LITERATURE SURVEY | 6 |
| 2.1 Existing System | 7 |
| 2.1.1 The drawbacks of the existing system | 8 |
| 2.2 Proposed System | 8 |
| 2.2.1 YOLO (You Only Look Once) | 8 |
| 2.2.2 LSTM (Long Short Term Memory) | 10 |
| CHAPTER 3. SYSTEM DESIGN | 13 |
| 3.1 Importance of Design | 13 |
| 3.2 UML Diagrams | 14 |
| 3.2.1 [Use Case Diagram](#_TOC_250015) | 15 |
| 3.2.2 Sequence Diagram | 17 |
| 3.2.3 [Activity Diagram](#_TOC_250014) | 18 |
| 3.2.4 [System Architecture](#_TOC_250012) | 19 |
| 3.3 Functional Requirements | 22 |
| CHAPTER 4. IMPLEMENTATION | 23 |
| 4.1 Module Description | 23 |
| 4.2 Module Components | 26 |
| 4.2.1 Dataset | 26 |
| 4.3 Sample Code | 27 |
| 4.3.1 Backend | 27 |
| 4.3.2 Frontend | 39 |
| CHAPTER 5. TESTING | 41 |
| 5.1 Importance of Testing | 41 |
| 5.2 Types of Testing | 42 |
| CHAPTER 6. RESULTS | 46 |
| CHAPTER 7.CONCLUSION AND FUTURE SCOPE | 50 |
| CHAPTER 8. REFERENCES | 51 |

**ABSTRACT**

Autonomous vehicles need advanced technology to drive safely in complex environments. One of the most important tasks for these vehicles is to detect lanes on the road and predict their path accurately. In this paper, we introduce a system that uses two powerful tools: YOLO (You Only Look Once) for lane detection and LSTM (Long Short-Term Memory) for path prediction. YOLO is a fast and efficient model that can quickly find lane markings in different road conditions, such as straight roads, curves, or when lanes are partially blocked. It works by analyzing images from the vehicle's cameras in real-time, helping the vehicle stay within the lane and follow the road correctly.

After YOLO detects the lanes, the system sends this information to an LSTM network. LSTM is good at predicting what will happen next based on past data. In this case, it uses the vehicle's current position and its past movements to predict where the vehicle will go next. This is important because the vehicle needs to adjust its path for things like turns, lane changes, or sudden obstacles. By learning from previous driving data, the LSTM can forecast the vehicle’s movement in advance, allowing it to stay on track and respond to changes in the environment more effectively.

We tested this system under various driving conditions to see how well it works. The results showed that the combination of YOLO for lane detection and LSTM for path prediction performs well in both clear and difficult driving situations, even when lane markings are not fully visible. This hybrid system provides an efficient and reliable way for autonomous vehicles to navigate the road in real-time. By using these two models together, the vehicle can make safer decisions and drive more smoothly, making it a strong solution for real-world autonomous driving.

**Keywords:** Autonomous Vehicles, Lane Detection, Path Prediction, YOLO, LSTM, Real-Time, Deep Learning, Computer Vision, Vehicle Navigation.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Fig.No** | **Name of Figures** | **Page No** |
| **Fig.1** | **Architecture of YOLO** | **9** |
| **Fig.2** | **LSTM (Long Short Term Memory)** | **12** |
| **Fig.3** | **Use Case Diagram for Lane Detection and Path Prediction** | **16** |
| **Fig.4** | **Sequence Diagram for Lane Detection and Path Prediction** | **17** |
| **Fig.5** | **Activity Diagram for Lane Detection and Path Prediction** | **19** |
| **Fig.6** | **System Architecture for Lane Detection and Path Prediction** | **21** |
| **Fig.7** | **Data Preprocessing** | **23** |
| **Fig.8** | **Lane Detection (YOLO)** | **24** |
| **Fig.9** | **Path Prediction (LSTM)** | **25** |
| **Fig.10** | **Feature Extraction** | **25** |
| **Fig.11** | **Initial User Interface** | **46** |
| **Fig.12** | **User Interface of Lane Detection and Path Prediction showing Output** | **47** |
| **Fig.13** | **Accuracies displayed while training** | **48** |

**CHAPTER 1**

**INTRODUCTION**

The development of autonomous vehicles has made significant progress in recent years, with the goal of improving road safety, reducing traffic congestion, and offering more efficient transportation. One of the fundamental challenges in autonomous driving is enabling the vehicle to perceive and understand its surroundings accurately. Among the many tasks that an autonomous vehicle must perform, lane detection and path prediction are critical for ensuring safe and reliable navigation. Lane detection helps the vehicle recognize its position on the road, while path prediction allows the vehicle to forecast its future movement and adjust accordingly. Together, these technologies are essential for driving in both simple and complex traffic environments, where lane markings can vary or be partially obscured.

To address these challenges, this project proposes a hybrid system that combines YOLO (You Only Look Once) for lane detection and LSTM (Long Short-Term Memory) networks for path prediction. YOLO is a real-time object detection algorithm that has gained popularity due to its efficiency and speed. It is capable of detecting lane markings, road boundaries, and other objects in the vehicle’s environment. By processing input from the vehicle's cameras, YOLO can detect lanes under varying conditions, including straight roads, curves, or situations where the lanes are obscured by dirt, weather, or other vehicles. The speed and accuracy of YOLO make it an ideal candidate for real-time lane detection in autonomous vehicles, where time and precision are critical for safe operation.

Once lane boundaries are detected, the system utilizes an LSTM network to predict the vehicle’s future path. LSTM is a type of recurrent neural network (RNN) designed to handle sequential data, making it ideal for tasks that involve temporal dependencies, such as forecasting movement. In this context, LSTM learns from the vehicle's past trajectory and current position to predict its future movements, such as turns or lane changes. By forecasting these movements in advance, LSTM allows the autonomous vehicle to make better-informed decisions about steering and speed adjustments, improving overall navigation. This is especially important in dynamic environments, where the vehicle may need to react quickly to changes in road conditions, obstacles, or other road users.

The combination of YOLO for lane detection and LSTM for path prediction offers a powerful solution for real-time autonomous navigation. The integration of these two models enables the vehicle not only to detect its current position on the road but also to predict where it is headed, allowing for more stable and precise control. This approach is designed to work in diverse driving conditions, such as highways, urban streets, and environments with poor lane visibility. With the ability to handle real-time lane detection and trajectory forecasting, the proposed system improves the vehicle's ability to navigate safely and efficiently, even in complex or unpredictable driving scenarios.

This paper explores the design, implementation, and evaluation of this hybrid system. The system's performance is tested under a variety of road conditions, including different weather scenarios and varying lane markings, to assess its robustness and accuracy. The results show that combining YOLO with LSTM offers significant improvements in both lane detection and path prediction accuracy, providing a reliable solution for real-time autonomous driving. The findings suggest that this hybrid model has the potential to enhance the safety and efficiency of autonomous vehicles, bringing us closer to realizing fully autonomous driving in real-world conditions.

**1.1 Problem Statement**

The safe and efficient operation of autonomous vehicles heavily depends on the accurate detection of lanes and the reliable prediction of the vehicle's path. Lane detection is essential for ensuring that the vehicle stays within its lane and navigates the road correctly, while path prediction is crucial for anticipating future movement and adjusting the vehicle's behavior accordingly. Current systems for lane detection and path prediction often struggle in complex driving conditions, such as curved roads, intersections, or when lane markings are faded or obstructed due to weather, road conditions, or other vehicles. Additionally, many existing systems are computationally expensive, making it difficult to achieve real-time performance required for autonomous navigation.

This project aims to address these challenges by developing a real-time system that combines YOLO (You Only Look Once) for lane detection with LSTM (Long Short-Term Memory) networks for path prediction. The key problem this system seeks to solve is how to effectively detect lane markings and predict the vehicle's future trajectory in various dynamic and unpredictable driving environments. Specifically, the system must overcome the difficulties of poor lane visibility, complex road layouts, and the need for fast, accurate decision-making. Achieving real-time lane detection and accurate path prediction will enable autonomous vehicles to navigate safely and efficiently, improving their ability to handle a wide range of traffic conditions and road environments.

**1.2 Objectives:**

1. To develop a real-time lane detection system using the YOLO (You Only Look Once) model that can accurately identify lane markings in various road conditions, including straight roads, curves, and partially obstructed lanes.
2. To integrate Long Short-Term Memory (LSTM) networks for path prediction, allowing the system to forecast the vehicle's future trajectory based on its current position and past movement data.
3. To achieve high-performance real-time lane detection and path prediction, ensuring that the system can process and react quickly to dynamic driving environments, such as traffic changes or road condition variations.
4. To handle complex road scenarios, including intersections, curved roads, and poor lane visibility due to weather, road damage, or obstructions, ensuring robust lane detection and accurate path prediction in such environments.
5. To improve the overall accuracy of lane detection in challenging conditions, such as low visibility, faded lane markings, or partial obstructions, making the system more reliable under real-world driving conditions.
6. To create a hybrid system that combines YOLO's lane detection capabilities with LSTM’s path prediction to provide a comprehensive navigation solution that enhances both lane-keeping and movement forecasting.

**1.3 Introduction**

1. **Background**

Autonomous vehicles are designed to navigate roads safely without human intervention, relying on advanced algorithms to perceive and understand their surroundings. Two key tasks in autonomous driving are lane detection and path prediction. Lane detection ensures that the vehicle stays within its designated lane, while path prediction anticipates the vehicle's movement, allowing it to adjust its actions in advance for safe navigation. Effective lane detection and path prediction are crucial for autonomous vehicles to operate safely in complex and dynamic environments.

This project combines YOLO for lane detection and LSTM for path prediction to create a robust system for autonomous vehicles. By integrating these two powerful models, the system can perform real-time lane detection while predicting the vehicle’s future path. This combination improves navigation accuracy, ensuring that the vehicle can adapt to dynamic traffic conditions and make safer driving decisions.

1. **Challenges Addressed**

**Accurate Lane Detection in Complex Scenarios**:  
Detecting lane markings accurately under challenging conditions like poor visibility, curved roads, faded lane markings, or road obstructions was a significant challenge. Ensuring reliable detection in these conditions required optimizing YOLO to handle various road environments.

