Visvesvaraya Technological University
Belgaum, Karnataka-590 014

A Project Phase - 2 Report on

"Enhancing Transportation Safety with YOLO-Based CNN Autonomous Vehicles"

submitted in partial fulfillment of the requirement for

the award of the degree of

Bachelor of Engineering in
Computer Science & Engineering

Submitted by

MOHAMMED SABIR	1HK20CS091
MOHAMMED SUHAIL	1HK20CS094
MOHAMMED UMARULLA	1HK20CS097
MOHAMMED YOUSUF	1HK20CS099

Under the Guidance of

Prof. Sarumathi S

Assistant Professor

Department of Computer Science and Engineering

HKBK College of Engineering

No.22/1, Opp., Manyata Tech Park Rd, Nagavara, Bengaluru, Karnataka 560045.

Approved by AICTE & Affiliated by VTU

Department of Computer Science & Engineering

2023-24

HKBK COLLEGE *of* ENGINEERING

No.22/1, Opp., Manyata Tech Park Rd, Nagavara, Bengaluru, Karnataka -45

(Approved by AICTE & Affiliated to VTU)

Department of Computer Science and Engineering

Certificate

Certified that the Project Work Phase-2 **" Enhancing Transportation Safety with YOLO- Based CNN Autonomous Vehicles"** carried out by are **Mohammed Sabir (1HK20CS091), Mohammed Suhail (1HK20CS094), Mohammed Umarulla (1HK20CS097) and Mohammed Yousuf (1HK20CS099)** Bonafide Students of **HKBK COLLEGE *of* ENGINEERING**, in partial fulfilment of eighth semester project, regards to the subject **"Major Project Phase – 2 (18CSP83)"** for the award of Bachelor of Engineering in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023–2024. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library.

The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Bachelor of Engineering.

Prof. Sarumathi.S	**Dr. Smitha Kurian**	**Dr. Mohammed Riyaz Ahmed**
Internal Guide	HOD-CSE	Principal, HKBKCE

External Viva:

Name of Examiner	Signature with Date
1)_____	_____
2)_____	_____

ACKNOWLEDGEMENT

First of all, we would take this opportunity to express our heartfelt gratitude to the Management of HKBK College of Engineering, **Mr. C.M. Ibrahim**, HKBKGI and Director **Mr. C.M. Faiz**, HKBKGI for providing a healthy environment for the successful completion of project work.

We would like to express thanks to our Principal, **Dr. Mohammed Riyaz Ahmed** for his encouragement that motivated us for the successful completion of Project Work.

We wish to express our gratitude to **Dr. Smitha Kurian,** Professor and Head of the Department of Computer Science & Engineering for providing healthy environment for the successful completion of the Project work.

We express our heartfelt appreciation and gratitude to our Project Guide, **Prof. Sarumathi.S**, Associate Professor of Computer Science and Engineering, HKBK College of Engineering, Bangalore, for her intellectually-motivating support, valuable guidance, Suggestions, and invaluable encouragement during our Project work. Her comprehensive knowledge and understanding of the research topic as well as her uncompromising and sensible attitude towards research and insistence on quality work have profoundly influenced us and will benefit our future work. Our heartfelt thanks to her painstaking modification of this report.

As a team we are grateful to our project Coordinators **Dr. Sharada K A.,** Professor, Dept of CSE and **Prof. Khallikkunaisa**, Associate professor, Department of CSE., HKBKCE. We are extremely thankful and indebted to them for sharing their expertise, valuable guidance, Suggestions, and encouragement extended to the team.

We would also like to thank all other teaching and technical staffs of Department of Computer Science and Engineering, who have directly or indirectly helped us in the completion of this Project Work. And lastly, we would hereby acknowledge and thank our parents who have been a source of inspiration and instrumental in the successful completion of this project.

