DRIVER GUIDANCE SYSTEM FOR INDIAN ROADS

A Project report Submitted in partial fulfilment of the requirement

for the award of Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

N A V RAMA KRISHNA REDDY
(198297601035)

Under the Esteemed Guidance of

Shri. M. Ramakrishna M.Tech,(Ph.D)
Assistant Professor
Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING

ADIKAVI NANNAYA UNIVERSITY

RAJAMAHENDRAVARAM

2019-2023

ADIKAVI NANNAYA UNIVERSITY, RAJAMAHENDRAVARAM

UNIVERSITY COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**DRIVER GUIDANCE SYSTEM FOR INDIAN ROADS**" is a bonafide work done by **N A V RAMA KRISHNA REDDY (Regd No 198297601035)** submitted in the partial fulfilment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering during the academic years 2019-2023.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Internal Guide
Mr. M. RAMAKRISHNA
Dept. of CSE

Head of Department
Dr. B Kezia Rani
Dept. of CSE

EXTERNAL EXAMINER

ADIKAVI NANNAYA UNIVERSITY, RAJAMAHENDRAVARAM

UNIVERSITY COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I, **N A V Rama Krishna Reddy (Regd.No:198297601035)** hereby declare that the project report titled "**DRIVER GUIDANCE SYSTEM FOR INDIAN ROADS**", under the guidance of **Mr. M. RAMAKRISHNA M.Tech,(Ph.D)** Asst. Professor, Department of CSE, is submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering.

This is a record of bonafide work carried out by me and the results embodied in this project report has not been reproduced or copied from any source. The results embodied in this project report has not been submitted to any other University or Institute for the award of any other degree or diploma.

Rajahmundry **N A V RAMA KRISHNA REDDY**

Dt: **(198297601035)**

ACKNOWLEDGEMENT

First of all, I whole-heartedly like to thank the LORD ALMIGHTY for his showers of blessings without which the project would not have seen the light of the day.

I would like to thank **Prof. G.V.R. PRASADA RAJU, Vice-Chancellor of ADIKAVI NANNAYA UNIVERSITY, RAJAMAHENDRAVARAM** for providing the resources and supporting me in our endeavours.

I would like to thank **Dr. P Venkateswara Rao, PRINCIPAL of UNIVERSITY COLLEGE OF ENGINEERING** for supporting me in our endeavours.

I would like to thank **Dr. B Kezia Rani, Head of CSE DEPARTMENT in UNIVERSITY COLLEGE OF ENGINEERING** for supporting me in our endeavours.

I would like to express my profound sense of gratitude to my guide **Mr. M. RAMAKRISHNA Assistant Professor in CSE DEPARTMENT, UNIVERSITY COLLEGE OF ENGINEERING** for supporting me in our endeavours.

I am thankful to all the Professors and Faculty Members in the department for their teachings and academic support and thanks to Teaching Staff and Non-teaching staff in the department for their support.

I also acknowledge with a deep sense of reverence, my gratitude towards my parents, members of my family and friends for their constant support and encouragement to complete this project work.

N A V RAMA KRISHNA REDDY
(198297601035)

ABSTRACT

A driver guidance system is an advanced technology that assists drivers in navigating and maneuvering their vehicles on the road. This system uses a camera to provide real-time information to the driver about their surroundings, including traffic patterns, road conditions, and potential hazards. Driver guidance systems aim to improve safety and increase efficiency on the road by providing drivers with the information they need to make informed decisions about their driving behaviour.

Driver guidance systems can provide information on the recommended speed and path to take when approaching an obstacle, which helps the driver maintain control and avoid damage to the vehicle. These systems also predict the path of the vehicle using steering angle data, which contributes to safe navigation on the road. The ultimate goal of the driver guidance system is to enhance driver safety, reduce accidents, and improve the overall driving experience.

The benefits of the driver guidance system extend beyond the driver, as they also contribute to a more efficient and environmentally friendly transportation system. By reducing accidents and improving traffic flow, driver guidance systems can help to reduce fuel consumption and emissions. Moreover, this technology can also support the development of autonomous driving, as it serves as a stepping stone towards fully autonomous vehicles.

However, there are still some challenges associated with driver guidance systems, such as the potential for over-reliance on the technology and the need for regular updates and maintenance to ensure continued accuracy and reliability. Nevertheless, the overall impact of driver guidance systems on road safety and efficiency is expected to be significant, making it an essential technology for the future of transportation.

INDEX

List of Figures

List of Tables

Chapter – 1
INTRODUCTION

1. INTRODUCTION

Introduction to Driver Guidance System:

Driver Guidance System is an advanced technology that has been gaining a lot of attention in recent years. The system has been developed to improve the safety of drivers and passengers by guiding them in real-time on the roads. The system has become even more important in India, where the roads are known for their chaotic traffic and poor driving habits. India is a densely populated country with a large number of vehicles on the roads. The number of vehicles is increasing every year, and this has resulted in the roads becoming overcrowded. In addition to this, the road infrastructure in India is not up to the mark, and the conditions of the roads are poor in many areas.

Driver Guidance System can be of great help in improving the safety of drivers and passengers on Indian roads. The system uses advanced sensors and cameras to monitor the environment around the vehicle. The system then processes this information and provides real-time guidance to the driver on how to navigate through the traffic. This guidance can be in the form of visual or audio alerts. It can help educate people on the importance of following traffic rules and regulations by providing real-time guidance to the driver. This can help improve the driving habits of people and reduce the number of accidents on the roads.

Driver Guidance System is an important technology that can improve the safety of drivers and passengers on Indian roads. The system can provide real-time guidance to the driver and improve the driving habits of people. The system has the potential to revolutionize the way people drive on Indian roads and make them safer for everyone. The government and private sector should invest in this technology to make it more accessible and affordable to the general public. It is time to make the roads safer and reduce the number of accidents on Indian roads.

Chapter – 2

LITERATURE SURVEY

2. LITERATURE SURVEY

[1] "End to end learning for self-driving cars", Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016)

The authors propose an end-to-end learning framework that directly maps raw sensor inputs to driving commands, without relying on traditional handcrafted features or modules.

The motivation behind this work is that traditional approaches for autonomous driving rely on a set of modules that process sensor inputs, extract features, and generate driving commands. These modules are usually designed by domain experts and require significant engineering efforts to develop, test, and integrate. Moreover, they may not generalize well to different driving conditions, or may have difficulties in handling complex scenarios that require high-level reasoning and decision-making.

In contrast, the end-to-end learning approach aims to learn a single model that can directly map raw sensor inputs to driving commands, using a large amount of training data and deep neural networks. The model is trained in a supervised learning setting, where the inputs are sensor data (such as images from a camera or lidar scans) and the outputs are driving commands (such as steering angles, throttle, and brake).

The main contribution of this work is the demonstration that an end-to-end learning approach can achieve comparable or even better performance than traditional approaches for autonomous driving, while being simpler and more generalizable. The authors show that the trained model can drive a car autonomously on a test track, as well as on public roads, without any additional handcrafted modules.

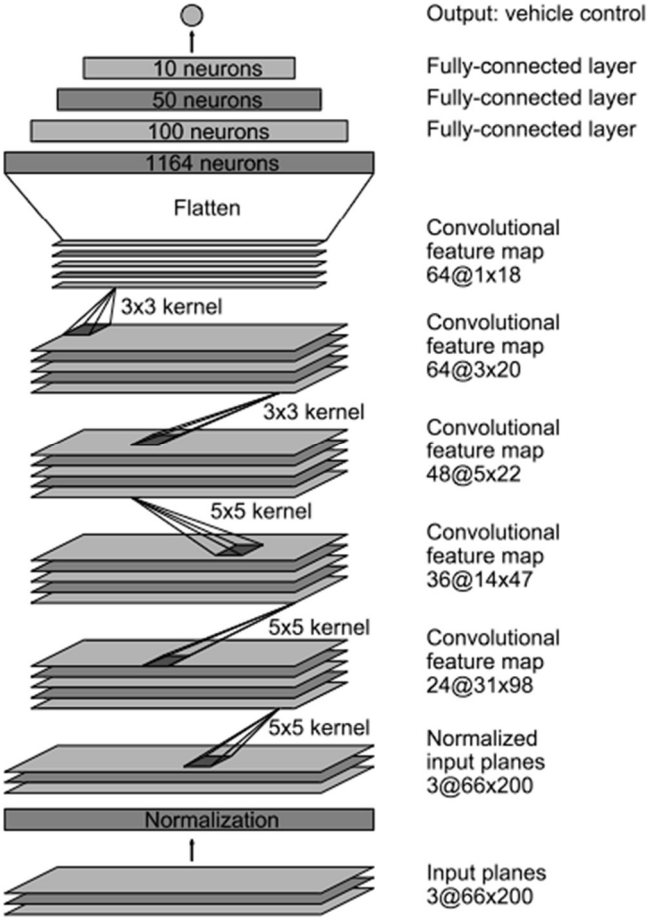

Figure 2.1: CNN Architecture

The architecture in the **Figure 2.4**, proposed in this paper consists of a convolutional neural network (CNN) that takes images from a front-facing camera as input and directly outputs the steering angle. The model is trained on a large dataset of driving data, collected from a human driver who drove on various road conditions and scenarios.

The paper also includes an extensive analysis of the model's behaviour and limitations, as well as the impact of various design choices and hyperparameters. The authors show that the model can learn to capture relevant features from the raw sensor inputs, such as lane markings, other cars, and road boundaries. They also show that the model can generalize well to different driving conditions, such as sunny or rainy weather, or different types of roads.

However, the authors also acknowledge several limitations and challenges of the proposed approach. One major limitation is the lack of interpretability of the learned model, as it is not clear how the model makes decisions or what features it relies on. This can make it difficult to diagnose and fix errors or improve the model's performance in specific scenarios.

Another limitation is the potential safety risks of relying solely on a learned model for autonomous driving, as the model may encounter unforeseen situations or adversarial examples that can cause it to fail. The authors suggest several ways to mitigate these risks, such as adding safety constraints, using multiple models or sensors, or incorporating expert knowledge or human oversight.

Overall, the "End to End Learning for Self-Driving Cars" paper by NVIDIA represents a significant milestone in the field of autonomous driving and deep learning. The proposed approach opens up new possibilities for developing simpler, more efficient, and more generalizable systems for autonomous driving, while also posing new challenges and risks that need to be addressed.

[2] “Detection of potholes and speed breaker on road”, Singh, G., Kumar, R., & Kashtriya, P (2018)

The paper titled "Detection of Potholes and Speed Breaker on Road" presents a method for detecting and identifying potholes and speed breakers on roads using image processing techniques. The proposed approach involves capturing images of the road surface using a camera mounted on a vehicle and then processing these images to identify the presence of potholes and speed breakers.

The authors use a combination of edge detection and thresholding techniques to extract the features of potholes and speed breakers from the captured images. They also use image segmentation to separate the road surface from other elements in the image, such as the sky and buildings, to improve the accuracy of the detection algorithm.