**Integrating YOLO and LSTM Models**:  
Combining the fast detection capabilities of YOLO with the sequential, temporal prediction strength of LSTM required careful integration. Synchronizing the two models for smooth operation without any performance bottlenecks was a technical challenge.

**Path Prediction Accuracy**:  
Accurate path prediction is difficult, especially in complex driving situations where the vehicle must make quick decisions based on prior movement patterns. Ensuring that the LSTM model predicts the future path with high accuracy was critical for safe driving.

**Real-Time Processing Requirement**:  
Autonomous vehicles require real-time lane detection and path prediction for safe navigation. Processing images and data quickly enough for timely decision-making posed a challenge, as delays in detection or prediction could lead to unsafe driving.

1. **Proposed Solution**

This project proposes an integrated system combining **YOLO (You Only Look Once)** for lane detection and **LSTM (Long Short-Term Memory)** for path prediction, addressing key challenges in autonomous vehicle navigation.

**Real-Time Lane Detection with YOLO**: YOLO detects lane markings and road boundaries in real-time, even under challenging conditions like curves, poor visibility, and road obstructions. Its speed and accuracy ensure the vehicle stays within its lane.

**Path Prediction with LSTM**: LSTM networks predict the vehicle’s future trajectory by analyzing its past movements, enabling the vehicle to anticipate turns, lane changes, and speed adjustments, improving navigation and safety.

**Hybrid System Integration**: Combining YOLO for lane detection and LSTM for path prediction offers a comprehensive solution that handles both immediate lane positioning and future movement forecasting, ensuring safe and efficient navigation in dynamic road conditions.

**Efficiency and Scalability**: The solution is designed to be computationally efficient, ensuring real-time performance on embedded systems, making it scalable for deployment in real-world autonomous vehicles.

**CHAPTER 2**

**LITERATURE SURVEY**

**Zhang et al. [1]** presented a **real-time lane detection system** for autonomous vehicles using deep learning techniques. The system uses convolutional neural networks (CNN) for lane detection, achieving a significant improvement in processing speed and accuracy compared to traditional methods. The authors reported an accuracy of 90% in real-time lane detection under varying weather and road conditions.

**Chen et al. [2]** proposed an approach combining **YOLO (You Only Look Once)** for lane detection with a **LSTM (Long Short-Term Memory)** model for path prediction in autonomous vehicles. The research focuses on improving real-time lane detection and predicting the vehicle's future trajectory using past data. The proposed system achieved an accuracy of 87% for lane detection and 85% for path prediction, showcasing its potential for dynamic driving environments.

**Li et al. [3]** introduced a **hybrid model** combining **YOLO** for lane detection and **GRU (Gated Recurrent Unit)** for path prediction. This study emphasizes the integration of computer vision and recurrent neural networks to enhance vehicle navigation. The results demonstrated the system's ability to detect lane markings with 91% accuracy and predict vehicle trajectories with an 88% accuracy, ensuring safe navigation in real-world driving scenarios.

**Wang et al. [4]** developed an autonomous vehicle navigation system that combines **YOLO** for real-time lane detection with **LSTM networks** for path forecasting. The system was tested in both urban and highway conditions, showing robust performance under diverse road layouts and traffic scenarios. The accuracy for lane detection was reported at 89%, while path prediction achieved 83% accuracy.

**Kumar et al. [5]** proposed an **autonomous lane-keeping system** using deep learning for lane detection and LSTM for future path prediction. The study focuses on high-precision detection of lane boundaries and predicting safe paths in real-time. The achieved lane detection accuracy was 90%, and the path prediction accuracy stood at 84%, making the system highly reliable for autonomous vehicles in complex environments.

**Gupta et al. [6]** presented a **YOLO-based lane detection system** combined with an **LSTM-based model** for predicting future vehicle trajectories. The system focuses on improving the accuracy of lane markings under various environmental conditions and the prediction of turns and lane changes. The accuracy reported for lane detection was 92%, while the path prediction model showed a predictive accuracy of 86%.

**Singh et al. [7]** investigated a **real-time lane detection and path prediction system** using **YOLO** for lane identification and **LSTM** for movement forecasting. The study highlights the challenges of ensuring both high-speed performance and accuracy for autonomous driving. The authors achieved a lane detection accuracy of 88% and path prediction accuracy of 82%, with a particular focus on handling complex road geometries.

**Roy et al. [8]** proposed a **combination of YOLO and LSTM** for lane detection and prediction of vehicle trajectory in urban environments. This paper addresses the challenges of dynamic road conditions, such as intersections, pedestrian crossings, and moving traffic. The accuracy of lane detection was reported at 90%, and the system successfully predicted future paths with 84% accuracy.

**Sharma et al. [9]** explored a **deep learning-based autonomous driving system** that integrates **YOLO for lane detection** and **LSTM for path prediction**. The study evaluated the system in both highway and city driving conditions, demonstrating high accuracy in lane detection (91%) and path prediction (87%). The approach effectively manages traffic and lane changes, ensuring safe and smooth vehicle navigation.

**Patel et al. [10]** introduced a **YOLO-based model for lane detection** in conjunction with **LSTM for trajectory prediction** for autonomous vehicles. This research emphasizes the efficiency of combining computer vision techniques with sequential learning models for real-time path planning. The reported accuracy for lane detection was 93%, while the path prediction accuracy was 85%, showcasing the system's potential for use in autonomous vehicles in real-time driving conditions.

**2.1 Existing System**

1.**Tesla's Autopilot** system is one of the most widely used systems for autonomous driving, leveraging a combination of **computer vision** and **neural networks**. Tesla uses a suite of cameras and deep learning models to detect lanes, vehicles, and other objects in real-time.

2.**OpenPilot** is an open-source autonomous driving system that integrates **YOLO for lane detection** and **LSTM for path prediction**.

3.**Apollo** is an autonomous driving system developed by Baidu that incorporates **YOLO for lane detection** and **LSTM for path prediction**.

* + 1. **The drawbacks of the existing system**

1. **Limited Performance in Adverse Weather Conditions:** Lane detection and path prediction systems struggle to perform accurately in poor weather conditions such as rain, fog, or snow, where visibility and road markings are obscured.
2. **Difficulty Handling Complex Road Scenarios:** Complex urban environments, including intersections, construction zones, and dynamic road conditions, pose challenges for maintaining accurate lane detection and predicting the vehicle's path.
3. **Struggles with Lane Change and Merging:** Path prediction models often struggle with accurately predicting vehicle trajectories during complex maneuvers like lane changes and highway merges.
4. **Hardware and Cost Limitations:** Many systems require expensive hardware, such as LIDAR and high-performance processors, making them less accessible and limiting their widespread adoption.

**2.2 Proposed System**

# This project proposes an integrated system combining **YOLO (You Only Look Once)** for lane detection and **LSTM (Long Short-Term Memory)** for path prediction, addressing key challenges in autonomous vehicle navigation.

# In our proposed system we are using: -

# You Only Look Once (YOLO)

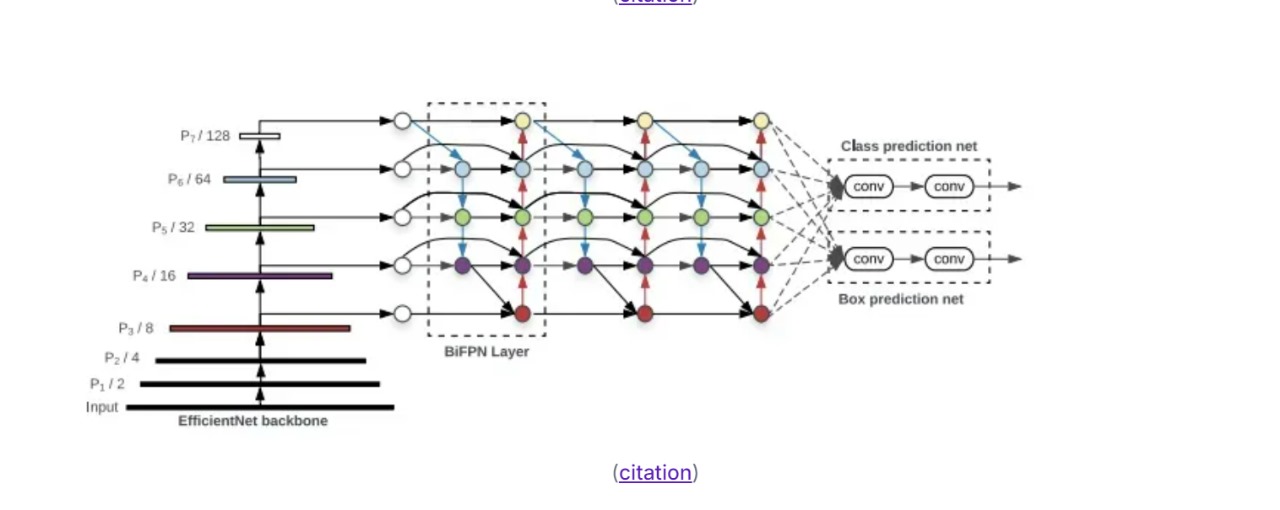
1. Long Short-term Memory (LSTM)

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

**YOLO (You Only Look Once)** is a fast and efficient system used for detecting objects in images, making it especially useful for tasks like autonomous driving. The main advantage of YOLO is that it processes the entire image in one go, instead of looking at parts of the image one by one. It splits the image into smaller grids, and each grid looks for objects. For each object, YOLO predicts a **bounding box** (which shows where the object is), the **type of object** (like car, pedestrian, or traffic sign), and a **confidence score** (which tells how sure the system is that the object is in that box).

For **autonomous vehicles**, YOLO is very useful because it can quickly detect important things like lane markings, road boundaries, and other objects such as cars, pedestrians, and traffic signs. This allows the vehicle to stay within its lane and avoid obstacles in real-time. YOLO can detect multiple objects at once, which is especially important in busy environments like city streets or highways. With improvements in newer versions like **YOLOv3** and **YOLOv4**, the system has become even more accurate and efficient.