ABSTRACT

Our endeavor, entitled "Enhancing Transportation Safety with YOLO-based CNN Autonomous Vehicle," embarks on a journey to redefine the landscape of autonomous driving by leveraging cutting-edge machine learning methodologies, notably the YOLO-based Convolutional Neural Network (CNN) algorithm. This innovative fusion of advanced algorithms with meticulously chosen hardware components, such as the robust Raspberry Pi 4B endowed with 8GB of RAM, fortified power supply, dependable battery system, indispensable IR sensors, and agile motor drives, sets forth to tackle paramount objectives encompassing object recognition and detection, lane delineation, wildlife awareness, traffic signal interpretation, and signboard comprehension. Our comprehensive approach entails extensive training and fine-tuning of the YOLO-based CNN algorithm using a diverse array of image datasets, thus culminating in the attainment of unparalleled levels of precision and efficacy in environmental perception and maneuvering. Noteworthy is the pivotal role played by IR sensors in facilitating meticulous lane detection, seamlessly complementing the algorithm's real-time object recognition prowess to ensure secure traversal through intricate traffic scenarios. Representing a groundbreaking milestone in the domain of autonomous driving technology, our project epitomizes a pioneering effort with far-reaching implications across a multitude of sectors, including automotive, logistics, and transportation. By placing paramount emphasis on safety and efficiency, our autonomous vehicle system heralds a new era in transportation systems, promising to significantly enhance road safety standards on a global scale. The transformative potential of our innovation underscores its capacity to revolutionize existing paradigms and catalyze positive.

TABLE OF CONTENTS

Chapter No.	Description	Page No.
	ACKNOWLEDGEMENT……	i
	ABSTRACT……...	ii
	TABLE OF CONTENTS…….....	iii
	LIST OF FIGURES……...	iv
	LIST OF TABLES…….	v
	ABBREVIATIONS…….....	vi
1	**INTRODUCTION**…….	1
	1.1 Self Driving Car……..	1
	1.2 Existing System……...	2
	1.3 Proposed System……..	3
	1.4 Scope of the project……..	5
	1.5 Summary……..	6
2	**LITERATURE REVEW**……..	7
	2.1 Literature review……..	7
	2.2 Summary……..	12
3	**METHODOLOGY**……..	13
	3.1 Software Requirements……..	13
	3.2 Hardware Requirements……..	13
	3.3 Introduction to Environment……..	13
	3.4 Our Approach……..	16
	3.5 Summary……..	19
4	**DESIGN AND DEVELOPMENT** ……..	20
	4.1 System Design……	20
	4.2 Architecture……	22
	4.3 Flowchart ……	23
	4.4 Sensor Based Control……	24
	4.5 Image Based Control (Animal / Lane / Signal……	25
	4.6 Use Case Diagram……..	26
	4.7 Source Code……..	27
	4.8 Summary……..	62
5	**RESULTS AND DISCUSSIONS**…….	63
	CONCLUSION…….	67
	REFERENCES…….	68

LIST OF FIGURES

Figure No.	Title	Page No.
1.1	Levels of automation	3
1.2	Self-driving Model	4
3.1	System Process	16
3.2	Raspberry Pi4 B	16
33	Pi Camera	17
3.4	Ultrasonic Sensor	17
3.5	Battery	17
3.6	Motor driver	18
3.7	12v Motor	18
3.8	Buzzer	18
4.1	System Design	20
4.2	System architecture	22
4.3	Working Flowchart	23
4.4	Sensor based control	24
4.5	Image Based Control	25
4.6	Use case diagram	26
5.1	Real time performance graphs of the vehicle	63
5.2	Real time Lane detection and following	64
5.3	Lane Detection through IR Sensors	65
5.4	Real time obstacle avoidance and overtaking	65
5.5	Getting back to its initial lane position	65
5.6	Real time Object detection and identifying	66
5.7	Traffic sign detection and following	66
5.8	Obstacle detection and avoidance	66

LIST OF TABLES

Table No.	Title	Page No.
5.1	Performance of YOLO based CNN using ML Techniques	64

ABBREVIATIONS

CNN	CONVOLUTIONAL NEURAL NETWORK
ML	MACHINE LEARNING
PSC	PI CAMERA
USS	ULTRASONIC SENSOR
MDM	MOTOR DRIVER MODULE
DVM	DC VOLT MOTOR
BZR	BUZZER
YOLO	YOU ONLY LOOK ONCE
CAN	CONROLLER AREA NETWORK
GPU	GRAPHICAL PROCESSOR UNIT
RPI	RASPBERRY PI
AI	ARTIFICIAL INTELLIGENCE
IR	INFRARED
CV	COMPUTER VISION

CHAPTER 1

INTRODUCTION

1.1 Self Driving Car

Road recognition is a critical issue for both road services and smart vehicles. Every year, a considerable amount of labour and budget is spent on road maintenance. A typical way to profile the pavement is to use a car equipped with certain devices to control the pavement change. These devices can be both visual, vibratory, and sensory. In Spite of the various potential benefits to increased vehicle automation, there are unresolved problems, Such as safety, technology issues, disputes concerning liability, resistance by individuals to forfeiting control of their carsisk of increased suburbanization as travel becomes less costly and time consuming,ith many related safety and security concerns.