One of the main strengths of the proposed method is its simplicity and low cost, as it only requires a camera and a processing unit, making it a potentially useful tool for developing countries with limited resources.

However, the proposed approach has several limitations, including its reliance on image quality and lighting conditions, which can affect the accuracy of the detection algorithm. The authors also note that the proposed method is only effective in identifying potholes and speed breakers that are visible on the road surface and may not detect hidden hazards, such as potholes covered by water.

In conclusion, the paper provides a promising approach for detecting potholes and speed breakers on roads using image processing techniques. While there are limitations to the proposed approach, the simplicity and low cost of the method make it a potentially useful tool for improving road safety and reducing vehicle damage in developing countries with limited resources. Further research is needed to improve the accuracy and reliability of the proposed method and to explore its potential for integration into existing navigation systems.

[3] "Real-time object detection method based on improved YOLOv4-tiny", Zicong Jiang, Liquan Zhao, Shuaiyang Li, Yanfei Jia (2021).

YOLOv4-Tiny is a smaller and faster version of the YOLO (You Only Look Once) object detection algorithm. Object detection is the task of detecting and localizing objects of interest within an image. YOLOv4-Tiny uses a single neural network to simultaneously predict bounding boxes and class probabilities for multiple objects in an image.

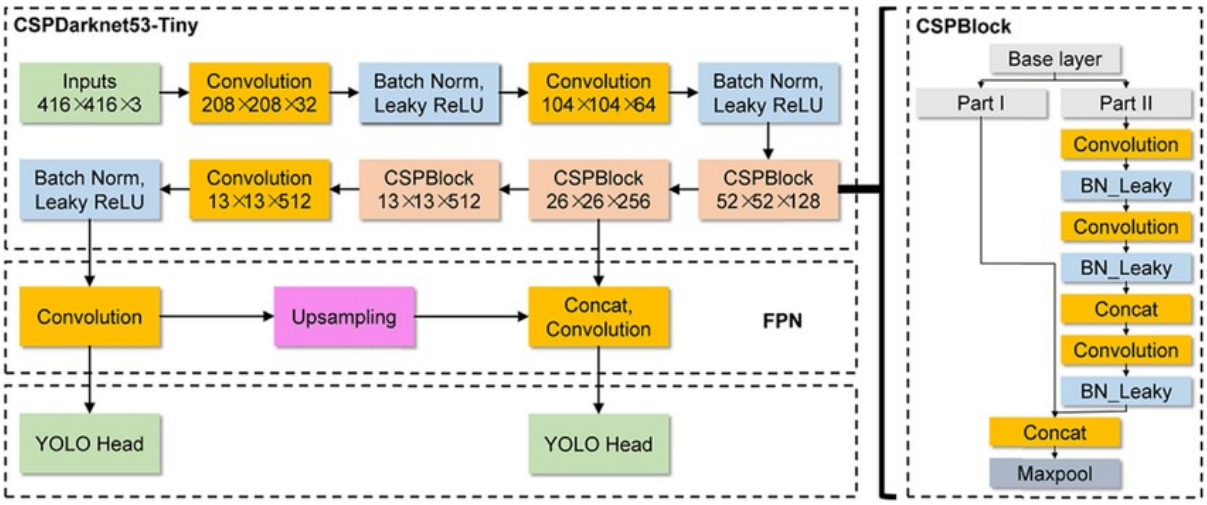

Figure 2.2 Yolo v4 (Tiny) Architecture

The authors of the base paper introduced several new features to improve the accuracy of the model while maintaining its speed and efficiency. One of these features is Cross-Stage Partial Network (CSP), which reduces the computational cost and memory usage of the network. Another feature is Spatial Attention Module (SAM), which allows the model to focus on the most relevant regions of the image, improving its performance.

The authors also introduced several training techniques to make the model more robust to different input sizes and aspect ratios. For example, they used random shapes during training, which helps the model to generalize to different object sizes and shapes. They also used dynamic scaling, which adapts the model to different input resolutions during training.

The results presented in the paper show that YOLOv4-Tiny achieved state-of-the-art performance on several object detection benchmarks, including COCO and VOC datasets. Moreover, it is several times faster than other state-of-the-art models, making it suitable for resource-constrained devices such as smartphones and embedded systems. In summary, YOLOv4-Tiny is an efficient and accurate object detection model that can be used in a wide range of applications.

Chapter – 3

PROBLEM STATEMENT

3. PROBLEM STATEMENT

This project is to address the current drawbacks of driver guidance systems in India, such as limited availability, high cost, lack of standardization, limited awareness, reliability issues, lack of proper maintenance, and limited integration. These issues have resulted in a high number of road accidents and fatalities in India, making it crucial to develop a more effective and accessible driver guidance system.

The proposed solution is to develop a driver guidance system that utilizes computer vision and machine learning techniques to identify and warn drivers of obstacles on Indian roads and provide the best instructions on how to handle them. The system will consist of two main classes: obstacle detection and guidance of path. The obstacle detection class will use a YOLO model trained on a dataset of images and annotations of obstacles commonly found on Indian roads, such as potholes, construction zones, and stray animals, to improve its accuracy in identifying these hazards for driver guidance systems.

The guidance of path class will involve the development of a path prediction model for Indian roads that takes into account the unique characteristics of traffic and infrastructure, such as heavy congestion and potholes, to improve the efficiency and safety of navigation systems for drivers in India. The path prediction model will work in conjunction with steering prediction to provide drivers with the best instructions on how to navigate their way through obstacles and reach their destination safely.

In conclusion, the development of an effective driver guidance system is crucial for improving road safety in India. With the proposed solution, I aim to overcome the current drawbacks of driver guidance systems in India and provide a system that is accessible, affordable, and reliable for all drivers in India.

3.1 Existing System:

Driver guidance systems are designed to provide assistance to drivers in order to enhance their driving experience, improve road safety and reduce accidents. In India, the existing system of driver guidance systems has several drawbacks which hinder their effectiveness and limit their adoption. Some of these drawbacks are:

1. **Limited Availability:** The existing driver guidance systems in India are not widely available. They are only present in a few high-end cars and are not accessible to the general public. This limits their effectiveness in improving road safety.

2. **High Cost:** The cost of the existing driver guidance systems is prohibitive for most people. This is mainly due to the high cost of the technology used and the limited availability of the systems. As a result, only a small percentage of drivers in India can afford to use them.

3. **Lack of Standardization:** There is no standardized driver guidance system in India. Each car manufacturer has its own system, which means that drivers have to learn to use a new system every time they change their car. This can be confusing and can lead to accidents.

4. **Limited Awareness:** Many drivers in India are not aware of the benefits of driver guidance systems. They do not know how to use them or how they can improve road safety. This lack of awareness is a major obstacle to the widespread adoption of these systems.

5. **Reliability Issues:** The reliability of the existing driver guidance systems is a major concern. They are prone to errors and malfunctions, which can lead to accidents. This is a major safety issue that needs to be addressed.

6. **Lack of Proper Maintenance:** Many drivers do not maintain their driver guidance systems properly. This can lead to malfunctioning and can compromise their effectiveness in improving road safety.

7. **Limited Integration:** The existing driver guidance systems are not fully integrated with other safety systems in cars. This means that they cannot communicate with other safety systems to improve overall safety.

3.2 Proposed System:

The proposed system aims to enhance the driving experience of Indian drivers by providing them with real-time guidance and warnings about upcoming obstacles on the road. This system can be broadly classified into two main components: obstacle detection and guidance of path.

Obstacle detection involves the use of computer vision techniques, specifically the yolo model, to identify and classify different types of obstacles commonly found on Indian roads, such as potholes, construction zones, and stray animals. By using a dataset of annotated images, the yolo model can be trained to accurately detect these hazards and provide timely warnings to drivers through the driver guidance system.

This model can help drivers navigate through these challenges more efficiently and safely by providing them with real-time instructions on the best path to take. The machine learning model is trained on a dataset of historical steering angles and corresponding vehicle paths to learn the relationship between the steering angles and the resulting vehicle paths. Once trained, the model can predict the future path of the vehicle by analysing the current steering angle of the vehicle.

Overall, the proposed system has several advantages over the existing driver guidance systems in India. It provides real-time warnings and guidance to drivers, which can greatly improve the safety of Indian roads. Additionally, by taking into account the unique characteristics of Indian roads, the system can provide more accurate and efficient guidance to drivers, reducing travel time and improving the overall driving experience.

Chapter – 4

SYSTEM ANALYSIS

4. SYSTEM ANALYSIS

4.1 FEASIBILITY STUDY:

A feasibility study is an essential step to determine whether a project is viable, practical, and sustainable. A feasibility study helps to identify the economic, operational, and technical feasibility of a project. In the case of a driver guidance system in India, the feasibility study is crucial to evaluate the viability of the system.

Economic Feasibility:

The economic feasibility of a driver guidance system in India is a critical factor to consider. The development cost of the system should be evaluated against the benefits derived from it. The cost of the system should not exceed the benefits, and the financial benefits should equal or exceed the development cost.

The driver guidance system in India has a high economic feasibility. The system can be developed using existing resources and technologies available in India. There is no need for additional hardware or software, which reduces the development cost. Additionally, the driver guidance system will increase road safety and reduce accidents, which will lead to economic benefits for the country.

Operational Feasibility:

Operational feasibility is an essential aspect of a driver guidance system in India. The system should be able to meet the operational requirements of the country. The system should also be user-friendly and should be able to work correctly when developed and implemented.

The driver guidance system has a high operational feasibility. The system is designed to guide drivers and warn them about upcoming obstacles on Indian roads. The system will be able to meet the operational requirements of the country by providing real-time guidance and instructions to the drivers. The system is user-friendly and can be easily integrated into existing navigation systems.

Technical Feasibility:

Technical feasibility is another crucial factor to consider when developing a driver guidance system in India. The necessary technology should exist to develop the system. The proposed equipment should have the technical capacity to hold the required data. The system should provide adequate response to inquiries, and it should be upgradable and secure.

- The driver guidance system has a high technical feasibility. The necessary technology already exists to develop the system. The proposed equipment has the technical capacity to hold the required data, and the system provides adequate response to inquiries. The system is upgradable, and there are technical guarantees of accuracy, reliability, ease of access.
- In conclusion, the feasibility study for the driver guidance system in India shows that the system is economically, operationally, and technically feasible. The development cost of the system is minimal, and the benefits derived from the system are substantial. The system will increase road safety, reduce accidents, and provide real-time guidance to drivers. The system is user-friendly, upgradable, and secure, making it an ideal solution for the Indian market.

4.2 FUNCTIONAL REQUIREMENTS:

A driver guidance system is designed to assist drivers in safely and efficiently operating a vehicle. Here are some functional requirements that a driver guidance system should typically meet:

- **Obstacle detection**: Ability to detect and identify obstacles on the road, such as vehicles, pedestrians, animals, speed breakers and potholes.
- **Path prediction**: Ability to predict the path of the vehicle and provide guidance to the driver on the best path to take.
- **User interface**: The system should have a user-friendly interface for interaction and guidance to drivers.