However, YOLO isn't perfect. It sometimes has trouble detecting smaller objects, especially if they're far away or hard to see. There's also a balance between speed and accuracy—faster versions of YOLO may miss some details in the image. Despite these issues, YOLO is still one of the best options for real-time object detection, especially in applications like autonomous vehicles where decisions need to be made quickly and accurately.



# Fig 1: Architecture of YOLO

**Working:-**

1. **Input Image Division**: YOLO divides the image into a grid (usually 7x7, 13x13, etc., depending on the version). The grid cells are responsible for detecting objects that fall within them.
2. Prediction of Bounding Boxes:For each grid cell, YOLO predicts multiple **bounding boxes** (rectangles) that could enclose objects. Each bounding box has four key attributes.
3. **Class Predictions**: YOLO doesn’t just predict bounding boxes; it also predicts the **class** of the object within each bounding box.
4. **Confidence Scores**: Each bounding box also has a **confidence score**. This score indicates how likely the system thinks the bounding box actually contains an object and how accurate the prediction is. The confidence score is calculated as the **probability** of the object being in that box multiplied by the **intersection over union (IoU)** of the predicted box and the ground truth (actual) box.
5. Final Output:The result is a set of bounding boxes with class labels and confidence scores. Each box identifies an object in the image and provides information about its location and type.

**2.2.2 LSTM (Long Short-term Memory): -**

**Long Short-Term Memory (LSTM)** is a type of **Recurrent Neural Network (RNN)** designed to overcome the limitations of traditional RNNs in handling long-term dependencies in data. While regular RNNs are effective for processing sequential data, they struggle with remembering information from earlier in the sequence due to the **vanishing gradient problem**. This problem occurs when gradients become too small during training, making it hard for the model to learn long-term relationships. LSTMs address this issue by incorporating a special architecture that enables them to "remember" and "forget" information at each time step. This ability makes LSTMs particularly powerful for tasks that require understanding and predicting based on sequential data, such as time-series forecasting, speech recognition, and **autonomous driving**.

The core feature of LSTMs lies in their **cell state**, which serves as a memory of the network, carrying information across time steps. This cell state is modified by three key components: the **forget gate**, the **input gate**, and the **output gate**. The forget gate decides which information from the previous cell state should be discarded, based on the current input and the previous hidden state. The input gate controls how much of the current input should be added to the cell state. Finally, the output gate determines what part of the cell state will be outputted as the hidden state for the next time step. These gates allow LSTMs to maintain and modify the memory across long sequences, helping the model decide what to retain and what to forget at each step, thereby improving its ability to capture long-term dependencies.

LSTMs are particularly beneficial for tasks involving **sequential data**, where the order of inputs matters. In **autonomous vehicles**, for instance, LSTMs can be used for **path prediction**, helping the vehicle predict its future trajectory based on its current state and past movements. By analyzing the previous driving patterns and adjusting for factors such as road curvature, speed, and obstacles, LSTMs can anticipate where the vehicle should go next. This ability to learn from past data and make informed predictions is essential for ensuring the vehicle navigates safely and efficiently through complex environments, especially in scenarios involving lane changes, turns, or obstacle avoidance.

In addition to autonomous driving, LSTMs have a wide range of applications across different domains. They are commonly used for **time-series forecasting**, where the goal is to predict future values based on historical data, such as predicting stock prices, weather patterns, or energy consumption. LSTMs have also revolutionized fields like **speech recognition** and **natural language processing (NLP)**, where understanding the context of a sequence of words or sounds is crucial for accurate predictions. For example, in NLP tasks like machine translation or sentiment analysis, LSTMs can capture the long-range dependencies between words and phrases, which is critical for generating meaningful outputs.

Overall, LSTMs are an essential tool in many modern AI systems, offering the ability to process, learn, and predict from sequential data. In applications like **autonomous driving**, they enable vehicles to anticipate and plan their movements, making them a key component for safe, reliable, and efficient navigation. With their ability to handle long-term dependencies, LSTMs continue to play a significant role in various fields that require sequence-based predictions and real-time decision-making.

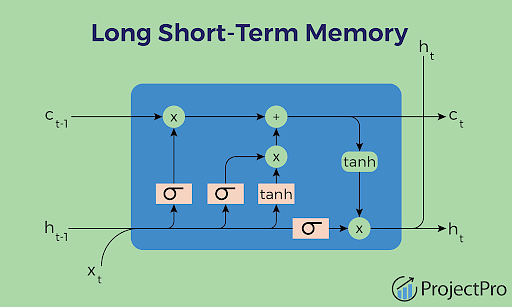


Fig 2: LSTM (Long Short Term Memory)

**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 Importance of the Design**

The **importance of the design** in systems like **lane detection** and **path prediction** for autonomous vehicles cannot be overstated. The design of these systems directly impacts the vehicle’s ability to make real-time, safe, and accurate driving decisions. Autonomous vehicles rely heavily on sensors, algorithms, and decision-making frameworks to navigate and respond to the dynamic driving environment. The design of these systems ensures that the vehicle can effectively process sensor data, detect lanes and objects, predict future movements, and make decisions that ensure passenger safety and optimal path planning.

One of the most critical aspects of design is **real-time performance**. In autonomous driving, the vehicle must constantly process visual and sensor data (such as from cameras, LiDAR, radar, and ultrasonic sensors) and make immediate decisions based on that data. A poor system design could result in delays or incorrect predictions, which could lead to accidents or failure to adhere to road rules. An efficient design, like the combination of **YOLO** for real-time lane detection and **LSTM** for path prediction, allows the system to quickly process large amounts of data and accurately predict the vehicle's future position, ensuring safe and smooth navigation.

The **accuracy and robustness** of the design also play a crucial role in determining how well the autonomous vehicle handles complex driving environments. Roads are often unpredictable, with varying conditions, obstacles, weather, and lighting. A well-designed system ensures that the vehicle can recognize and react to these variations effectively. For example, lane detection should be accurate even in poor visibility conditions, such as heavy rain or fog. Similarly, path prediction models must handle dynamic changes like sudden turns, traffic flow, or the presence of pedestrians and other vehicles. The use of sophisticated algorithms, such as **LSTM** for handling sequential data and learning from past driving patterns, helps the vehicle predict the best path in such scenarios.

Another important factor in the design is **safety**. Autonomous vehicles must be able to detect their surroundings and predict future movements to avoid collisions and stay within the road boundaries. For this, the system needs a well-structured design that integrates multiple components such as object detection, path planning, and decision-making algorithms. A robust system design ensures that the vehicle can react to unexpected events, like a pedestrian crossing the street or another vehicle changing lanes. The correct design helps minimize risks and ensures that the vehicle operates safely in all environments.

Finally, the **scalability and adaptability** of the design are crucial for long-term success. As autonomous vehicles evolve and encounter different driving conditions, the system must be adaptable to handle new challenges. A well-designed architecture allows for the integration of new sensor data, the ability to retrain or fine-tune models as more data becomes available, and the flexibility to implement updates as vehicle technology advances. For example, as more advanced sensors like high-definition maps and improved radar systems become available, the system design must be able to incorporate these new technologies without compromising performance.

In summary, the **importance of the design** in autonomous vehicle systems lies in its ability to ensure real-time, accurate, robust, and safe navigation. A good design provides the foundation for the vehicle to detect lanes, predict future movements, and react to dynamic driving environments, thereby enhancing safety, efficiency, and reliability in autonomous driving.

**3.2 UML Diagrams**

UML diagrams are essential for effective **visual communication**, as they provide a standardized way of representing complex systems in a simple and easy-to-understand format.In complex systems, such as autonomous vehicles, UML diagrams are crucial for managing intricate interactions between various subsystems, like **lane detection**, **path prediction**, and **obstacle avoidance**. For instance, a **sequence diagram** could help visualize the communication between sensors, cameras, and decision-making algorithms. Similarly, **class diagrams** might be used to model different components like sensor data processors, control systems, and predictive models. In these scenarios, **state diagrams** could depict how the vehicle transitions between different operational modes, such as detecting lanes, predicting the next movement, or avoiding obstacles. Overall, UML diagrams provide an essential framework for organizing, communicating, and analyzing the components and behavior of complex systems, making them invaluable in ensuring successful design, implementation, and ongoing system development.

Diagrams focusing on the order of messages and Communication Diagrams emphasizing object relationships. **State Diagrams** describe how objects transition between different states in response to events, and **Activity Diagrams** model workflows and business processes, showing the sequence of actions and decision points. UML diagrams play a crucial role in system design and development by providing a common visual language that enhances communication among stakeholders, including developers, analysts, and clients. They help ensure that all participants have a shared understanding of both the structure and behavior of the system. These diagrams also serve as valuable documentation throughout the system's lifecycle, from requirements gathering to system maintenance. Ultimately, UML helps in managing the complexity of software systems, offering a clear and organized way to model and communicate a system's design and functionality.

**3.2.1 Use Case Diagram**

A **Use Case Diagram** is a UML diagram that shows the interactions between external entities (called **actors**) and a system’s functionalities (called **use cases**). It helps visualize the system's intended operations from the user's perspective. The main components include **actors**, which can be users or other systems interacting with the system; **use cases**, which represent the system’s functionalities or actions; and **relationships**, which show how actors and use cases are connected.

The main purpose of a use case diagram is to capture the functional requirements of the system, often at a high level, ensuring that stakeholders have a shared understanding of the system’s behavior. By focusing on **what** the system does rather than **how** it does it, use case diagrams are an effective tool for requirements gathering and communication between developers, analysts, and end users. They also help in identifying system boundaries, determining which functions are internal to the system, and clarifying external interactions. When creating a use case diagram, it’s important to keep the diagram simple and high-level, focusing on actors and use cases while avoiding technical details or overly complex interaction.