The visual approach relies on image processing, using texture extraction. Koch et al. uses a high-speed wide-angle camera, which is mounted at the rear of the car and tilted down the road. Using vibration-based methods, the data collected is usually in the form of acceleration. The data collected can be obtained from professional equipment or mobile sensors.

Self-driving car (also known as a **robot car**, **Autonomous car**, or **Driverless car**) is a robotic vehicle that is designed to travel between destinations without human intervention. It is capable of sensing environment and navigate without human input.Autonomous cars must have control systems that are capable of analyzing sensor data to distinguish between different cars on the road.

The potential benefits of autonomous cars include reduced mobility and infrastructure costs, increased safety, increased mobility, increased customer satisfaction and reduced crime. Specifically, a significant reduction in traffic collisions; the resulting injuries; and related costs, including less need for insurance. Autonomous cars are predicted to increase traffic flow; provide enhanced mobility for children, the elderly and disabled; review travelers from driving and navigation chores; lowerlevel fuel consumption; significantly reduce needs for parking space; and facilitate business models for transportation as a service, especially via the sharing economy.This shows the vast disruptive potential of the emerging technology. In Spite of the various potential benefits to increased vehicle automation, there are unresolved problems, Such as safety, technology issues, disputes concerning liability, resistance by individuals to forfeiting control of their carsisk of increased suburbanization as travel becomes less costly and time consuming,ith many related safety and security concerns.

1.2 Existing System

In the dynamic realm of autonomous vehicles, the ongoing evolution of existing systems signifies a paradigm shift in the pursuit of elevating transportation safety standards. My project, titled "Enhancing Transportation Safety with YOLO-based CNN Autonomous Vehicle," delves into the intricate landscape of self-driving cars, unraveling the multifaceted layers that constitute their current state. One pivotal aspect of the prevailing system lies in the sophisticated sensor array employed by autonomous vehicles, including LiDAR and radar technologies. These sensors operate as the eyes of the self-driving car, capturing and deciphering real-time data from the surrounding environment, a crucial precursor to effective decision-making.

Another cornerstone of the existing self-driving car infrastructure is the reliance on advanced mapping technologies. These intricate maps, augmented with detailed information about road structures and potential obstacles, serve as the backbone for robust route planning and adept obstacle avoidance. Simultaneously, Global Positioning System (GPS) technology collaborates seamlessly, offering precise location data to ensure accurate navigation, thereby contributing to the overall efficiency and reliability of the autonomous vehicle.

Despite these notable advancements, challenges persist within the current self-driving car systems. Notably, the need for continual refinement in object detection and recognition algorithms remains paramount. While strides have been made in implementing sophisticated Computer Vision techniques, the accurate identification and swift response to dynamic elements in the vehicle's vicinity, such as pedestrians, cyclists, and unforeseen road conditions, warrant further attention for optimal safety outcomes.

Moreover, the integration of connectivity features in existing systems raises concerns about cybersecurity vulnerabilities. As autonomous vehicles become more interconnected, securing communication channels and fortifying data transfer mechanisms is imperative to forestall unauthorized access and manipulation of critical driving functions. Addressing these challenges head-on is pivotal for the seamless integration of self-driving cars into the broader transportation ecosystem, ensuring not only enhanced safety parameters but also bolstering public trust in the transformative potential of autonomous vehicle technology. The journey towards fully autonomous vehicles is an intricate dance between technological innovation, regulatory frameworks, and the imperative of fostering public confidence.

1.3 Proposed System

The envisioned self-driving car project represents a groundbreaking fusion of cutting-edge hardware and advanced deep learning methodologies. At its core is a Raspberry Pi, seamlessly integrated with a Pi cam positioned on the vehicle, forming the eyes through which the autonomous system perceives its surroundings. This dynamic setup is a testament to the project's commitment to robust image capture as the foundation for informed decision-making.

In establishing a robust communication channel, the Raspberry Pi and a connected laptop synchronize on the same network. This strategic connection facilitates the rapid transfer of real-time images captured by the Pi-cam to the Neural Network, laying the groundwork for instantaneous analysis and decision-making. This connectivity underscores the project's emphasis on creating a responsive and dynamic self-driving experience.