4.3 NON-FUNCTIONAL REQUIREMENTS:

In addition to functional requirements, there are also non-functional requirements that are important for the system. These requirements include:

- **Reliability:** The system must be reliable and provide consistent performance.
- **Accuracy:** The system must accurately detect and identify obstacles on the road.
- **Responsiveness:** The system must respond in real-time to changes in the road environment.
- **Usability:** The system must be easy to use and understand for the driver.
- **Scalability:** The system must be scalable to accommodate future growth and changes in technology.
- **Maintainability:** The system must be easily maintained and updated.

4.4 SOFTWARE SPECIFICATIONS:

Software is a set of programs to do a particular task. Software is an essential requirement of computer system the following are the software requirements for this project:

	In Development	**In Production**
Operating System	Windows 10 or Ubuntu	Raspberry Pi OS
Programming Language	Python	Python
IDE	Google Collaboratory, VSCode	Geany/Thony Python
Libraries and Frameworks	MakeSense.ai, OpenCV, Tensorflow, scipy, numpy.	Numpy, OpenCV, TensorflowLite.

Table 4.4: Software Specifications

SOFTWARE REQUIREMENTS DESCRIPTION:

1. Anaconda

The open-source Anaconda Distribution id the easiest way to perform Python/R data science and machine learning in Linux, Windows, and Mac OS. With over 11 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling individual data scientists to:

- Quickly download 1500+ Python/R data science packages.
- Manage libraries, dependencies, and environments with anaconda.
- Develop and train machine learning and deep learning models with sci-kit learn, Tensorflow
- Analyse the data with scalability and performance with NumPy, Pandas, and matplotlib

2. Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. It's high-level built-in data structures, combined with dynamic typing and dynamic binding; make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

3. Google Colaboratory

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

Google Colab is Jupyter Notebook + Cloud + Google Drive

Colaboratory is a data analysis tool which combines code, output and descriptive text into one document (interactive notebook).

Colab provides GPU and is totally free. By using Google Colab, you can:

- Build your analytics products quickly in a standardized environment.
- Facilitates popular DL libraries on the go such as PyTorch, and TensorFlow
- Share code & results within your Google Drive
- Save copies and create playground modes for knowledge sharing
- Colab is runnable on the cloud or on local server with Jupyter

4. MakeSense.ai

MakeSense is an open-source and free-to-use annotation tool under GPLv3 license. It does not require any advanced installations; it just needs a web browser to run it.

Benefits:

- Open-source
- Free
- Web-based

The user interface is easy to use. You simply upload the images you want to annotate, annotate the images, and export the labels.

MakeSense supports multiple annotations: bounding box, polygon, and point annotation. You can export the labels in different formats including YOLO, VOC XML, VGG JSON, and CSV.

5. Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.Keras is the high-level API of TensorFlow an approchable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity.Keras

empowers engineers and researchers to take full advantage of the scalability and cross-platform capabilities of TensorFlow 2.0: you can run Keras on TPU or large clusters of GPUs, and you can export your Keras models to run in the browser or on a mobile device.

6. Tensorflow

TensorFlow is a popular open-source platform used for building and training machine learning models. It was developed by the Google Brain team and is widely used for developing applications related to image recognition, natural language processing, and deep learning. TensorFlow provides a comprehensive range of tools and APIs for building and training models that can run on CPUs, GPUs, and even on mobile devices. The platform offers a range of pre-trained models, which can be used as starting points for building custom models. TensorFlow also supports distributed computing, making it suitable for developing models that can process large amounts of data.

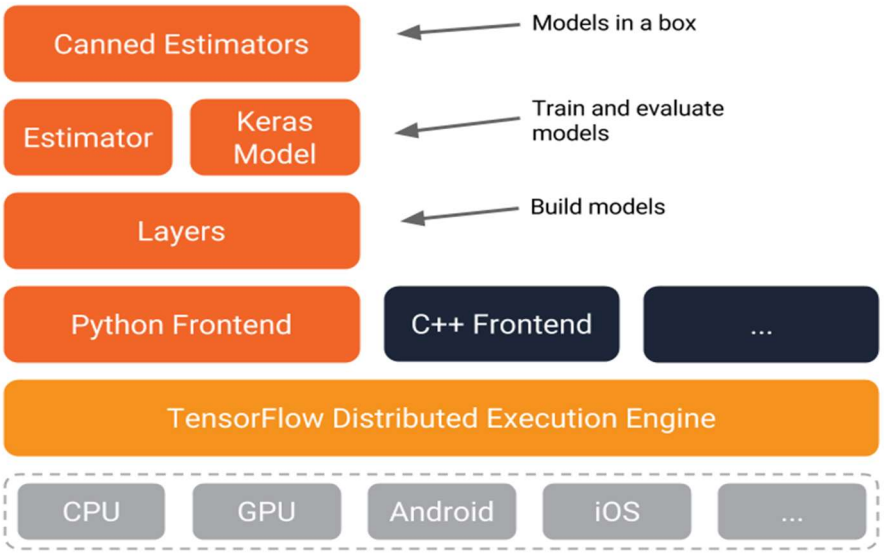

Figure 4.4.6 TensorFlow Architecture

The TensorFlow architecture follows a data flow graph where nodes represent mathematical operations, and edges represent multidimensional data arrays or tensors. This architecture from **Figure 4.4.6** allows for distributed computing across multiple CPUs and GPUs, making it efficient for large-scale machine learning tasks.

TensorFlow uses an API that allows developers to define complex computations as a graph of data flows, making it easy to parallelize tasks and optimize computation. The architecture also includes high-level APIs that simplify model building, data preparation, and evaluation. The flexible architecture allows for easy customization and integration with other tools and libraries.

4.5 HARDWARE SPECIFICATIONS:

Hardware requirements are the necessary hardware required for the project. The following are the hardware requirements of this project:

	In Development	**In Production**
Processor	Intel i3 or above	Raspberry Pi Model 4
RAM	Min 8 GB RAM	8 GB RAM
Storage	500 GB SSD	32 GB SD Card
Camera	**-**	Pi Cam

Table 4.5: Hardware Specifications

Chapter – 5
SYSTEM DESIGN

5. SYSTEM DESIGN

5.1 System Architecture:

System architecture refers to the design and organization of a computer-based system. It involves the high-level structure of hardware and software components, their interactions and relationships, and how they work together to achieve the system's goals. A well-designed system architecture ensures that the system is reliable, scalable, maintainable, and secure. It also enables efficient communication between different components of the system and facilitates system integration and development. The system architecture is a crucial part of the system development life cycle and is typically developed in the early stages of the project.

- The following block diagram shows the architecture of the Interfacing of different aspects used in this project.

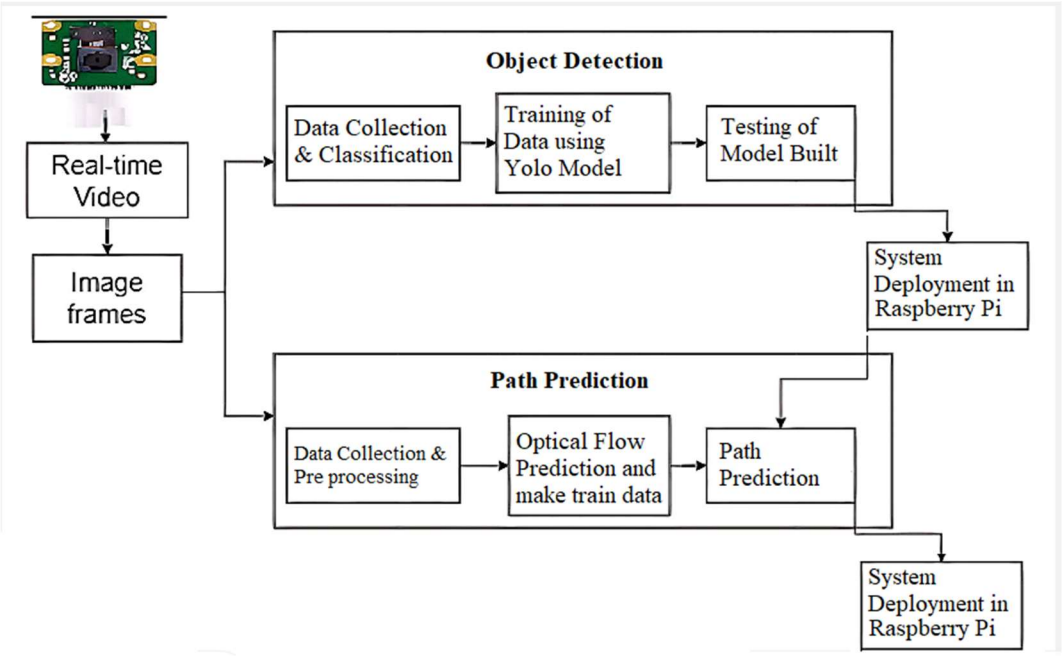

Figure 5.1: System Architecture

- **The architecture of the system mainly consists of:**

1. Object Detection

2. Path Prediction

3. Raspberry Pi

5.2 UML Diagrams:

In the field of software engineering, the Unified Modelling Language (UML) is a standardized visual specification language for object modelling. UML is a general-purpose modelling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model. The model also contains a "Semantic backplane"-documentation such as written use cases that drive the model elements and diagrams.

5.2.1 Importance of UML in Modelling:

A modelling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. A modelling language such as UML is thus a standard language for software blueprints. The UML is not a visual programming language, but its models can be directly connected to various programming languages. This means that it is possible to map from a model in the UML to a programming language Java, C++ or Visual Basic, or even to tables in relational database or the persistent store of an object-oriented database. This mapping permits forward engineering: the generation of code from a UML model into a programming language. The reverse is also possible you can reconstruct a model from an implementation back into UML. This is a programming language that is used for object-oriented software development. To organize program code more efficiently, programmers often create "objects" that are sets of structured data within programs. UML, which has been standardized by the Object Management Group (OMG), was designed for this purpose. The language has gained enough support that it has become a standard language for visualizing and constructing software programs.

5.2.2 A conceptual model of UML:

The three major elements of UML are:

- The UML's basic building blocks
- The rules that dictate how those building blocks may be put together
- Some common mechanism that applies throughout the UML

5.2.3 Use Case Diagram:

The Use Case Diagram illustrates the interactions between the actors and the system. The driver is the primary user of the system who interacts with the user interface to provide input and receive guidance. The sensors provide data to the system for processing, while the algorithm processes the data and provides the necessary guidance to the driver. The use cases describe the specific functionality of the system, such as object detection, path prediction, emergency stop, and sensor calibration. The Use Case Diagram provides a high-level view of the system and its interactions with the actors, helping to identify the key features and requirements of the system.