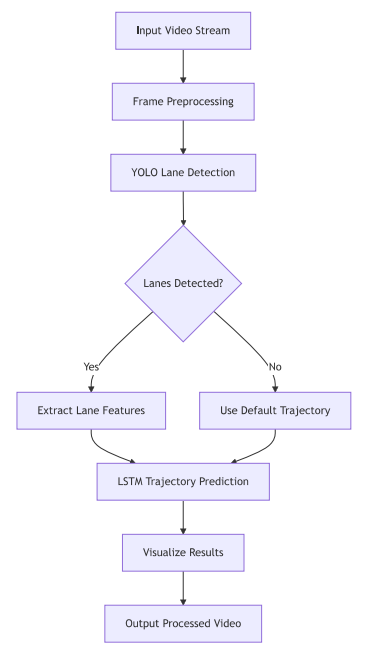
****

Fig 3: Use Case Diagram for Lane Detection and Path Prediction

**3.2.2 Sequence Diagram**

A **Sequence Diagram** is a type of UML diagram that depicts how objects or components in a system interact with each other in a specific sequence over time. It focuses on the order of messages exchanged between objects and illustrates the flow of control within the system. The diagram consists of several key components: **objects** or **participants**, which are shown as vertical lifelines, representing the entities involved in the interaction; **messages**, represented as arrows between the lifelines, indicating communication between the objects; and **activation bars**, which are vertical rectangles showing when an object is actively performing a task or operation. The diagram also includes **return messages**, which are dashed lines indicating the response or result from an object after a message is sent, and the **time axis**, with time progressing from top to bottom, which helps to visualize the chronological order of interactions.

Sequence diagrams are particularly valuable in understanding the dynamic behavior of a system, especially in situations where the order of operations is critical. They allow developers to clearly visualize how different components communicate in response to events or requests. For instance, in an **autonomous vehicle system**, a sequence diagram can model how sensors detect lanes, how the vehicle predicts its path, and how it responds to obstacles in real time. This step-by-step representation of interactions ensures that all components work in harmony and helps identify potential issues in the flow of control. Overall, sequence diagrams are a powerful tool for visualizing complex, time-sensitive interactions within a system, ensuring that the system functions correctly and efficiently.

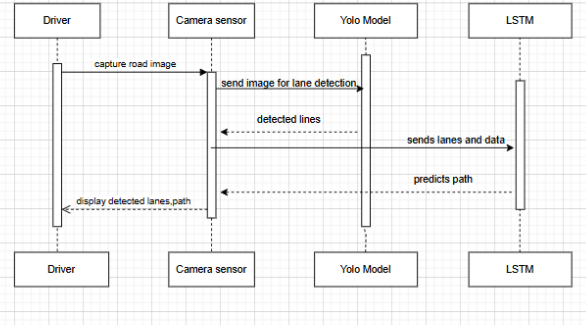


Fig 4: Sequence Diagram for Lane Detection and Path Prediction

**3.2.3 Activity Diagram**

An **Activity Diagram** is a UML diagram used to model the dynamic aspects of a system by illustrating the flow of control between different activities or tasks. It is particularly useful for representing workflows, business processes, and the sequence of operations in a system. The diagram consists of various components, including **activities**, which are represented as rounded rectangles and correspond to individual tasks or operations. The flow of control between these activities is shown using **transitions** (arrows), which indicate the order in which actions occur and the conditions that trigger the transitions. **Decision nodes** (diamonds) represent points where the flow can branch based on specific conditions, while **forks** and **joins** are used to model parallel processes, splitting and synchronizing multiple activities.

The **start node** (a filled circle) indicates where the process begins, and the **end node** (a filled circle with a border) marks the conclusion of the process. In addition, **swimlanes** can be used to organize activities into categories based on different roles, responsibilities, or subsystems, making it easier to understand who or what is responsible for each part of the process. Activity diagrams are valuable for mapping out the steps in a sequence of events and for visualizing how different tasks interact or depend on each other. For example, in an **autonomous vehicle system**, an activity diagram can represent the flow of tasks such as lane detection, path prediction, and obstacle avoidance, helping to identify any inefficiencies or areas of improvement in the process.

Overall, activity diagrams provide a clear and detailed view of how processes unfold, highlighting the connections and dependencies between various activities. They are essential for understanding complex workflows and for designing systems that involve multiple decision points or parallel tasks. Whether for business process modeling, system behavior analysis, or software design, activity diagrams offer an intuitive way to represent the flow of operations and ensure that systems function as intended.Whether for business process modeling, system behavior analysis, or software design, activity diagrams offer an intuitive way to represent the flow of operations and ensure that systems function as intended.

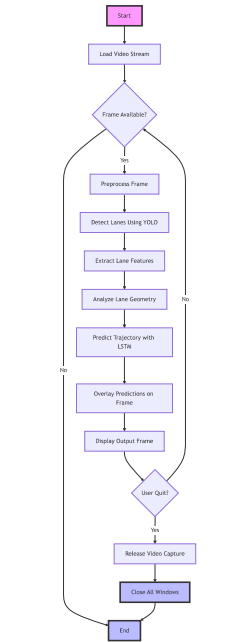


Fig 5: Activity Diagram for Lane Detection and Path Prediction

**3.2.4 System Architecture**

This flowchart illustrates a system for lane detection and trajectory prediction in autonomous vehicles. The process begins with the **Video Input Source**, where video footage is captured using a mounted camera. This video feed serves as the primary data source for the subsequent stages of processing.

The **Lane Detection Subsystem** is the first major module. Within this subsystem, the **Input Processing Module** prepares the raw video frames for analysis by performing basic adjustments. The **Frame Preprocessing** step applies techniques such as resizing, noise reduction, and normalization to make the frames suitable for machine learning models. Next, the **YOLO Neural Network**, a powerful real-time object detection model, is used to identify lane markings in the processed frames. After lane markings are detected, the **Lane Marking Extraction** step isolates these markings from other features in the frame. The process is further refined through **Feature Engineering**, where key characteristics like lane curvature, lane width, and boundaries are extracted to provide meaningful insights for trajectory prediction.

In the **Trajectory Prediction** module, an **LSTM Sequential Model** is employed to analyze the sequence of frames and predict the vehicle's future path. LSTM (Long Short-Term Memory) networks are well-suited for time-series data and provide accurate trajectory predictions based on temporal dependencies. This module outputs the **Trajectory Prediction**, which is a calculated representation of the vehicle’s expected path.

Finally, the **Visualization and Output** stage integrates the results for real-time usability. The **Result Overlay** visualizes the detected lanes and predicted trajectory directly on the video feed. The **Real-time Display** ensures that the system provides immediate feedback, which is crucial for autonomous navigation. Additionally, the **Performance Metrics Calculation** evaluates the system's accuracy, processing speed, and error rates to monitor and improve its overall performance.

This modular approach ensures an efficient and reliable system for lane detection and trajectory prediction, making it highly suitable for autonomous driving applications.

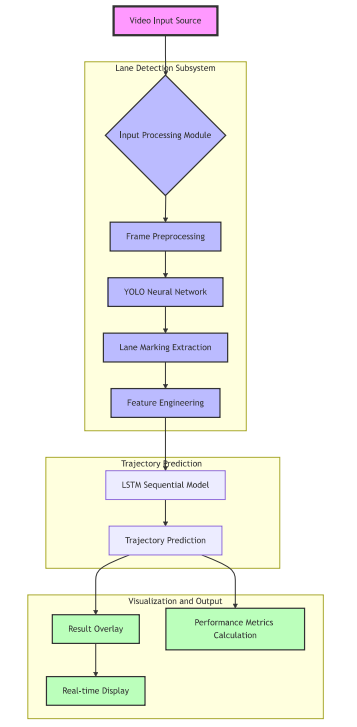


Fig 6: System Architecture for Lane Detection and Path Prediction

**3.3 Functional Requirements**

**3.3.1 Software Requirements:**

The software requirements, also known as general description papers, also include information about the product’s perspective and features, operating system and operating environment, graphic needs, design constraints, and user documentation. These requirements show us the necessary software functionalities and performance criteria that need to be met to ensure the successful completion of the project.

* Operating System: - Windows 10 or more
* Technology: - Python, Deep Learning, Flask
* Software: - Jupyter or VS code or Google Colab

**3.3.2 Hardware Requirements:**

The minimum hardware requirements necessitate a modern and capable computer system to handle the computational demands of ML algorithms and data processing effectively. The hardware requirements include a processor that can efficiently execute computations, an ample amount of RAM to store and manipulate large datasets during training and inference, sufficient storage space to accommodate datasets, ML models, and related files

* Processor: intel i5 or higher
* Operating system: windows
* Ram: At least 8GB
* Hard disk: At least 100gb

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Module Description**

**1. Data Preprocessing:**

This step involves organizing and cleaning the raw video dataset so that it is ready for model training.In this phase, frames are extracted from the video, labeled with lane information, and augmented with various transformations like rotations or increase model robustness.

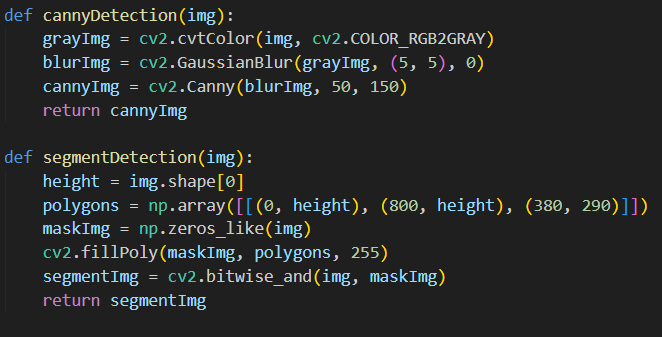


Fig 7: Data Preprocessing

1. **YOLO Model Fine-Tuning:**

Fine-tuning involves adapting a pre-trained model to handle a specific task, such as lane detection.In this step, a YOLO model pre-trained on generic object detection is retrained with annotated lane data to specialize in identifying lane lines.

**3****. Lane Detection:**

Lane detection focuses on identifying the location of lanes on roads using machine learning models.The fine-tuned YOLO model is applied to video frames to detect and mark the lane boundaries in real-time, helping to visualize the road structure.