Figure 1.1 Levels of automation

The preprocessing phase emerges as a critical step in enhancing the model's adaptability and accuracy. Before entering the Neural Network, each captured image undergoes a meticulous grayscale transformation. This preprocessing step not only optimizes computational efficiency but also refines the input data, emphasizing essential visual features crucial for nuanced decision-making. The commitment to this preprocessing stage speaks to the project's dedication to achieving optimal performance in real- world scenarios.

Central to the proposed work is the Neural Network's capacity to decipher complex scenes and predict optimal driving actions. With four distinct output possibilities—left, right, forward, or stop—the model encapsulates a comprehensive understanding of the environment. This predictive prowess is a cornerstone of the self-driving car's ability to navigate diverse scenarios with a level of precision that instills confidence in its users.

The intricate dance between prediction and action unfolds seamlessly through a precisely designed signaling mechanism. Each predicted output triggers a corresponding signal, communicating directly with the car's controller to execute the intended maneuver. This harmonious integration between predictive analysis and real-time control epitomizes the project's commitment to creating an intelligent, responsive, and user-friendly autonomous driving experience.

Real-time decision-making stands out as a hallmark of the proposed model. Leveraging the Neural Network's capabilities, the system ensures swift and accurate responses to the dynamic challenges of the environment. This emphasis on real-time adaptability positions the project as a pioneering force in the quest for safer and more efficient autonomous transportation.

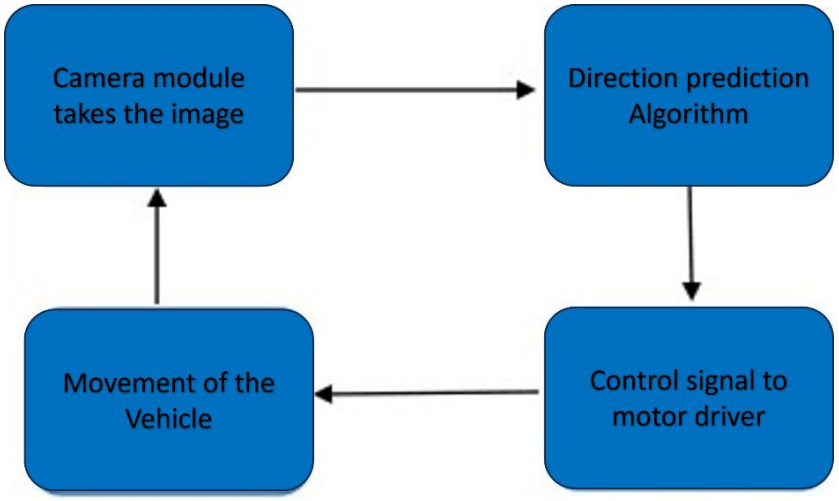

Figure 1.2 Self-driving Model

Beyond its technical sophistication, the project embodies a spirit of innovation and accessibility. The compact design, seamlessly integrating the Pi cam and Raspberry Pi, not only enhances portability but also opens avenues for scalable deployment across a spectrum of vehicles. This scalability broadens the project's potential applications, extending its impact beyond a singular context.

In conclusion, the proposed model for a self-driving car based on the YOLO algorithm is more than a technical marvel—it's a testament to human ingenuity and the relentless pursuit of creating intelligent, user-centric mobility solutions. By weaving together advanced hardware components and state-of-the-art neural networks, the project envisions a future where autonomous vehicles navigate our world with unmatched precision, safety, and allure.

1.4 Scope of the project

In embarking upon this expansive research initiative, the primary thrust is towards an exhaustive modeling of diverse parameters and the meticulous compilation of accident data. This holistic approach encompasses a multifaceted strategy, commencing with the scrupulous collection of historical accident data gleaned from authoritative sources. This reservoir of information serves as the foundation for subsequent analyses, notably the identification of blackspot sections, achieved through a rigorous examination of historical accident patterns.

Delving deeper, the research scrutinizes accident parameters intricately linked to causation factors. These include a granular investigation into variables such as vehicle speed, inter-vehicle gaps, hourly vehicular volume, and the presence of access points. Each parameter is dissected for its role in shaping the accident landscape, providing valuable insights into the complex dynamics of vehicular mishaps.