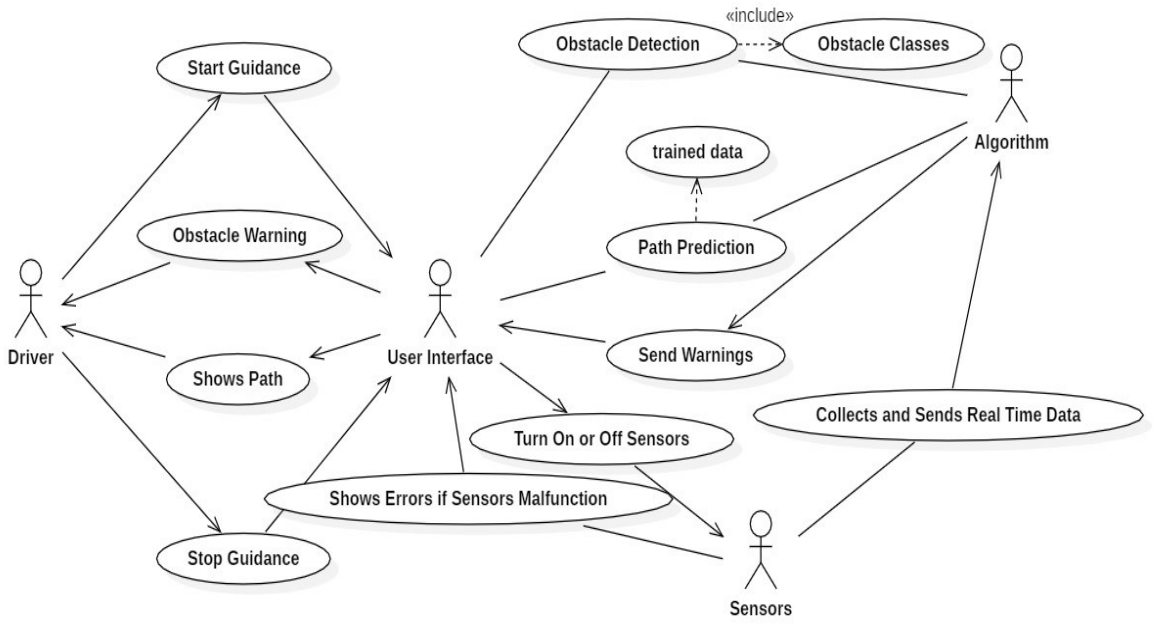

Figure 5.2.3: Use Case Diagram

From the **Figure 5.2.3** above, actors and their use cases can be understood as below:

Actors:

- Driver: The main user of the system who interacts with the user interface.
- User Interface: The interface that provides the user with the necessary information and options to interact with the system.
- Sensors: The input devices that collect data from the environment and provide it to the system for processing.

- Algorithm: The software component that processes the sensor data and provides the necessary guidance to the driver.

Use Cases:

- Object Detection: The system detects the presence of objects in the environment using sensors and alerts the driver through the user interface.
- Path Prediction: The system predicts the path of the vehicle using sensor data and provides the necessary guidance to the driver through the user interface.
- User Input: The driver can provide input to the system through the user interface, such as setting a destination or adjusting the sensitivity of the sensors.
- Emergency Stop: The driver can stop the ruining of model immediately in case of an emergency through the user interface.

5.2.4 Class Diagram:

By using a Class Diagram, we can model the components of the system, their attributes, and their interactions. This helps in the design and implementation of a more structured and organized Driver Guidance System.

IDENTIFYING RELATIONSHIPS:

In UML the ways that things can connect to each other wither logically or physically are modelled as relationships. In object-oriented modelling there are three kinds of relationships that are most important, they are:

1. **Dependencies:** It is a using relationship that states that a change in specification of one thing may affect another thing that uses each. Graphically dependency is rendered as a dashed directed line.
2. **Generalization:** It is relationship between a general thing (called parent) and a more specific kind of that thing (called the child). Generalization is sometimes called ‘is-a-kind-of’ relationship. It means that objects of the child may be used anywhere that parent may appear. Graphically it is rendered as a solid directed line with a large open arrow head pointing to the parent.
3. **Association:** It is a structural relationship that specifies that objects of one thing are connected to the objects of another. The associations that connect more than two classes are called n-array associations. Graphically it is rendered as solid line connecting the same or different classes.

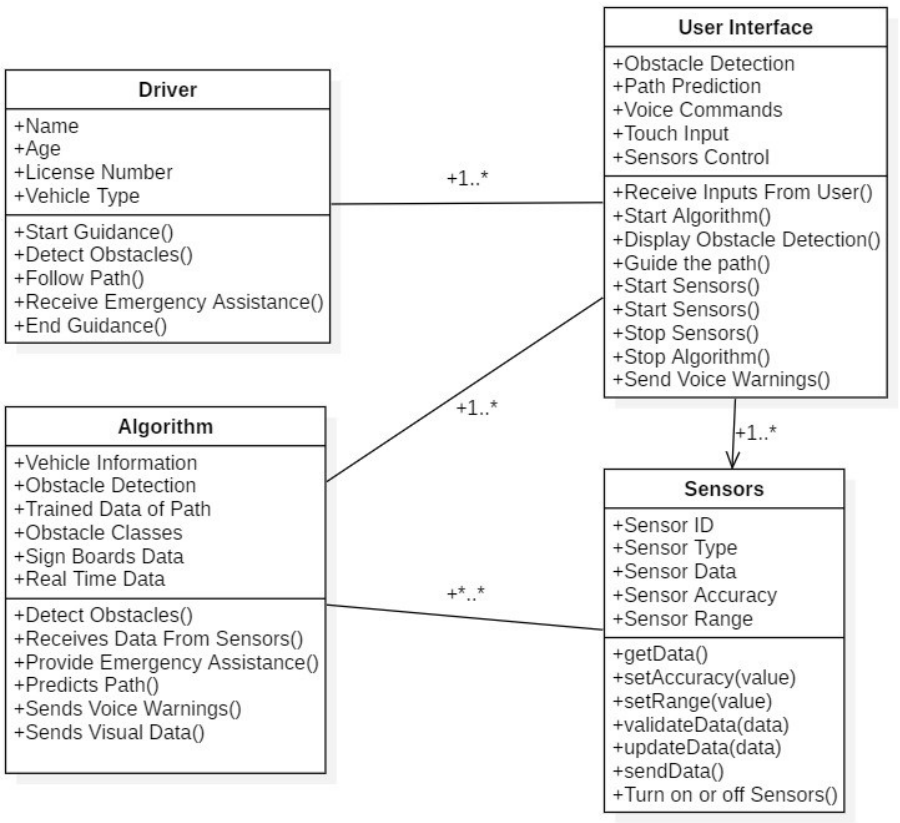

Figure 5.2.4: Class Diagram

From the **Figure 5.2.4**, The class diagram for Driver Guidance System consists of four main classes:

1. **Driver Class:** This class represents the driver and contains the driver's information such as name, license number, and contact details. It also includes methods related to the driver, such as starting the model.
2. **User Interface Class**: This class represents the user interface of the Driver Guidance System and contains methods for displaying the different screens of the interface. It includes methods for displaying the main menu, displaying the camera feed, and displaying the results of the algorithms.
3. **Algorithm Class:** This class represents the algorithms used by the Driver Guidance System for object detection and path prediction. It includes methods for initializing the algorithms, processing sensor data, and outputting results.

5.2.5 Activity Diagram:

The Activity Diagram provides a visual representation of the flow of activities involved in the DGS system, helping to understand the functioning of the system in a clear and concise manner.

Activity diagram commonly contains:

1. **Initial Node:** This node represents the start of the activity diagram and is typically represented by a filled-in circle.
2. **Activity:** An activity is a task or action that is performed in the system or process. It is represented by a rounded rectangle with a label inside.
3. **Decision Node:** A decision node is used to represent a point in the process where the flow branches based on a decision. It is represented by a diamond-shaped symbol.
4. **Merge Node:** A merge node is used to bring together multiple flows back into a single flow. It is represented by a diamond-shaped symbol with multiple incoming flows and a single outgoing flow.
5. **Fork Node:** A fork node is used to represent a point in the process where the flow splits into multiple parallel flows. It is represented by a vertical bar with multiple outgoing flows.
6. **Join Node:** A join node is used to bring together multiple parallel flows back into a single flow. It is represented by a vertical bar with multiple incoming flows and a single outgoing flow.
7. **Final Node:** A final node represents the end of the activity diagram and is typically represented by a filled-in circle with a border.

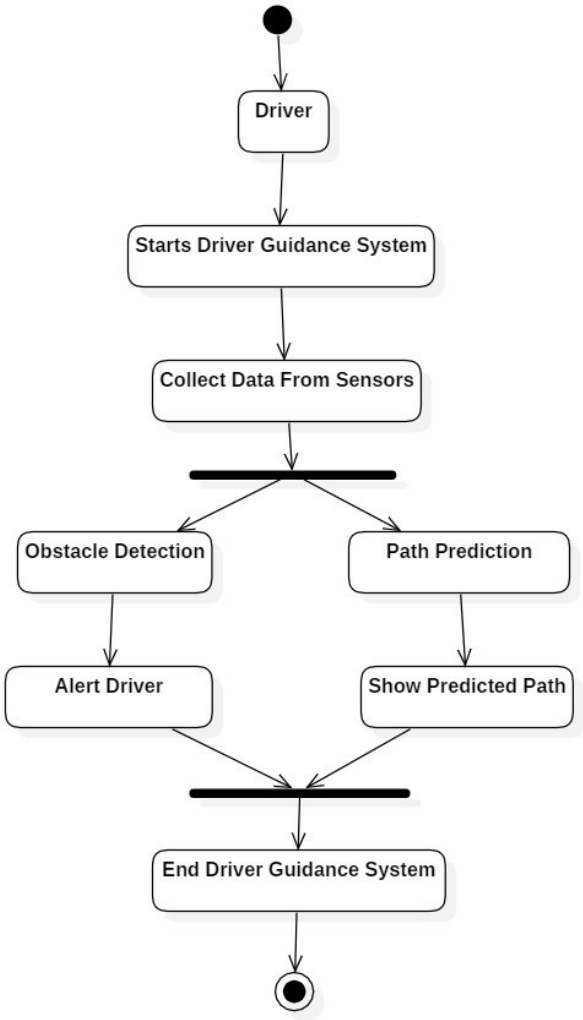

Figure 5.2.5: Activity Diagram

From the **Figure 5.2.5**, The activity diagram for Driver Guidance System consists of following actions:

- The first action is to "Start Driver Guidance System", which involves the selection of Obstacle Detection and Path Prediction models from the user interface.
- These Models perform actions such as “Alert Driver” and “Show Predicted Path” to the user.
- The Activity Diagram Ends with Ending the Driver Guidance System and reaching the end node.

5.2.6 Sequence Diagram:

By using a sequence diagram, we can model the interactions and communications between the components of the system. This helps in the design and implementation of a more organized and structured Driver Guidance System, and also provides an understanding of the flow of events in the system.