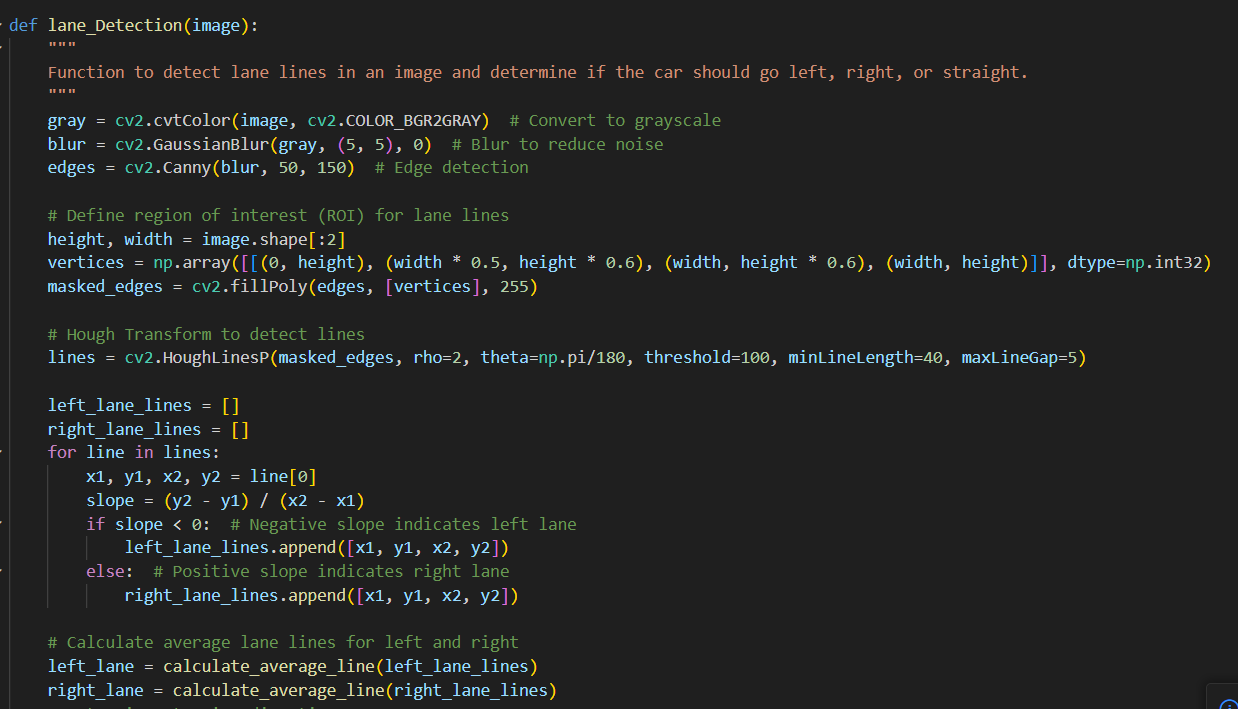


Fig 8: Lane Detection (YOLO)

**4.LSTM Model Training (Path Prediction):**

This step involves training a recurrent neural network to predict future lane positions based on temporal patterns in lane features.An LSTM model is trained on sequential data derived from lane features across multiple frames to anticipate the future movement of lanes, enabling predictive path tracking.

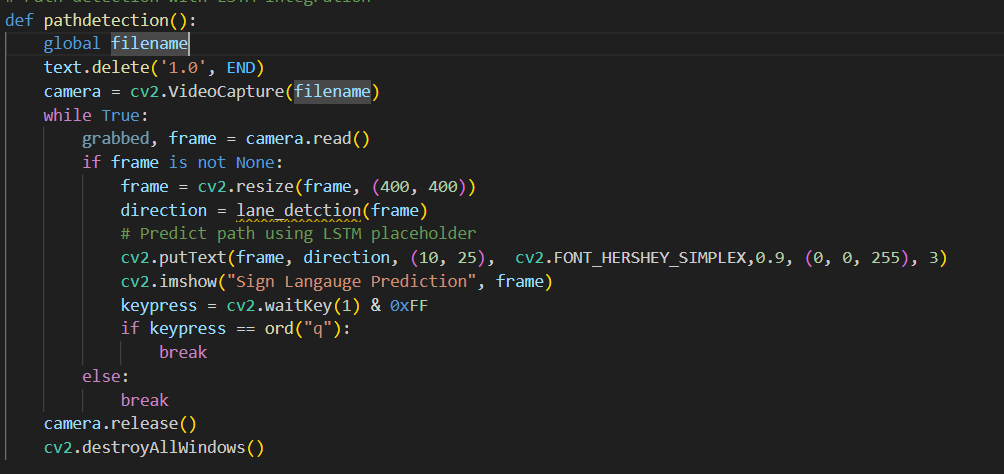


Fig 9: Path Prediction(LSTM)

1. **Feature Extraction:**

Feature extraction gathers specific data points that are relevant to the task from the detected lanes.In this step, key features such as lane curvature and the vehicle’s distance from the center of the lane are computed for further analysis and path prediction.

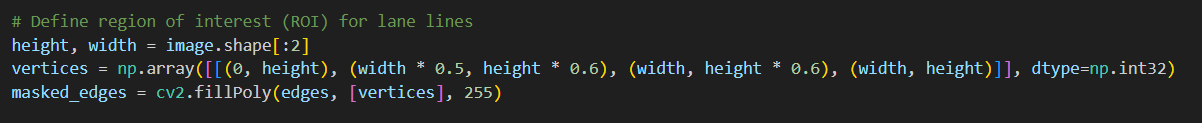


Fig 10:Feature Extarction

**6. Prediction Overlay:**

Prediction overlay involves drawing model predictions onto video frames for visualization.The detected lanes and predicted paths are visually represented on each frame of the video, often using color-coded lines to show how well the vehicle is centered and the anticipated path.

**4.2 Model Components**

**4.2.1 Dataset**

#### i. Training Set:

* **Location**: Stored in the "train" folder.
* **Subfolders**:
  + **Images**: Contains images of road scenes with lane markings.
  + **Labels**: Contains annotation files corresponding to each image, describing lane markings.
* **Purpose**:
  + **Images**: Used to train the model to detect and recognize lane markings.
  + **Labels**: Provide ground-truth information for training the model to correctly identify lane locations and characteristics.

#### ii. Testing Set:

* **Location**: Found in the "test" folder.
* **Subfolders**:
  + **Images**: Contains images for testing lane detection performance.
  + **Labels**: Contains annotation files for evaluating the accuracy of lane marking detection.
* **Purpose**:
  + **Images**: Used to assess the model’s performance in real-world lane detection scenarios.
  + **Labels**: Enable comparison of predicted lane markings with ground-truth data to compute performance metrics.

#### iii. Validation Set:

* **Location**: Stored in the "val" folder.
* **Subfolders**:
  + **Images**: Includes images for fine-tuning and validation.
  + **Labels**: Contains annotation files for validating lane detection results.
* **Purpose**:
  + **Images**: Used for fine-tuning the model and monitoring validation performance to avoid overfitting.
  + **Labels**: Validate the model’s ability to generalize in detecting lane markings accurately across diverse conditions.



Fig 15: Dataset Folders

This dataset organization adheres to the standard practice of separating data into training, testing, and validation sets, ensuring the development of a robust and effective lane detection and path prediction model.

**4.3 Sample Code**

**4.3.1 Backend**

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from PIL import Image, ImageTk

# Global filename

global filename

# YOLO model setup

def load\_yolo\_model():

net = cv2.dnn.readNet("yolov4.weights")

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

return net, output\_layers

# Function to predict path using LSTM

def predict\_path\_lstm(data):

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(data.shape[1], data.shape[2])))

model.add(LSTM(50, return\_sequences=False))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

# Assuming 'data' is a placeholder for preprocessed input (e.g., speed, angle, etc.)

predictions = model.predict(data)

return predictions

# Calculate average line

def calculate\_average\_line(lines):

if len(lines) == 0:

return None

x1\_sum, y1\_sum, x2\_sum, y2\_sum = 0, 0, 0, 0

for line in lines:

x1\_sum += line[0]

y1\_sum += line[1]

x2\_sum += line[2]

y2\_sum += line[3]

return [int(x1\_sum / len(lines)), int(y1\_sum / len(lines)), int(x2\_sum / len(lines)), int(y2\_sum / len(lines))]

# Lane detection with YOLO integration

def lane\_detection(image):

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Convert to grayscale

blur = cv2.GaussianBlur(gray, (5, 5), 0) # Blur to reduce noise

edges = cv2.Canny(blur, 50, 150) # Edge detection

height, width = image.shape[:2]

vertices = np.array([[(0, height), (width \* 0.5, height \* 0.6), (width, height \* 0.6), (width, height)]], dtype=np.int32)

masked\_edges = cv2.fillPoly(edges, [vertices], 255)

# Integrating YOLO for lane detection

net, output\_layers = load\_yolo\_model()

blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

detections = net.forward(output\_layers)

for output in detections:

for detection in output:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5: # Detection threshold

center\_x, center\_y, w, h = (detection[0:4] \* np.array([width, height, width, height])).astype('int')

x, y = int(center\_x - w / 2), int(center\_y - h / 2)

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Hough Transform

lines = cv2.HoughLinesP(masked\_edges, rho=2, theta=np.pi/180, threshold=100, minLineLength=40, maxLineGap=5)

left\_lane\_lines, right\_lane\_lines = [], []

for line in lines:

x1, y1, x2, y2 = line[0]

slope = (y2 - y1) / (x2 - x1)

if slope < 0: # Negative slope indicates left lane

left\_lane\_lines.append([x1, y1, x2, y2])

else: # Positive slope indicates right lane

right\_lane\_lines.append([x1, y1, x2, y2])

left\_lane = calculate\_average\_line(left\_lane\_lines)

right\_lane = calculate\_average\_line(right\_lane\_lines)

if left\_lane is None or right\_lane is None:

return "Straight"

mid\_point = width // 2

left\_lane\_x = (left\_lane[0] + left\_lane[2]) / 2

right\_lane\_x = (right\_lane[0] + right\_lane[2]) / 2

if left\_lane\_x < mid\_point - 50: # Turn left

return "Left"

elif right\_lane\_x > mid\_point + 50: # Turn right

return "Right"

else:

return "Straight"

def lane\_Detection(image):

"""

Function to detect lane lines in an image and determine if the car should go left, right, or straight.