Moving beyond the theoretical realm, the research adopts a pragmatic approach through on-site surveys and exhaustive site studies. These endeavors encompass a thorough evaluation of road conditions and intricate traffic studies. Metrics such as speed studies, volume assessments, and gap analyses form an integral part of this empirical exploration. The blackspot area, identified along the federal route FT 050, becomes the focal point for these empirical investigations, with its geographical boundaries meticulously delineated using advanced tools like the Google Earth application.

The crux of this research lies in the aspiration to construct an advanced accident prediction model. This involves the intricacies of developing an architectural design for a neural network model, leveraging the sophistication inherent in the Artificial Neural Network model system. The aim is not merely to decode accident patterns but to synthesize a predictive tool capable of anticipating and preempting potential accidents. Such a model holds the promise of enhancing road safety through a nuanced understanding of contributing factors, heralding a transformative stride in the realm of accident prevention.

1.5 Summary

In the ever-evolving landscape of autonomous transportation, our project, "Enhancing Transportation Safety with YOLO-based CNN Autonomous Vehicle," stands as a beacon of innovation and technological prowess. Introducing a sophisticated integration of hardware and cutting-edge neural networks, our self- driving car project redefines the paradigm of autonomous navigation. Leveraging a Raspberry Pi and Pi cam duo, the system captures real-time images, initiating a seamless exchange with a connected laptop on the same network. The Neural Network, a computational virtuoso, interprets grayscale images with remarkable accuracy, orchestrating split-second decisions that direct the car's every move – be it a graceful left turn, a precise right turn, a steady forward progression, or a poised complete stop. This dynamic interplay between prediction and action, synchronized through tailored signals, transforms the car into a sentient entity navigating the roads with finesse and responsiveness. Real-time decision-making and adaptability further elevate the project, ensuring safety in the face of unpredictability. Beyond its functional brilliance, our model embodies portability and scalability, envisioning a future where autonomous navigation transcends beyond cars to revolutionize transportation universally. In essence, our self-driving car project epitomizes not just autonomy but a symphony of technological innovation, where vehicles intelligently engage with their environment, redefining the future of transportation.

CHAPTER 2

LITERATURE REVEW

2.1 Literature review

[1] The research work Titled "Deep Learning Techniques for Obstacle Detection and Avoidance in Driverless Cars" It proposes data points are required to build a neural network model. The information is collected via the training procedure. The car is be operated wirelessly. In order to complete this, we used VNC viewer, which gives us tpo operate Raspberry Pi remotely using Wi-Fi in gadgets. Building a track is the next phase toward having the automobile trained. The Raspberry Pi camera and the input commands (advance, stop, right, left) is used to capture pixel data then the car is being driven on the track. The neural network model will get trained using this collection of dataset, this convolutional neural networks, or CNNs, has become to be widely used. The reason behind CNN is that it consistently produces accurate object recognition and identification which results whose practical for daily use. Convolutional networks are used in classification tasks, for which an image's provided output is a single class label that shall to be classified. Convolution is a function that represents that the shape of one function which is now is changed by the another and is collected by two supplied functions via through integration. In this setup CNN uses images as its input, not the following neural networks.

[2] The research work Titled "Solar Operated IoT based Smart System to Monitor Illegal L[2]The research work Titled "Real-time Pothole Detection using Deep Learning" The study utilizes two datasets: an online database and a collection of images from Lebanese roads. Over 1,000 images, featuring more than 2,000 potholes, are used for analysis. YOLO, or You Only Look Once, is highlighted as a powerful object detection algorithm that only requires a single scan of an image to predict objects and their positions, providing a speed advantage over other models. The latest iteration, YOLOv4, released in 2020, is claimed to be 10 percent better in Average Precision (AP) and 12 percent faster than its predecessor, YOLOv3.

[3] The research work Titled "Self-Driving Car: Using OpenCV2 and Machine Learning" issued by Shoeb, Mohammed, Mohammed Akram Ali, Mohammed Shadeel, and Dr Mohammed Abdul Bari. The car's input is a continuous stream of images captured by the Raspi cam2, which involves installing and building necessary camera libraries in the operating system. Algorithms for image and video capturing, along with frame-per-second calculation, are implemented. Since CV2 operates on BGR images, a conversion from BGR to RGB is done.