Sequence-diagram has two features:

1. **The object life line:** An object life line is the vertical dashed line that represents the existence of an object over a period of time. Most objects that appear in an interaction diagram will be in existence for the duration of the interaction, so these objects are all aligned on the top of the diagram, with their lifelines drawn from the top of the diagram to the bottom.
2. **The focus of control:** It is a tall, thin rectangle that shows the period of time during which an object is performing an action, either directly or a sub ordinate procedure. The top of the rectangle is aligned with the start of the action the bottom is aligned with its completion.

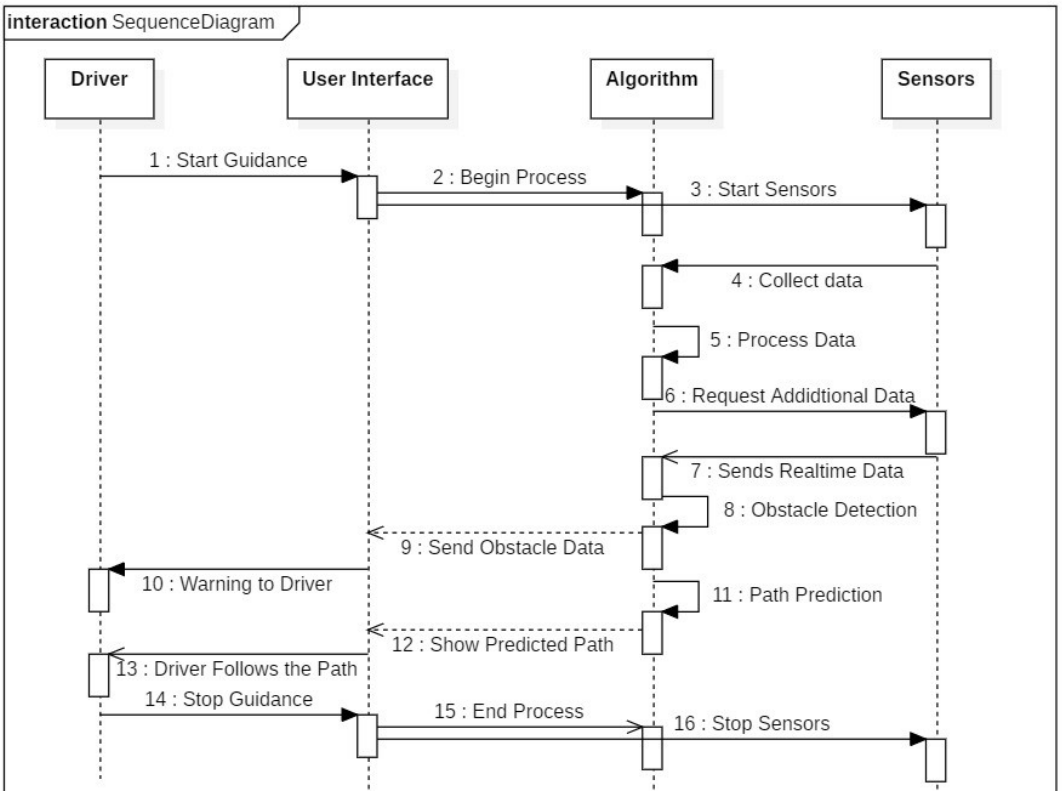

Figure 5.2.6: Sequence Diagram

5.2.7. Deployment Diagram

By using a deployment diagram, we can visualize the physical deployment of the system components and their relationships. It provides a clear understanding of how the components are deployed, how they communicate, and the hardware required for deployment. This information is crucial for planning, implementing, and maintaining the Driver Guidance System.

Some of the components of Deployment Diagram are:

1. **Nodes:** These are the physical hardware components such as servers, printers, and routers.
2. **Artifacts:** These are the software components such as JAR files, WAR files, and DLLs.
3. **Communication Paths:** These are the channels of communication between the nodes, such as network cables, Wi-Fi, and Bluetooth connections.
4. **Associations:** These are the links between the nodes and artifacts, which represent the deployment relationships.
5. **Dependencies:** These are the relationships between the artifacts that represent the required libraries or dependencies of the system.

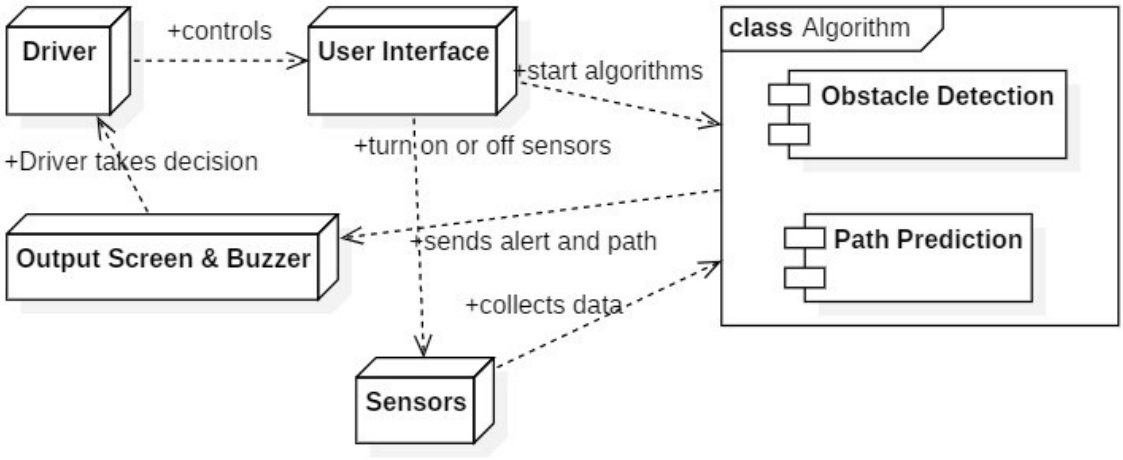

Figure 5.2.7: Deployment Diagram

Chapter – 6
IMPLEMENTATION

6. IMPLEMENTATION

6.1 Datasets

A dataset is a collection of data. Most commonly a dataset corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the dataset in question. The data set lists values for each of the variables such as the height or weight of an object for each member in the dataset. A data set is organized into some type of data structure. In a database, for example, a data set might contain a collection of business data (names, salaries, contact information, sales figures, and so forth). The database itself can be considered a data set, as can bodies of data within it related to a particular type of information, such as sales data for a particular corporate department.

DATASET DESCRIPTION:

Dataset is taken from Kaggle user **Anand P V** with title **Self Driving Car on Indian Roads.** This data set provides easy to use training data for self-driving vehicles. Steering Angle corresponding to each frame in driving video is provided. The video is recorded using a camera fixed on the windshield of a car, driven along the roads of Kerala, India.

16.5 minutes = 16.5{min} x 60{1 min = 60 sec} x 30{fps} =29,700 images~2 GB

The self-recorded driving videos required some pre-processing steps before it can be fed to the network. Following softwares were used, in sequence to generate the video frames:

- Lossless Cut-To cut out relevant video without loss.
- Anytime Video Converter-To resize video to reduce data.
- Video to JPG Converter-To extract frames from video.

The dataset also includes other relevant information, such as the speed of the car and the position of other objects on the road. This additional data can be used to improve

the performance of the self-driving car system and make it more robust in different scenarios.

6.2 Work Flow Design:

The workflow design of a Driver Guidance System (DGS) involves multiple components and stages. The system is designed to guide drivers and provide warnings about upcoming obstacles on Indian roads while offering instructions on how to handle them.

The workflow design of DGS includes the following components:

- Guidance of Path: The system uses a path prediction model for Indian roads to offer the best possible route to the driver while taking into account the unique characteristics of traffic and infrastructure such as heavy congestion and potholes.
- Obstacle Detection: The system uses sensors to detect obstacles like potholes, construction zones, and stray animals on Indian roads.
- User Interface: The system is designed to provide a user-friendly interface for drivers to easily navigate and use the system.
- Alert System: The system has an alert system that warns drivers about upcoming obstacles and provides instructions on how to handle them.