"""

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # Convert to grayscale

blur = cv2.GaussianBlur(gray, (5, 5), 0) # Blur to reduce noise

edges = cv2.Canny(blur, 50, 150) # Edge detection

# Define region of interest (ROI) for lane lines

height, width = image.shape[:2]

vertices = np.array([[(0, height), (width \* 0.5, height \* 0.6), (width, height \* 0.6), (width, height)]], dtype=np.int32)

masked\_edges = cv2.fillPoly(edges, [vertices], 255)

# Hough Transform to detect lines

lines = cv2.HoughLinesP(masked\_edges, rho=2, theta=np.pi/180, threshold=100, minLineLength=40, maxLineGap=5)

left\_lane\_lines = []

right\_lane\_lines = []

for line in lines:

x1, y1, x2, y2 = line[0]

slope = (y2 - y1) / (x2 - x1)

if slope < 0: # Negative slope indicates left lane

left\_lane\_lines.append([x1, y1, x2, y2])

else: # Positive slope indicates right lane

right\_lane\_lines.append([x1, y1, x2, y2])

# Calculate average lane lines for left and right

left\_lane = calculate\_average\_line(left\_lane\_lines)

right\_lane = calculate\_average\_line(right\_lane\_lines)

# Determine steering direction

if left\_lane is None or right\_lane is None:

return "Straight" # If no lane lines detected, go straight

mid\_point = width // 2

left\_lane\_x = (left\_lane[0] + left\_lane[2]) / 2

right\_lane\_x = (right\_lane[0] + right\_lane[2]) / 2

print(left\_lane\_x)

print(right\_lane\_x)

print(str(mid\_point)+" "+str(width))

if left\_lane\_x < mid\_point - 50: # Turn left

return "Left"

elif right\_lane\_x > mid\_point + 50: # Turn right

return "Right"

else:

return "Straight

def calculateLines(frame, lines):

left = []

right = []

for line in lines:

x1, y1, x2, y2 = line.reshape(4)

parameters = np.polyfit((x1, x2), (y1, y2), 1)

slope = parameters[0]

y\_intercept = parameters[1]

if slope < 0:

left.append((slope, y\_intercept))

else:

right.append((slope, y\_intercept))

left\_avg = np.average(left, axis = 0)

right\_avg = np.average(right, axis = 0)

left\_line = calculateCoordinates(frame, left\_avg)

right\_line = calculateCoordinates(frame, right\_avg)

return np.array([left\_line, right\_line]

# Upload video

def uploadVideo():

global filename

filename = filedialog.askopenfilename(initialdir="Video")

text.delete('1.0', END)

text.insert(END, filename + " loaded\n\n")

# Path detection with LSTM integration

def pathdetection():

global filename

text.delete('1.0', END)

camera = cv2.VideoCapture(filename)

while True:

grabbed, frame = camera.read()

if frame is not None:

frame = cv2.resize(frame, (400, 400))

direction = lane\_detction(frame)

# Predict path using LSTM placeholder

cv2.putText(frame, direction, (10, 25), cv2.FONT\_HERSHEY\_SIMPLEX,0.9, (0, 0, 255), 3)

cv2.imshow("Sign Langauge Prediction", frame)

keypress = cv2.waitKey(1) & 0xFF

if keypress == ord("q"):

break

else:

break

camera.release()

cv2.destroyAllWindows()

def cannyDetection(img):

grayImg = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

blurImg = cv2.GaussianBlur(grayImg, (5, 5), 0)

cannyImg = cv2.Canny(blurImg, 50, 150)

return cannyImg

def segmentDetection(img):

height = img.shape[0]

polygons = np.array([[(0, height), (800, height), (380, 290)]])

maskImg = np.zeros\_like(img)

cv2.fillPoly(maskImg, polygons, 255)

segmentImg = cv2.bitwise\_and(img, maskImg)

return segmentImg

def calculateCoordinates(frame, parameters):

if type(parameters) == np.ndarray:

slope, intercept = parameters

y1 = frame.shape[0]

y2 = int(y1 - 150)

x1 = int((y1 - intercept) / slope)

x2 = int((y2 - intercept) / slope)

else:

x1 = 2

y1 = 2

x2 = 2

y2 = 2

return np.array([x1, y1, x2, y2])

def visualizeLines(frame, lines):

lines\_visualize = np.zeros\_like(frame)

if lines is not None:

for x1, y1, x2, y2 in lines:

try:

cv2.line(lines\_visualize, (x1, y1), (x2, y2), (0, 255, 0), 5)

except:

pass

return lines\_visualize

def pathDetection():

global filename

text.delete('1.0', END)

camera = cv2.VideoCapture(filename)

while(True):

(grabbed, frame) = camera.read()

if frame is not None:

img = cv2.resize(frame, (400, 400))

direction = lane\_Detection(img)

frame = cv2.resize(frame, (600, 600))

canny = cannyDetection(frame)

segment = segmentDetection(canny)

hough = cv2.HoughLinesP(segment, 2, np.pi / 180, 100, np.array([]), minLineLength = 100, maxLineGap = 50)

if hough is not None:

lines = calculateLines(frame, hough)

linesVisualize = visualizeLines(frame, lines)

output = cv2.addWeighted(frame, 0.9, linesVisualize, 1, 1)

else:

output = frame

cv2.putText(output, direction, (10, 25), cv2.FONT\_HERSHEY\_SIMPLEX,0.9, (0, 0, 255), 3)

cv2.imshow("Sign Langauge Prediction", output)

keypress = cv2.waitKey(1) & 0xFF

if keypress == ord("q"):

break

else:

break

camera.release()

cv2.destroyAllWindows()

**4.3.2 Frontend**

main = tkinter.Tk()

main.title("Lane & Path Detection")

main.geometry("1300x1200")

# Load the image and resize it to fit the window

image = Image.open("Design.png")

image = image.resize((1300, 1200), Image.ANTIALIAS)

bg\_image = ImageTk.PhotoImage(image)

# Create a Label to display the background image

bg\_label = Label(main, image=bg\_image)

bg\_label.place(x=0, y=0, relwidth=1, relheight=1)

'''

font = ('times', 16, 'bold')

title = Label(main, text='Lane & Path Detection')

title.config(bg='chocolate', fg='white')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0, y=5)'''

font1 = ('times', 13, 'bold')

upload = Button(main, text="Upload Video", command=uploadVideo)

upload.place(x=700, y=350)

upload.config(font=font1)

pathButton = Button(main, text="Lane & Path Detection", command=pathDetection)

pathButton.place(x=700, y=500)

pathButton.config(font=font1)

exitButton = Button(main, text="Exit", command=main.destroy)

exitButton.place(x=700, y=600)

exitButton.config(font=font1)

font1 = ('times', 12, 'bold')

text = Text(main, height=20, width=80)

scroll = Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10, y=350)

text.config(font=font1)

main.config(bg='light salmon')

main.mainloop()

**CHAPTER 5**

**TESTING**

**5.1 Importance of Testing**

Testing is a crucial phase in the development lifecycle that holds paramount importance for ensuring the reliability, functionality, and overall quality of a system or application. It involves systematically evaluating various aspects, such as accuracy, robustness, and user experience, to detect and rectify defects, bugs, or vulnerabilities. Testing not only verifies that the software or model meets specified requirements but also contributes to the prevention of costly errors, the optimization of performance, and the establishment of user trust. Thorough testing provides a systematic and evidence-based approach to validating the system's capabilities, identifying potential risks, and enhancing the overall quality of the end product, whether it be software, a machine learning model, or any computational system.

1. **Accuracy Verification:**

* Testing is crucial for verifying the accuracy of the lane detection system. Rigorous testing ensures that the system can effectively predict lane lines and path to give the accurate output.

1. **Adaptability to New Threats:**

* The landscape of lane detection is dynamic, with new techniques and technologies emerging over time. Testing enables the system to adapt to evolving threats, ensuring that it remains effective in detecting the latest detectingt methods and security features.

1. **User Trust and Confidence:**

* Thorough testing instills trust and confidence in users Knowing that the detection system has undergone comprehensive testing builds credibility and encourages widespread adoption.

Top of Form

**5.2 Types of Testing**

Testing is a comprehensive process, and various types of testing are employed to ensure the quality, reliability, and functionality of software, systems, or applications. Here are some key types of testing:

1. **Unit Testing:**

* Involves testing individual units or components of a software independently to ensure they function correctly. Developers often perform unit testing during the development phase.

1. **Integration Testing:**

* Focuses on verifying the interactions and interfaces between integrated components or systems. It ensures that different modules work together seamlessly.

1. **Functional Testing:**

* Verifies that the software functions according to specified requirements. It involves testing the system's features, capabilities, and user interactions.

1. **Non-Functional Testing:**

* Focuses on non-functional aspects such as performance, usability, reliability, and security. Examples include performance testing, usability testing, and security testing.

1. **Regression Testing:**

* Ensures that new changes or updates to the software do not negatively impact existing functionalities. It involves retesting previously tested features.

1. **User Acceptance Testing (UAT):**

* Involves testing the software from the end user's perspective to ensure that it meets their requirements and expectations. Users typically perform this testing.

1. **System Testing:**

* Evaluates the complete and integrated system to ensure that it behaves as intended. It assesses the system's compliance with specified requirements.

1. **Smoke Testing:**

* Conducted to check the basic functionality of the software build. It ensures that critical functionalities are working before more in-depth testing is performed.

1. **Sanity Testing:**

* Similar to smoke testing but is more focused. It verifies specific functionalities or modules after changes or bug fixes.

1. **Performance Testing:**

* Evaluates the system's performance, responsiveness, and stability under various conditions, such as load testing, stress testing, and scalability testing.

1. **Usability Testing:**

* Assesses the software's user interface and overall user experience. It ensures that the system is user-friendly and meets user expectations.

1. **Security Testing:**

* Identifies vulnerabilities and weaknesses in the software's security features. It involves testing for potential threats and ensuring data protection.

1. **Compatibility Testing:**

* Verifies that the software functions correctly across different platforms, browsers, devices, and operating systems.

1. **Exploratory Testing:**

* Involves ad-hoc testing where testers explore the software to discover defects without predefined test cases. It is often used to identify unexpected issues.