[4] The research work Titled "Exploiting the Joint Potential of Instance Segmentation and Semantic Segmentation in Autonomous" It proposes the developing semantic segmentation network architectures for the dual backbone of the network designs. which are Broadened in the convolutions within final residual blocks, which provide the context capture while keeping the final output's decisiveness eight to sixteen times lower than input picture, Due to which the best-performing designs are Retaining detailed information which consist of balacing high-resolution feature responses, that facilitates the process of spatial dimension and object limits of the design.

[5] The research work Titled "Probabilistic logic Markov decision processes for modeling driving behaviors in self-driving cars" Development of the Probabilistic Logic MDP In our study, we focus on the intricate task of modeling driving behaviors for autonomous vehicles using Probabilistic Logic Markov Decision Processes (MDP). Our scenario involves an autonomous vehicle maintaining a constant speed, with decisions to be made regarding whether to continue at a steady pace, overtake, or adjust its distance based on the presence of other vehicles nearby. illustrates some of the states considered to construct the environment. Moving forward, we refer to North, North-West, West, and South-West as N, NW, W, and SW, respectively. The states are characterized by four Boolean state variables: free N, free NW, free W, and free SW, representing regions around the vehicle that can be either free or occupied. In MDP-ProbLog, the declaration of state variables is succinct, for instance, " state fluent(free N)" defines the variable free N. Similarly, action declarations are straightforward, such as" action(keep distance)."

[6] The research work Titled "Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis", In this study, the authors harnessed the power of advanced deep learning techniques for object detection, particularly focusing on the CenterNet framework. CenterNet, an improvement upon CornerNet, employs a triplet instead of a pair of keypoints for detecting objects within cropped images. This modification enhances both precision and recall values, addressing the limitations of CornerNet, which struggles to provide a global perspective of objects. CenterNet's intuition relies on the higher Intersection over Union (IoU) with groundtruth boxes, making it superior for anchor-based detection approaches.

[7] The research work Titled "SDR -Self Driving Car Implemented using Reinforcement Learning and Behavioural Cloning" issued by Sanjay S Tippannavar, Yashwanth S D, Puneeth K M. Introducing DRIVE-IT (Driving in Realistic Interactive Virtual Environments), our novel approach combines cutting-edge technologies to redefine the training, development, and validation of autonomous driving models. DRIVE-IT incorporates CARLA (Car Learning to Act), an open simulator meticulously crafted by a team of digital artists, offering free access to urban environment assets.

[8] The research work Titled "Image recognition in self-driving cars using CNN" It proposes multiple layers of artificial neurons that are made up the neural networks collectively known as nodes. It simply extracts minimal features just like horizontal or diagonal edges in the top layer. The corresponding output is forwarded to the upcoming next layer which detects lot more complex features some them are as corners or combinational edges. The neural network is highly more capable of identifying more complex features as we go deeper into the objects, and so on. Hidden layers in CNNs are is composed of convolutional layers with inputs and the outputs.

[9] The research work Titled "Image recognition in self-driving cars using CNN" It proposes multiple layers of artificial neurons that are made up the neural networks collectively known as nodes. It simply extracts minimal features just like horizontal or diagonal edges in the top layer. The corresponding output is forwarded to the upcoming next layer which detects lot more complex features some them are as corners or combinational edges. The neural network is highly more capable of identifying more complex features as we go deeper into the objects, and so on. Hidden layers in CNNs are is composed of convolutional layers with inputs and the outputs. The ReLU is the activation function, followed by pooling layers, which are fully connected layers, and normalization layers as normalilzed. Thus back propagation method is used for adjusting weights and errors. More layers and more networks are more efficient in convolutional neural networks, i.e., the larger dataset, the more and more greater efficiency.

[10] The research work Titled "Toward Attack Modeling Technique Addressing Resilience in Self- Driving Car" The proposed model, known as Severity-based Analytical Attack Model for Resilience (SAAMR), draws inspiration from existing frameworks like STRIDE, STAMP, PASTA, and CVSS. STAMP was chosen for its ability to model feedback mechanisms and accidental disturbances. This study assesses both accidental and deliberate disruptions to the system, using STAMP to map events, with the exception of malicious attempts. To address this, STRIDE was incorporated and modified to account for malicious attacks, specifically tailored to Self-Driving Car (SDC) scenarios based on a comprehensive literature review of vulnerabilities. The architecture of SDC is classified into three layers: perception, decision-control, and action layers, based on autonomy levels defined by the Society of Automotive Engineers (SAE). The perception layer collects data through sensors, creating state awareness, and includes sensor-interface and sensor-fusion modules. The decision control layer processes this data for decision-making, and the action layer executes actions. The STAMP model is employed to identify potential system failures in SDC. Despite STAMP's primary focus on socio-technical systems, the proposed model limits its scope to automated feedback control, aligning with the autonomous nature of SDC Level 5.