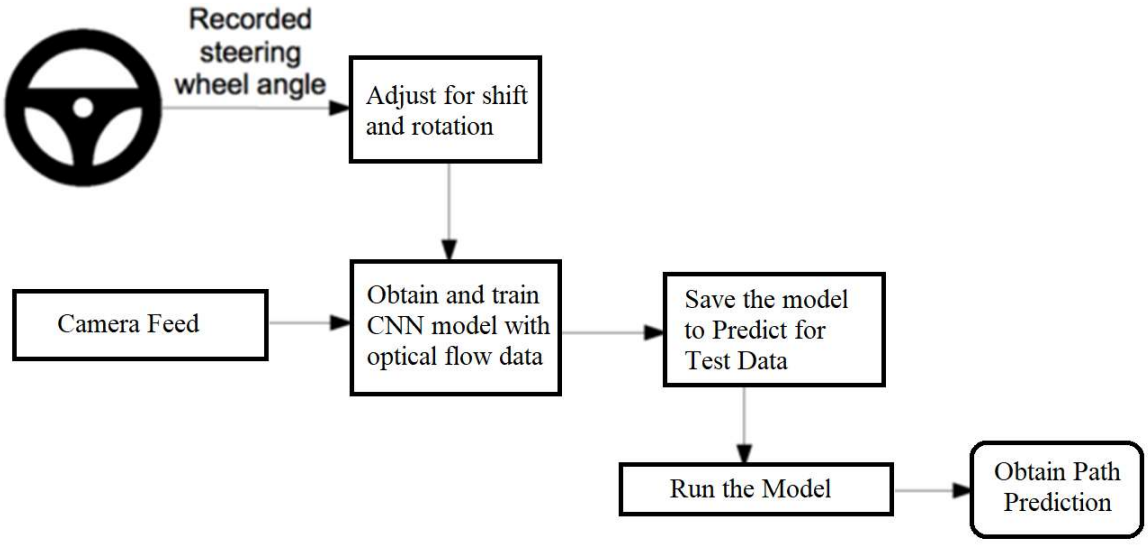

Figure 6.2.1: Work Flow Design of Path Prediction

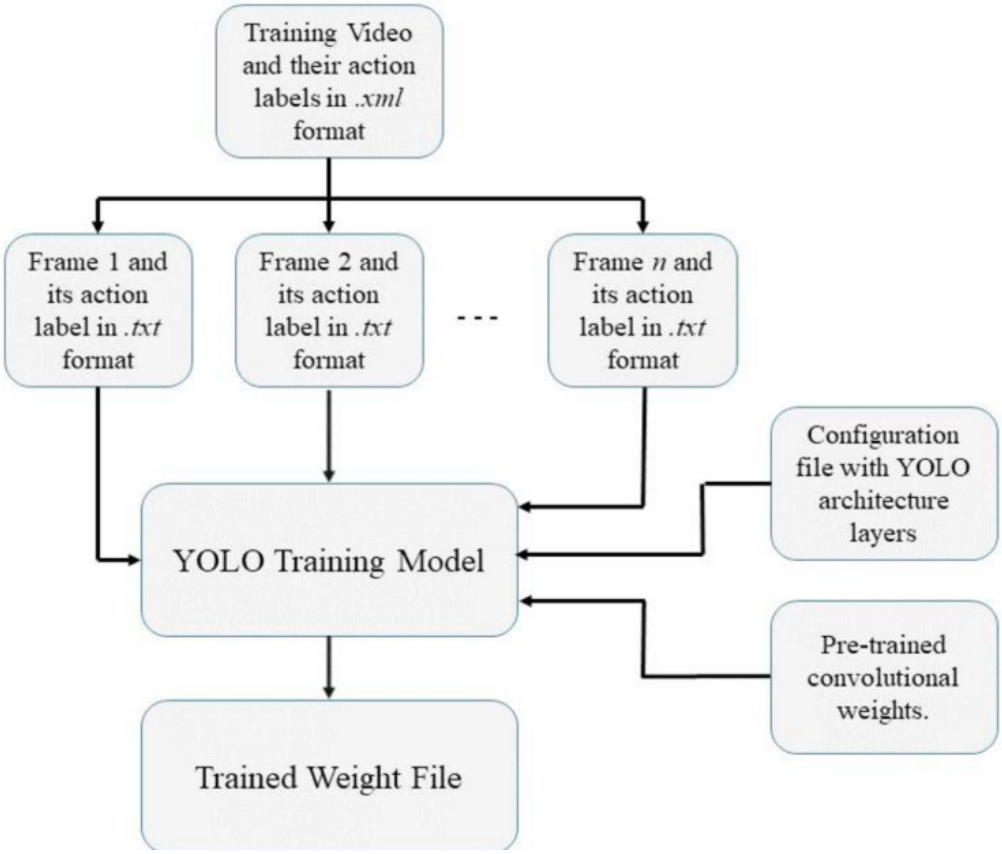

Figure 6.2.2: Work Flow Design of Object Detection

6.3 Source Code:

6.3.1 Generate Optical Flow Train Data: -

Below python program analyze recorded video & compute the optical flow parameter, 'gamma' across each adjacent frame in video and output to file.

```
import numpy as np
import cv2
import scipy.misc
import math
from scipy import signal
#read data.txt
xs = []
ys = []
accels = []
with open("indian_dataset/data.txt") as f:
    for line in f:
        xs.append("indian_dataset/circuit2_x264.mp4 " + str(line.split()[0]).zfill(5) +
".jpg")
        ys.append(float(line.split()[1]))
feature_params = dict(maxCorners = 100,
```

```
                qualityLevel = 0.3,
                minDistance = 7,
                blockSize = 7)
lk_params = dict(winSize = (15,15),
            maxLevel = 2,
            criteria = (cv2.TERM_CRITERIA_EPS |
cv2.TERM_CRITERIA_COUNT, 10, 0.03))
out_imgpath = 'indian_dataset/optFlow.txt'
optF = open(out_imgpath, "w")
# Function to calculate distance
def distance(x1 , y1 , x2 , y2):
   # Calculating distance
   return math.sqrt(math.pow(x2 - x1, 2) +
           math.pow(y2 - y1, 2) * 1.0)
# Take first frame and find corners in it
frame = scipy.misc.imread(xs[0], mode="RGB")
previous_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(previous_frame, mask = None, **feature_params)
frameNum = 0
while(frameNum < len(xs)):
   frame = scipy.misc.imread(xs[frameNum], mode="RGB")
   present_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   mask = np.zeros_like(frame)
   p0 = cv2.goodFeaturesToTrack(present_frame, mask = None,
**feature_params)
   # calculate optical flow
   p1, st, err = cv2.calcOpticalFlowPyrLK(previous_frame, present_frame, p0,
None, **lk_params)
   # Select good points
   good_new = p1[st==1]
   good_old = p0[st==1]
   distSum = 0
   # draw the tracks
   for i,(new,old) in enumerate(zip(good_new,good_old)):
      a,b = new.ravel()
      c,d = old.ravel()
      dist = distance (a, b, c, d)
      distSum += dist
   # When number of detected corners (good features) are less than 10 then
distance becomes erroneous.
   if (good_old.shape[0] < 10):
      opticalParam = prev_opticalParam
   else:
```

```
        opticalParam = distSum/(math.pow(good_old.shape[0],
2)+math.pow(ys[frameNum], 2)+1)*100
    optF.write(str(opticalParam) + '\n')
    prev_opticalParam = opticalParam
    # Now update the previous frame and previous points
    previous_frame = present_frame.copy()
    frameNum += 1
cv2.destroyAllWindows()
optF.close()
```

6.3.2: CNN model creation and save it as model.py file: -

The below code defines a neural network with several convolutional layers and fully connected layers. The network is designed for image classification and takes as input images of size 66x200 with three color channels (RGB).

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import scipy
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)
def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
def conv2d(x, W, stride):
  return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='VALID')
x = tf.placeholder(tf.float32, shape=[None, 66, 200, 3])
y_ = tf.placeholder(tf.float32, shape=[None, 1])
x_image = x
#first convolutional layer
W_conv1 = weight_variable([5, 5, 3, 24])
b_conv1 = bias_variable([24])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1, 2) + b_conv1)
#second convolutional layer
W_conv2 = weight_variable([5, 5, 24, 36])
b_conv2 = bias_variable([36])
h_conv2 = tf.nn.relu(conv2d(h_conv1, W_conv2, 2) + b_conv2)
#third convolutional layer
W_conv3 = weight_variable([5, 5, 36, 48])
b_conv3 = bias_variable([48])
h_conv3 = tf.nn.relu(conv2d(h_conv2, W_conv3, 2) + b_conv3)
#fourth convolutional layer
W_conv4 = weight_variable([3, 3, 48, 64])
```

```
b_conv4 = bias_variable([64])
h_conv4 = tf.nn.relu(conv2d(h_conv3, W_conv4, 1) + b_conv4)
#fifth convolutional layer
W_conv5 = weight_variable([3, 3, 64, 64])
b_conv5 = bias_variable([64])
h_conv5 = tf.nn.relu(conv2d(h_conv4, W_conv5, 1) + b_conv5)
#FCL 1
W_fc1 = weight_variable([1152, 1164])
b_fc1 = bias_variable([1164])
h_conv5_flat = tf.reshape(h_conv5, [-1, 1152])
h_fc1 = tf.nn.relu(tf.matmul(h_conv5_flat, W_fc1) + b_fc1)
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
#FCL 2
W_fc2 = weight_variable([1164, 100])
b_fc2 = bias_variable([100])
h_fc2 = tf.nn.relu(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)
h_fc2_drop = tf.nn.dropout(h_fc2, keep_prob)
#FCL 3
W_fc3 = weight_variable([100, 50])
b_fc3 = bias_variable([50])
h_fc3 = tf.nn.relu(tf.matmul(h_fc2_drop, W_fc3) + b_fc3)
h_fc3_drop = tf.nn.dropout(h_fc3, keep_prob)
#FCL 3
W_fc4 = weight_variable([50, 10])
b_fc4 = bias_variable([10])
h_fc4 = tf.nn.relu(tf.matmul(h_fc3_drop, W_fc4) + b_fc4)
h_fc4_drop = tf.nn.dropout(h_fc4, keep_prob)
#Output
W_fc5 = weight_variable([10, 1])
b_fc5 = bias_variable([1])
y = tf.multiply(tf.atan(tf.matmul(h_fc4_drop, W_fc5) + b_fc5), 2)
#scale the atan output
```

6.3.3 Training of Steering Prediction Model: -

The below code trains a model to predict a driving path from input images with 30 epochs and a batch size of 100. The training loop iterates over the dataset 30 times (epochs), and at each epoch, it loads batches of 100 images and their corresponding steering angles to train the model.

```
import os
import tensorflow.compat.v1 as tf
from tensorflow.core.protobuf import saver_pb2
```

```
import driving_data
import model
LOGDIR = './save'
sess = tf.InteractiveSession()
L2NormConst = 0.001
train_vars = tf.trainable_variables()
loss = tf.reduce_mean(tf.square(tf.subtract(model.y_, model.y))) +
tf.add_n([tf.nn.l2_loss(v) for v in train_vars]) * L2NormConst
train_step = tf.train.AdamOptimizer(1e-4).minimize(loss)
sess.run(tf.initialize_all_variables())
# create a summary to monitor cost tensor
tf.summary.scalar("loss", loss)
# merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()
saver = tf.train.Saver(write_version = saver_pb2.SaverDef.V1)
# op to write logs to Tensorboard
logs_path = './logs'
summary_writer = tf.summary.FileWriter(logs_path,
graph=tf.get_default_graph())
epochs = 30
batch_size = 100
# train over the dataset about 30 times
for epoch in range(epochs):
  for i in range(int(driving_data.num_images/batch_size)):
    xs, ys = driving_data.LoadTrainBatch(batch_size)
    train_step.run(feed_dict={model.x: xs, model.y_: ys, model.keep_prob:
0.8})
    if i % 10 == 0:
      xs, ys = driving_data.LoadValBatch(batch_size)
      loss_value = loss.eval(feed_dict={model.x:xs, model.y_: ys,
model.keep_prob: 1.0})
      print("Epoch: %d, Step: %d, Loss: %g" % (epoch, epoch * batch_size +
i, loss_value))
    summary = merged_summary_op.eval(feed_dict={model.x:xs, model.y_:
ys, model.keep_prob: 1.0})
    summary_writer.add_summary(summary, epoch *
driving_data.num_images/batch_size + i)
    if i % batch_size == 0:
      if not os.path.exists(LOGDIR):
        os.makedirs(LOGDIR)
      checkpoint_path = os.path.join(LOGDIR, "model.ckpt")
      filename = saver.save(sess, checkpoint_path)
  print("Model saved in file: %s" % filename)
print("Run the command line:\n" \
```

```
    "--> tensorboard --logdir=./logs " \
    "\nThen open http://0.0.0.0:6006/ into your web browser")
```

6.3.4 Running Trained Model for Path Prediction: -

The below code uses trained model to predict the steering angle of a car from a video input and process it to predict steering path.