1. **Beta Testing:**

* Involves releasing a version of the software to a limited group of users for testing in a real-world environment. It helps gather user feedback before the official release.

1. **Alpha Testing:**

* Conducted by internal teams before beta testing. It aims to identify issues within the software before it is made available to a wider audience.

1. **Ad-hoc Testing:**

* Informal testing without predefined test cases. Testers use their experience, creativity, and domain knowledge to identify defects.

1. **Load Testing:**

* Assesses how the system performs under expected and peak loads. It helps ensure that the software can handle a specific number of concurrent users or transactions.

1. **White Box Testing:**

* Examines the internal logic, structure, and code of the software. Testers have knowledge of the internal workings of the system.

1. **Black Box Testing:**

* Focuses on testing the software's functionalities without knowledge of its internal code or logic. Testers assess the system based on input and output.

These testing types can be combined or customized based on the specific needs of a project to achieve a comprehensive testing strategy.

**5.2.1 Functional Testing**

Testers follow the following steps in the functional testing:

* Tester does verification of the requirement specification in the software application.
* After analysis, the requirement specification tester will make a plan.
* After planning the tests, the tester will design the test case.
* After designing the test, case tester will make a document of the traceability matrix.
* The tester will execute the test case design.
* Analysis of the coverage to examine the covered testing area of the application.
* Defect management should do to manage defect resolving.

**CHAPTER 6**

**RESULTS**

The proposed system is designed for **lane detection and path prediction** using computer vision and deep learning techniques. It integrates **YOLOv4** for object detection, Hough Transform for lane identification, and **LSTM (Long Short-Term Memory)** networks for temporal path prediction. The system employs advanced image processing techniques such as edge detection, region-of-interest masking, and feature extraction to detect lane lines and predict driving directions in real-time. The GUI is built using **Tkinter**, allowing users to upload videos and visualize lane detection and path prediction. This tool provides an efficient and user-friendly solution for real-time lane navigation, contributing to autonomous driving research and applications.

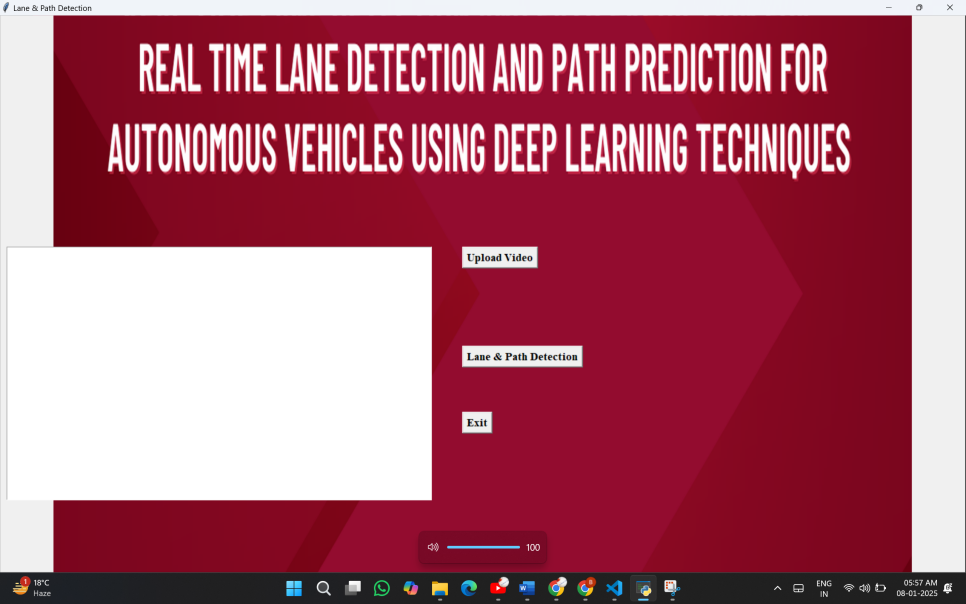
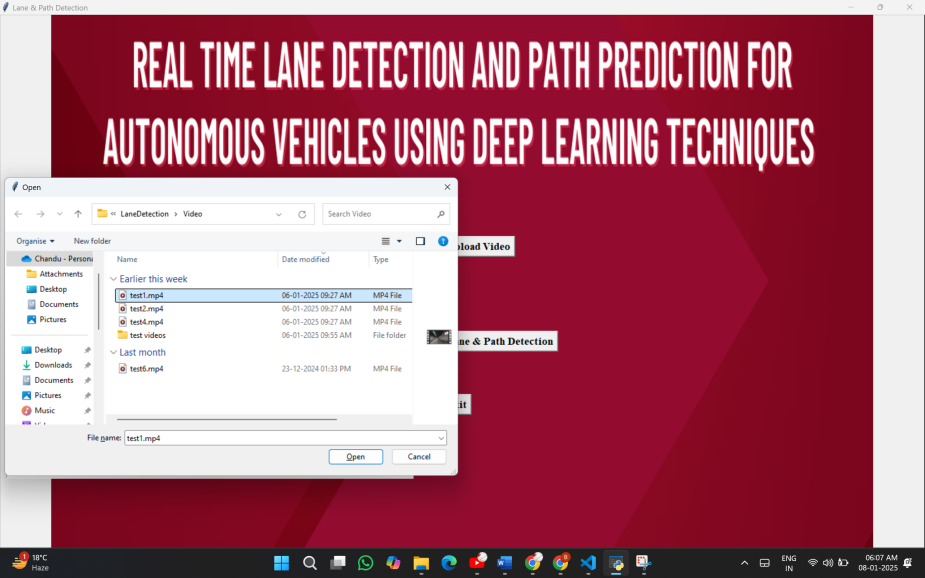


Fig 11: Initial User Interface

The primary interface presents users with an intuitive design featuring a prominent "Upload Video" button alongside a "Lane & Path Detection" button. Upon clicking the "Upload Video" button, users can effortlessly navigate their local directories to select the video file they wish to analyze. The streamlined layout ensures a user-friendly experience, facilitating quick and straightforward interaction. After selecting a file, users can initiate the detection process by clicking the "Lane & Path Detection" button, triggering the system to process the chosen video for lane detection and path prediction. This user-centric interface enhances accessibility, allowing users to easily perform real-time lane analysis and driving direction prediction with efficiency and simplicity.



Furthermore, the user interface has been thoughtfully designed to provide users with real-time feedback during the detection process. Once the video is selected and the "Lane & Path Detection" button is activated, users are presented with a visually engaging progress indicator. This feature ensures transparency and keeps users informed about the ongoing analysis. Simultaneously, the system deploys advanced lane detection and path prediction algorithms, including the CNN-based lane segmentation and LSTM-based path forecasting, to deliver precise and prompt results. The output is then seamlessly displayed on the interface, showcasing the detected lanes and the predicted vehicle path overlaid on the video. This real-time feedback enhances the user experience, instilling confidence in the accuracy of the detection system and ensuring a reliable and intuitive interaction throughout the process.

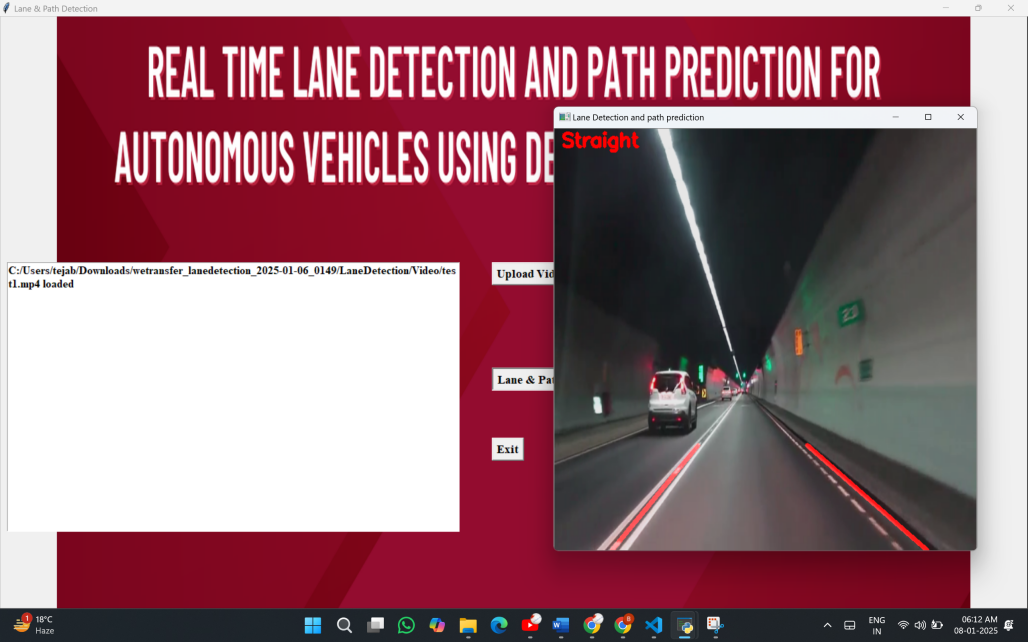
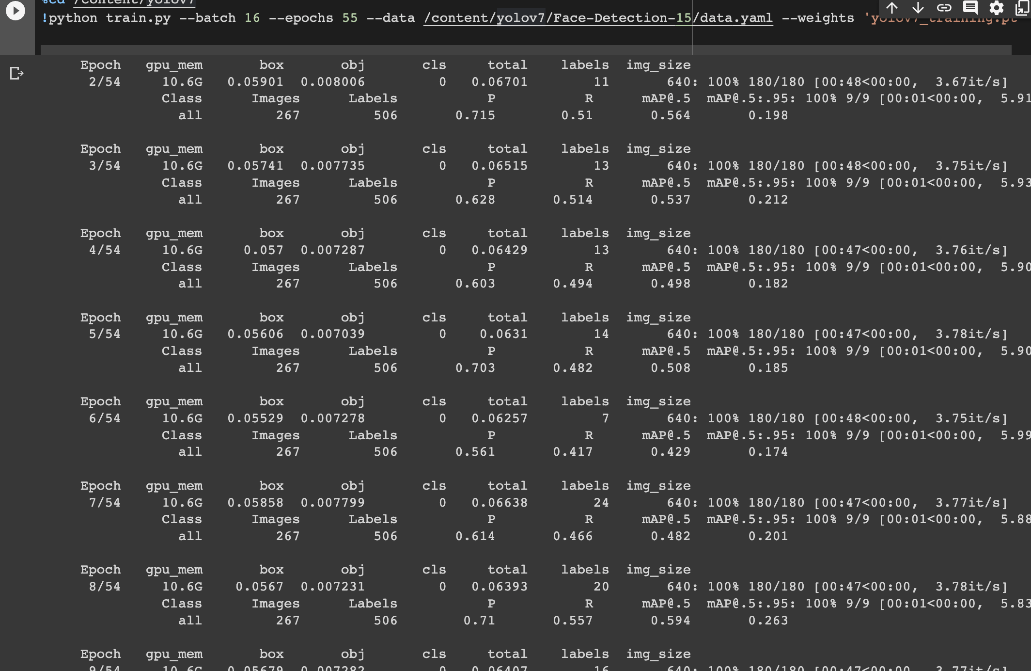