[11] The research work Titled "An Improved Deep Network Based Scene Classification Method for Self-Driving Cars" It proposes a novel deep network for scene classification in self-driving cars. The architecture comprises four key components: an enhanced Faster RCNN, an improved Inception V1 module, a feature fusion module, and a classification network. The Faster RCNN is upgraded with a spatial attention based residual connection module, enhancing its performance. Additionally, the Inception V1 module incorporates a mixed activation function using ELU and Leaky ReLU. The enhanced Faster RCNN, pretrained and outputting features of predefined representative objects, plays a pivotal role. Seven common traffic scene objects—zebra crossing, pedestrian, gas tank, parking car, parking line, house, and isolation belt—are predefined and serve as labels for network training. The scenes considered are crosswalk, gas station, parking lot, street, and highway.

[12] The research work Titled "Fixed Settling Time Control for Self-Driving Car: Two-Timescales Approach" It proposes a control law for a system with two-time scales. The control strategy involves two distinct signals, each designed for the fast and slow systems described in these equations. This composite control, designed for a two-time scale system, consists of two control signals catering to the fast and slow systems mentioned earlier. The subsequent part discusses the application of fixed time Sliding Mode Control (SMC) with a barrier function. This control design initially focuses on the quasi- steady-state model and then extends to the boundary layer model. The fixed settling time SMC with a barrier function is developed by considering linear sliding surfaces for both the slow and fast systems. Regarding the fixed-time stability concept by Polyakov, equations (40-41) are presented to demonstrate that trajectories converge to corresponding sliding manifolds within a fixed time. Once the sliding mode is achieved, the construction of the sliding manifold ensures exponential stability. Consequently, achieving convergence of outputs to a error bound in fixed time only requires the maximum output at the time when Si equals 0.

[13] The research work Titled "Pothole and plain road classification using adaptive mutation dipper throated optimization and transfer learning for self-driving cars." The proposed methodology for pothole detection in road images presents a comprehensive approach to address the challenges in this domain. The process begins with data augmentation, where a set of relevant features is extracted from input images using a ResNet-50 deep neural network. Feature selection is then performed using a novel binary particle swarm optimization algorithm (bPSDTO), and the dataset is balanced using an optimized SMOTE algorithm based on locality-sensitive hashing. The optimization process involves a hybrid approach combining particle swarm optimization (PSO) and dipper-throated optimization (DTO). The resulting features are utilized to train a random forest model, and its parameter.

[14] The research work Titled "Attacks on Self Driving Cars and Their Countermeasures: A Survey" It describes the overarching structure of Intelligent Transportation Systems (ITS), divided into three primary domains: the vehicle domain, the Vehicle-to-Vehicle (V2V) domain, and the infrastructure domain. These domains communicate through in-vehicle and V2X communication, encompassing V2V, V2I (Vehicle-to-Infrastructure), and I2V (Infrastructure-to-Vehicle) communication. Within the vehicle domain, an Onboard Unit (OBU) is installed to facilitate communication between applications. The V2X realm establishes a dynamic system involving OBUs and Roadside Units (RSUs) strategically placed along highways, ITS paths, and rail networks. Vehicular messages (V2X) serve as wireless communication means for data exchange between OBUs and neighboring ITS units. The RSU, a central element in the automated vehicles field, is positioned on roadsides or yellow curbs, connecting vehicles via their OBUs.

[15] The research work Titled "A Self-Driving Decision Making with Reachable Path Analysis and Interaction Aware Speed Profiling." In recent literature, substantial research efforts have been directed towards the development of decision-making algorithms tailored for autonomous vehicles. This paper classifies these algorithms into three primary categories: rule-based, graph-search, and sampling- based methods. The establishment of traffic rules serves as a pivotal reference for autonomous driving behavior. Among the commonly employed approaches for implementing traffic rules, the rule-based methodology stands out, often utilizing a finite state machine (FSM). FSM involves defining states associated with the autonomous driving service and determining state transitions based on a flow chart. Despite its straightforward implementation, FSM exhibits limited adaptability to varying road conditions, making it susceptible to errors. For optimal decision-making in autonomous driving, graph search and Markov Decision Process (MDP) approaches are widely utilized. The graph-search method, aimed at deriving the optimal vehicle position sequence, encompasses state lattice, elastic band, and A- star algorithm. These methods yield a sequence of optimally selected positions on the road surface. McNaughton et al. proposed a motion planning approach using a conformal spatiotemporal lattice for optimal trajectory planning in complex driving scenarios.