```
sess = tf.InteractiveSession()
saver = tf.train.Saver()
saver.restore(sess, "save/model.ckpt")
img = cv2.imread('steering_wheel_image.jpg',0)
rows,cols = img.shape
smoothed_angle = 0
# Video capture
cap = cv2.VideoCapture('test.mp4')
while(cv2.waitKey(10) != ord('q')):
    ret, frame = cap.read()
    image = cv2.resize(frame, (200, 66)) / 255.0
    degrees = model.y.eval(feed_dict={model.x: [image], model.keep_prob:
1.0})[0][0] * 180 / 3.14159265
    if not windows:
        call("clear")
    print("Predicted steering angle: " + str(degrees) + " degrees")
    frame = cv2.resize(frame, (1280, 720))
    # Run object detection on the preprocessed frame
    frame = modelobject(frame)
    # Compute the steering advice based on the predicted angle
    if degrees > 10:
        dir.append('R')
    elif degrees < -10:
        dir.append('L')
    else:
        dir.append('F')
    if len(dir) > 10:
        dir.pop(0)
    W = 400
    H = 275
    widget = np.copy(frame[:H, :W])
    widget //= 2
    widget[0,:] = [0, 0, 255]
    widget[-1,:] = [0, 0, 255]
    widget[:,0] = [0, 0, 255]
    widget[:,-1] = [0, 0, 255]
    frame[:H, :W] = widget
    direction = max(set(dir), key = dir.count)
```

```
    if direction == 'L':
        y, x = left_curve_img[:,:,2].nonzero()
        frame[y, x-100+W//2] = left_curve_img[y, x, :3]
        msg = "Left Curve Ahead"
    if direction == 'R':
        y, x = right_curve_img[:,:,2].nonzero()
        frame[y, x-100+W//2] = right_curve_img[y, x, :3]
        msg = "Right Curve Ahead"
    if direction == 'F':
        y, x = keep_straight_img[:,:,2].nonzero()
        frame[y, x-100+W//2] = keep_straight_img[y, x, :3]
        msg = "Keep Straight Ahead"
    cv2.putText(frame, msg, org=(10, 240),
fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, color=(255, 255,
255), thickness=2)
     # Calculate and print the FPS on the frame
    frame_count += 1
    elapsed_time = time.time() - start_time
    fps = frame_count / elapsed_time
    cv2.putText(frame, "FPS: {:.2f}".format(fps), (10, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
    cv2.imshow("frame", frame)
    #make smooth angle transitions by turning the steering wheel based on the
difference of the current angle
    #and the predicted angle
    smoothed_angle += 0.2 * pow(abs((degrees - smoothed_angle)), 2.0 / 3.0)
* (degrees - smoothed_angle) / abs(degrees - smoothed_angle)
    M = cv2.getRotationMatrix2D((cols/2,rows/2),-smoothed_angle,1)
    dst = cv2.warpAffine(img,M,(cols,rows))
    cv2.imshow("steering wheel", dst)
```

6.3.5 Inferencing Object Detection Model with TensorFlow: -

The below code uses a custom TensorFlow Lite model for object detection, which takes in an image frame and returns a frame with detected objects and labels drawn on it.

```
interpreter = tf.lite.Interpreter(model_path="detect1.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
min_confidence = 0.60
def modelobj(frame):
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

```
    frame_resized = cv2.resize(frame_rgb, (320, 320))
    input_data = np.expand_dims(frame_resized, axis=0)
    # Normalize pixel values if using a floating model (i.e. if model is non-
quantized)
    input_data = (np.float32(input_data) - 127.5) / 127.5
    # Perform the actual detection by running the model with the image as
input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()
    # Retrieve detection results
    boxes = interpreter.get_tensor(output_details[1]['index'])[0] # Bounding
box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[3]['index'])[0] # Class index
of detected objects
    scores = interpreter.get_tensor(output_details[0]['index'])[0] # Confidence
of detected objects
    # Loop over all detections and draw detection box if confidence is above
minimum threshold
    for i in range(len(scores)):
        if ((scores[i] > min_confidence) and (scores[i] <= 1.0)):
            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image
dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * 720)))
            xmin = int(max(1,(boxes[i][1] * 1280)))
            ymax = int(min(720,(boxes[i][2] * 720)))
            xmax = int(min(1280,(boxes[i][3] * 1280)))
            cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 4)
            # Draw label
            object_name = labelsobj[int(classes[i])] # Look up object name from
"labels" array using class index
            label = object_name+': {:.2f}%'.format(scores[i]*100)  # Example:
'person: 72%'
            labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
            label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw
label too close to top of window
            cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-10),
(xmin+labelSize[0], label_ymin+baseLine-10), (255, 255, 255),
cv2.FILLED) # Draw white box to put label text in
            cv2.putText(frame, label, (xmin, label_ymin-7),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text
    #return it detected frame
    return(frame)
```

6.3.6 Tkinter User Interface for controlling the model: -

```
def toggle_checkbuttons(var, checkbuttons):
    state = var.get()
    for cb in checkbuttons:
        if cb is not var:
            cb.set(state)
# function to run selected scripts
def run_scripts():
    if obj_var.get() == 1 and path_var.get() == 1:
        run()
    elif obj_var.get() == 1:
        obj()
    elif path_var.get() == 1:
        path()
    else:
        print("No scripts selected")

# create tkinter window
root = tk.Tk()
root.geometry("720x1080")
root.title("Driver Guidance System")

items = ["obj_var", "path_var"]
listbox = tk.Listbox(root)
for item in items:
    listbox.insert(tk.END, item)
# create a BooleanVar to track the toggle button state
select_all_var = tk.BooleanVar()
# create labels for checkboxes
obj_label = tk.Label(root, text="Object Detection")
path_label = tk.Label(root, text="Path Prediction")
both_label = tk.Label(root, text="Both")

# create checkboxes
obj_var = tk.IntVar()
obj_check = tk.Checkbutton(root, text="", variable=obj_var)
path_var = tk.IntVar()
path_check = tk.Checkbutton(root, text="", variable=path_var)
both_var = tk.IntVar()
both_check = tk.Checkbutton(root, text="", variable=both_var,
                            command=lambda: toggle_checkbuttons(both_var, [obj_var,
path_var]))
```

```
# add images next to checkboxes
obj_image = tk.PhotoImage(file="object_detection.png")
obj_label2 = tk.Label(image=obj_image)
obj_label2.image = obj_image

path_image = tk.PhotoImage(file="path_prediction.png")
path_label2 = tk.Label(image=path_image)
path_label2.image = path_image

both_image = tk.PhotoImage(file="both.png")
both_label2 = tk.Label(image=both_image)
both_label2.image = both_image



# create button to run selected scripts
run_button = tk.Button(root, text="Run", command=run_scripts)

# place the button at the center of the window
run_button.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

# place all elements in window
obj_check.grid(row=0, column=0, padx=10, pady=10)
obj_label.grid(row=0, column=1, padx=10, pady=10)
obj_label2.grid(row=0, column=2, padx=10, pady=10)
path_check.grid(row=1, column=0, padx=10, pady=10)
path_label.grid(row=1, column=1, padx=10, pady=10)
path_label2.grid(row=1, column=2, padx=10, pady=10)
both_check.grid(row=2, column=0, padx=10, pady=10)
both_label.grid(row=2, column=1, padx=10, pady=10)
both_label2.grid(row=2, column=2, padx=10, pady=10)
run_button.grid(row=3, column=2, padx=10, pady=10)

root.mainloop()
```

6.4 MODEL DEPLOYMENT AND TESTING:

Model deployment on Raspberry Pi involves the process of running and executing a trained machine learning model on a Raspberry Pi device. The deployment process varies depending on the type of model, the hardware of the Raspberry Pi.

The deployment process involves the following steps:

- Prepare the Raspberry Pi: This involves installing the required dependencies, libraries, and packages that are necessary for running the model. The Raspberry Pi should also be configured with the necessary hardware components such as a camera or sensors.
- Convert the model: The trained machine learning model needs to be converted to a format that is compatible with the Raspberry Pi. This involves converting the Yolo model to a lightweight TensorFlow Lite.
- Test the model: Before deployment, the model should be tested on the Raspberry Pi to ensure that it works correctly and is optimized for the hardware.
- Deploy the model: Once the model is tested, it can be deployed on the Raspberry Pi. This involves integrating the model with the application, whether it is a standalone application or a web-based application.
- Monitor and optimize: After deployment, the model should be monitored to ensure that it is performing optimally. This may involve fine-tuning the model or optimizing the hardware configuration of the Raspberry Pi.

Overall, Both the Path Prediction and Object Detection models are deployed in the Raspberry Pi successfully for real time testing and achieved a frame-rates as below.

Only Path Prediction Model:

The path prediction model is a computationally intensive task and requires a high-end machine to run efficiently. The Raspberry Pi, being a low-powered device, may not be able to match the performance of a laptop or desktop computer.

During my testing, I observed that the path prediction model was able to achieve a frame rate of 6 fps on the Raspberry Pi. This is significantly lower than the frame rate achieved on a laptop, which was around 24 fps.

Overall, the performance of the path prediction model on the Raspberry Pi may not match that of a high-end computer, but with the right optimization techniques, it can still be deployed effectively in applications where real-time performance is not critical.

Only Object Detection Model:

When running an object detection model on a Raspberry Pi, the performance can be significantly slower compared to running the same model on a more powerful computer which gives around 8~12 fps. This is due to the limited processing power of the Raspberry Pi, which can result in a lower frames-per-second (fps) rate.

Running Both the Models Combined:

Running an inferred model on the Raspberry Pi resulted in a frame rate of 2 frames per second (fps). This is significantly lower than the frame rate of the same model running on a laptop, which achieved 12 fps. The low fps on the Raspberry Pi is likely due to the limited processing power and memory of the device compared to a laptop.

To improve the performance of the model on the Raspberry Pi, several strategies can be implemented, including:

- Hardware acceleration: The Raspberry Pi can be connected to a hardware accelerator, such as the Intel Movidius Neural Compute Stick, to offload some of the computation and improve the performance of the model.
- Parallel processing: The Raspberry Pi can be configured to use multiple cores or threads to run the model in parallel, which can improve its performance.
- Lowering image resolution: The resolution of the input images can be reduced to improve the performance of the model on the Raspberry Pi. However, this may come at the cost of reduced accuracy.

By implementing these strategies, the performance of the model on the Raspberry Pi can be improved, allowing it to achieve higher fps and better real-time performance.