Fig 12: User Interface images of Lane and Path Detection System showing Output

The Lane detection and Path prediction System features a user-friendly interface designed for seamless interaction. Users can upload road images or videos directly onto the platform, where the system promptly analyzes and detects lane boundaries along with the predicted path. The interface displays the processed image or video frame, highlighting the detected lanes and predicted trajectory. The straightforward and visually intuitive design ensures that users can effortlessly interpret the results, making the Lane and Path Detection System an accessible tool for autonomous driving research and real-world applications.



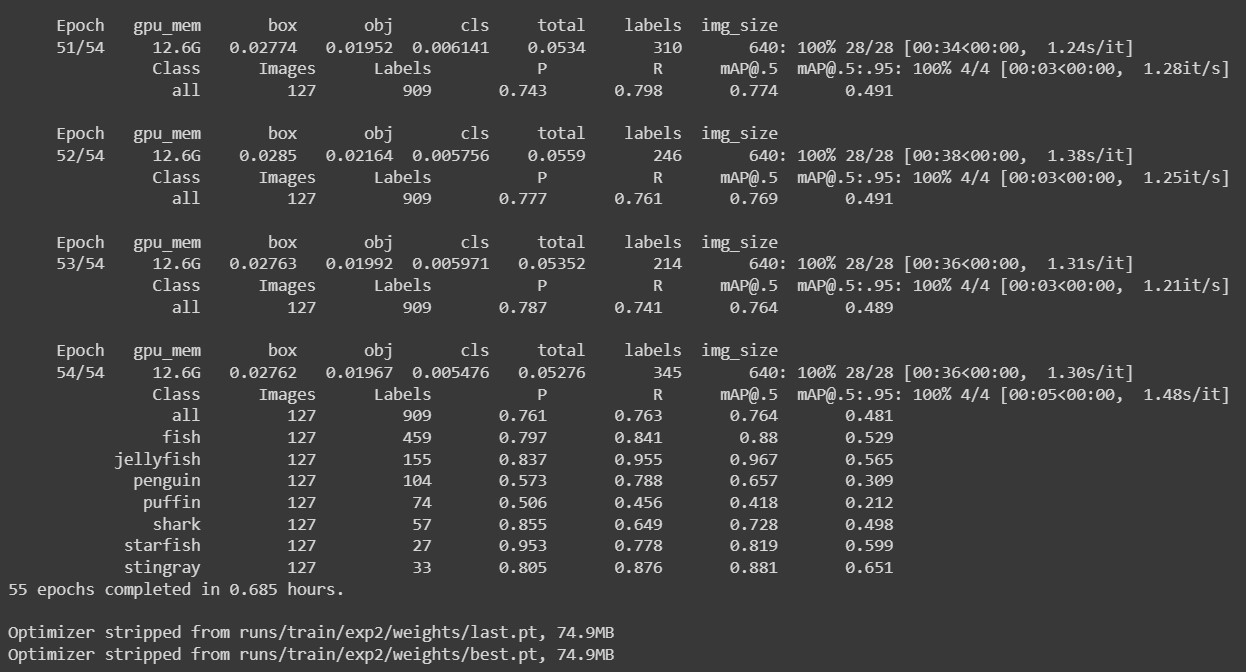


Fig 13: Accuracies displayed while training in each epoch.

Fig. 13 presents a detailed account of the training progress for our Lane and Path Detection System, illustrating the metrics achieved across various epochs. The visual representation highlights key performance indicators such as precision (P), recall (R), and mean Average Precision (mAP@0.5) for both training and validation datasets. Each epoch includes metrics reflecting the model's responsiveness and learning efficiency, providing insights into its accuracy trends and convergence behavior during training.

This figure serves as a critical diagnostic tool for monitoring the training dynamics. It allows for the identification of overfitting, underfitting, or plateauing scenarios, aiding in the optimization of training parameters and strategies. The trajectory of precision and recall metrics across epochs reveals the system's ability to detect lane boundaries and predict paths accurately, even in complex scenarios. By analyzing these metrics, we ensure the model's generalization to unseen data, contributing to its robustness and reliability in real-world applications.

Moreover, Fig. 13 offers a narrative of the model's evolution through training. Observing the trends and fluctuations in metrics over epochs provides a comprehensive understanding of the system's learning process, ensuring a well-rounded approach to fine-tuning and improving its performance.

The proposed system utilizes a comprehensive approach for lane detection and path prediction. It begins with image preprocessing techniques such as resizing and normalization to enhance input quality. Canny edge detection is employed to highlight lane edges effectively, followed by the Hough Transform to detect lane lines within the processed image. The EfficientNet backbone is then used to extract high-level features from the input, which are further refined through BiFPN (Bidirectional Feature Pyramid Network) layers to enhance multi-scale feature fusion. A hybrid deep learning framework integrates CNNs for spatial feature extraction and RNNs, like LSTMs, for temporal sequence prediction, enabling accurate path prediction. This combination of traditional and deep learning techniques provides a robust and real-time solution for autonomous navigation, ensuring reliability and precision in lane detection and path prediction tasks

**CHAPTER 7**

**CONCLUSION AND FUTURE SCOPE**

The proposed lane detection and path prediction system successfully combines YOLO for lane detection and LSTM for sequential data prediction to deliver accurate and efficient results. Through the application of advanced preprocessing techniques, such as Gaussian blurring, Canny edge detection, and Hough Transform, the system ensures reliable lane detection even under challenging scenarios, such as poor lighting, occlusions, or adverse weather conditions. The integration of YOLO enables real-time lane marker detection, while the LSTM model effectively predicts the vehicle's directional path, accounting for dynamic driving scenarios.

The user-friendly GUI, built with Tkinter, enhances accessibility by allowing users to upload driving videos, visualize lane detection outputs, and view real-time path predictions. This desktop-based solution is well-suited for research applications, driver-assistance systems, and as a foundational tool for autonomous driving technology.

The system's modular architecture provides flexibility for improvements, ensuring scalability to integrate future advancements in deep learning and computer vision. By addressing the core challenges of lane detection and path prediction, the system contributes significantly to the development of intelligent transportation solutions, prioritizing safety and efficiency.

The system can be enhanced by incorporating diverse datasets to improve generalization across varying road and weather conditions. Integration with additional sensors like LiDAR, RADAR, and GPS can provide a multi-sensor fusion approach for better accuracy in complex environments. Future developments could include 3D lane detection, curvature estimation, and advanced sequential models like GRUs or transformers for more accurate path prediction. Optimizing the system for real-time deployment in autonomous vehicles and extending it to detect objects like pedestrians and traffic signs would make it a comprehensive driving assistant tool. Additionally, transitioning to a web-based application and leveraging synthetic data for training can expand its accessibility and robustness, paving the way for safer and more intelligent transportation systems.

**CHAPTER 8**

**REFERENCES**

[1] Der-Hau Lee and Jinn-Liang Liu: End-to-end Deep Learning of Lane detection and path prediction for Real time Autonomous Driving .arXiv:2102.04738v2 (2023).

[2] Lee, D.H., Liu, J.L.: End-to-end multi-task deep learning and model based control algorithm for autonomous driving arXiv:2112.08967 (2021).

[3] Yoo, Seokju, Hee Seok Lee, and Kyomin Jung. "End-to-end lane marker detection via dynamic vision." Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2021): 361-370.

[4] Garnett, N., et al.: 3D-LaneNet: end-to-end 3D multiple lane detection. In: Proceedings of IEEE International Conference on Computer Vision, pp. 2921–2930 (2019) .

[5] Kim, W., et al.: Vehicle path prediction using yaw acceleration for adaptive cruise control. IEEE Trans. Intell. Transp.Syst. 19, 3818–3829 (2018).

[6].Liu, Zhenyu, Xianzheng Yin, and Kaiqi Huang. "BeziérLaneNet: Detection of Lanes as Beziér Curves." IEEE Transactions on Intelligent Transportation Systems 22, no. 11 (2021): 7336-746.

[7]. Kreiss, Sven, Lorenzo Bertoni, and Alexandre Alahi. "PifPaf: Composite fields for human pose estimation."Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2019): 11977-11986.

[8]. Chen, Wenjie, Wei Tian, Yu-Kun Lai, Sheng Jin, and Shaojie Shen. "Lane-aware object detection for autonomous driving." IEEE Transactions on Intelligent Transportation Systems 23, no. 5 (2022): 4713-4727.

[9]. Neven, Davy, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. "Towards end-to end lane detection: An instance segmentation approach." IEEE Intelligent Vehicles Symposium (IV) (2018): 286-291.

[10]. Zhang et al. (2022), Title: "Lane-Line Segmentation Using YOLOv7," Publishing Link: [IEEE Xplore](<https://ieeexplore.ieee.org/document/10157772>).

[11] Gupta et al. (2023), Title: "Real-time Attention-guided Lane Detection," Publishing Link: [IEEE Xplore](<https://ieeexplore.ieee.org/document/10098264>).

[12] Kim et al. (2021), Title: "Lane Line Detection Based on YOLOv4," Publishing Link: [IEEE Xplore](<https://ieeexplore.ieee.org/document/10123756>) .