Chapter – 7
OUTPUT SCREENS

7. OUTPUT SCREENS

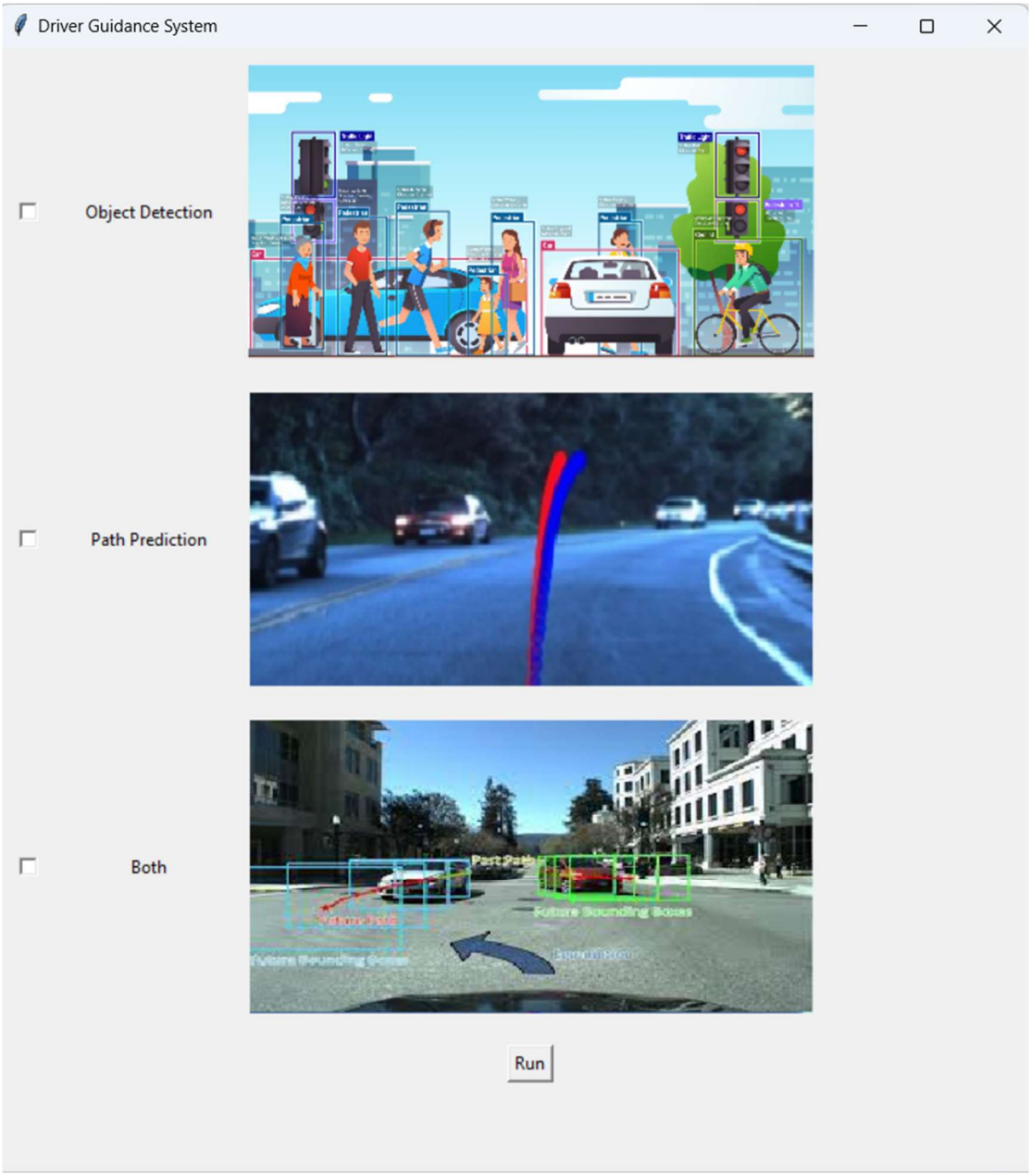

Figure 7.1: User Interface with tkinter

The above **Figure 7.1** shows GUI consists of a window with three checkboxes, each corresponding to a different script (object detection, path prediction, or both), and a "Run" button to execute the selected scripts.

When the "Both" checkbox is clicked, and it toggles the state of both the "Object Detection" and "Path Prediction" checkboxes.

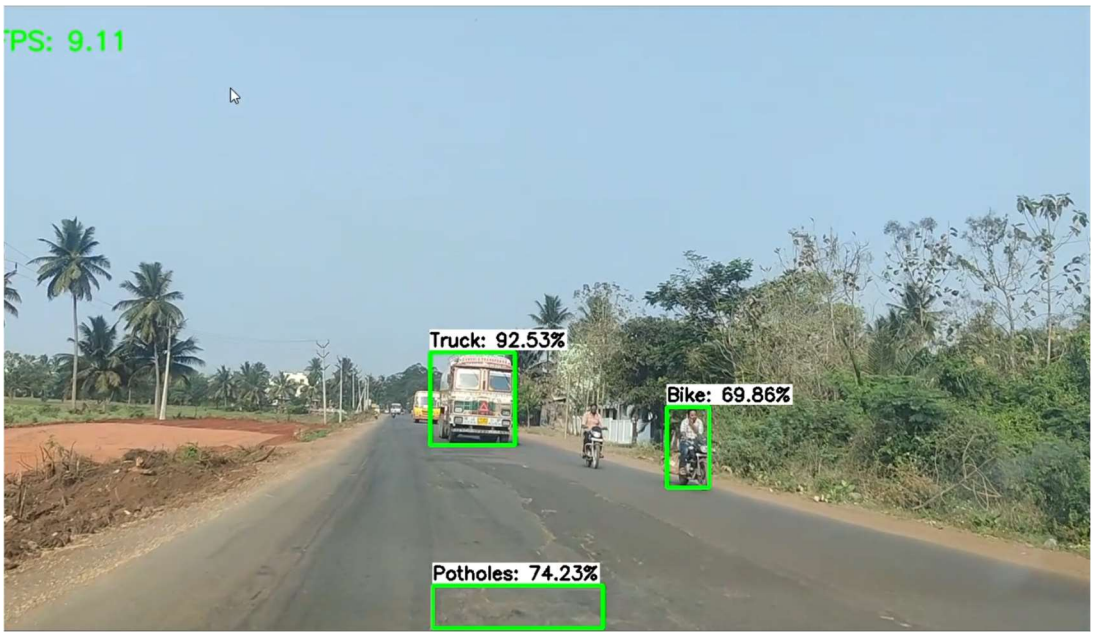

Figure 7.2: Output of only Object Detection Model

The above **Figure 7.2** shows successful detection of obstacles in the given frame. The resulting output screen will show the video frame with bounding boxes drawn around the detected objects and labels showing the object name and confidence score.

Figure 7.3: Output of only Path Prediction Model

The above **Figure 7.3** shows successful prediction of path to take right along with steering prediction. It works well with new tracks, unseen while training the network.

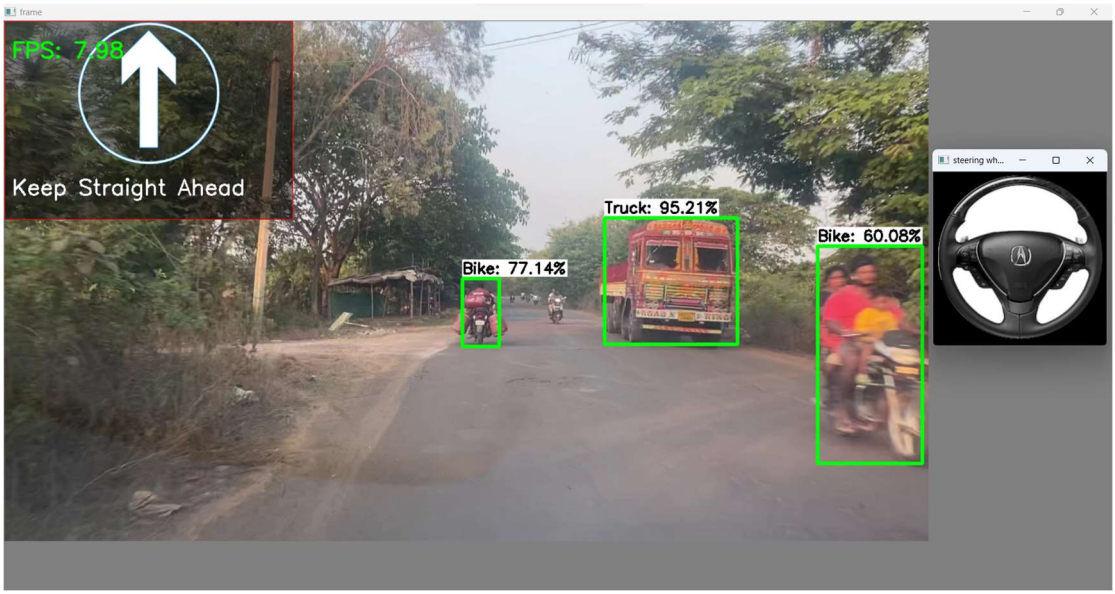

Figure 7.4: Output of both models together

The above **Figure 7.4** shows successful Prediction of both object detection and path prediction models with an average fps of 8.

Figure 7.5: Output of both models together in RPi

The above **Figure 7.5** shows successful Prediction of both object detection and path prediction models on Raspberry Pi with an average fps of 2.

Chapter – 8

CONCLUSION & FUTURE SCOPE

8. CONCLUSION AND FUTURE SCOPE

8.1 Conclusion:

In this project, I have developed a Driver Guidance System which includes both Object Detection and Path Prediction models. I used the TensorFlow Lite Object Detection model which is obtained using trained Yolo Weights and OpenCV to process the frames as required for the model.

- The Object Detection model detects the objects in the video stream captured by the camera mounted on the dashboard of the car. It identifies objects like pedestrians, vehicles, potholes, speed breakers and provides a bounding box around them. The Path Prediction model predicts the steering angles in which the vehicle needs to follow the path.
- I used the Tkinter library to create a GUI for the system interface that displays the settings for configuration of Guidance System to run both the model or run only either of the two models.
- I have tested the system on both a laptop and a Raspberry Pi. We observed a significant decrease in performance on the Raspberry Pi compared to the laptop. The Object Detection model achieved a frame rate of 3 fps on the Raspberry Pi, and the Path Prediction model achieved a frame rate of 6 fps and both the models combined gives a frame rate of 2 fps.
- In conclusion, the Driver Guidance System we developed can provide valuable assistance to drivers and help them avoid accidents. However, further optimization is required to make the system more efficient, especially when deploying it on low-power devices like the Raspberry Pi.

8.2 Future Scope:

Several improvements can be made to the system to enhance its performance and functionality. Some of them are:

- Improve the real-time performance of the system by using a more powerful processor or implementing the system on cloud infrastructure.
- Expand the system's functionality to include lane detection and tracking to help drivers stay in their lane and avoid collisions with other vehicles.
- Implement the system in autonomous vehicles to improve their safety and reliability.
- Integrate the system with other driver assistance systems such as adaptive cruise control and automatic emergency braking to provide a comprehensive safety system.
- Improve the accuracy of the object detection and path prediction models by training them on larger datasets and fine-tuning their parameters.
- Implement a more user-friendly interface to allow users to easily interact with the system and receive alerts in real-time.
- Test the system on different driving conditions and scenarios to evaluate its performance and reliability.

Chapter – 9

REFERENCES

9. REFERENCES

References from Books and online Papers:

[1] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[2] Singh, G., Kumar, R., & Kashtriya, P. (2018). Detection of potholes and speed breaker on road. 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), 490-495.

[3] Li, M., & Zhou, H. (2021). Real-time object detection method based on improved YOLOv4-tiny. IOP Conference Series: Earth and Environmental Science, 744(1), 012135.

Other relevant articles:

Self-Driving Car on Indian Roads Article:

https://towardsdatascience.com/self-driving-car-on-indian-roads-4e305cb04198

Yolov4 implementation in Raspberry Pi:

https://pyimagesearch.com/2020/01/27/yolo-and-tiny-yolo-object-detection-on-the-raspberry-pi-and-movidius-ncs/

A Raspberry Pi 4 64-OS image with deep learning dependencies installed:

https://github.com/Qengineering/RPi-image