2.2 Summary

The extensive literature survey explores diverse aspects of self-driving cars, spanning deep learning techniques for obstacle detection, real-time pothole detection using algorithms like YOLOv4, and the integration of machine learning with OpenCV2 for lane and object detection. Additional focus areas include joint instance and semantic segmentation, probabilistic logic Markov decision processes, and a comparative analysis of object detection and tracking algorithms like CenterNet, Detectron2, YOLOv4, and EfficientDet. The survey delves into DRIVE-IT, a novel simulator combining CARLA with realistic virtual environments, and examines image recognition methodologies using CNNs and YOLO. Noteworthy models like Severity-based Analytical Attack Model for Resilience (SAAMR) and an improved deep network-based scene classification method are covered. The survey also discusses fixed settling time control, innovative approaches for pothole and plain road classification, and provides a comprehensive overview of self-driving car advancements from 15 distinct research papers.

CHAPTER 3

METHODOLOGY

3.1 Software Requirements:

- Makesense.ai
- Roboflow.com
- Google colab
- Python -version 3 and higher
- YOLO v5 packages
- Python packages
- Raspbian OS

3.2 Hardware Requirements:

- Raspberry Pi 4B 8gb ram
- Web Camera
- Ultrasonic Sensor
- Battery
- Motor driver (L293D)
- 12v-5v-3v Motor
- Buzzer
- Voltage Converter
- 12v Battery
- Power bank
- IR Sensors

3.3 Introduction to Environment

Machine Learning is a subset of artificial intelligence that enables machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed. It combines computer science and statistics for creating predictive models.

Deep learning, a subset of machine learning, revolves around neural networks with multiple layers, enabling the modeling of intricate patterns and representations. These networks, inspired by the human brain, learn hierarchies of features from data, adapting and improving performance over time. In essence, deep learning excels in tasks requiring complex pattern recognition, making it pivotal in various fields.

Significantly, deep learning finds applications in image and speech recognition, natural language processing, and autonomous systems. In Python, frameworks like TensorFlow and PyTorch provide powerful tools for implementing deep learning models. Python's simplicity and extensive libraries make it a preferred language for deep learning, fostering innovation and advancements in artificial intelligence.

According to Google trends and GitHub, the top language for Machine Learning is Python. It is the primary programming language used for much of the research and development in machine learning. Some of the reasons why python is preferred for machine learning tasks are:

- Python is easy to use:
  Machine Learning is simply recognizing patterns in your data to be able to make improvements saving time which in turn makes Python even more popular. There are many Python libraries that are specifically useful for Artificial Intelligence and Machine Learning.
- Python is portable and extensible:
  This is an important reason why Python is so popular in Machine Learning. A lot of cross-language operations can be performed easily on Python because of its portable and extensible nature. There are many data scientists who prefer using Graphics Processing Units (GPUs) for training their ML models on their own machines and the portable nature of Python is well suited for this. Also, many different platforms support Python such as Windows, Macintosh, Linux, Solaris, etc. In addition to this, Python can also be integrated with Java, .NET components or C/C++ libraries because of its extensible nature.
- Community:
  Python provides broad support. Because a lot of people, both programmers and average users, view Python as a standard, its support community is huge, increasing Python's popularity even more.

Some of the Python libraries that we plan to use in our project are:

- **YOLO:**
  1. Definition:
     YOLO is an algorithm that divides an image into a grid and predicts bounding boxes and class probabilities for objects within each grid cell in a single forward pass.

  2. Usage in Self-Driving Vehicles:
     - YOLO is employed for real-time object detection in the surrounding environment of self- driving vehicles.
     - It helps in identifying and tracking various objects like pedestrians, vehicles, and obstacles.

  3. Benefits:
     - Speed: YOLO is known for its speed, enabling real-time processing, which is crucial for applications like autonomous driving.
     - Accuracy: By simultaneously predicting multiple bounding boxes

and efficient object detection.

- Unified Approach: YOLO provides a unified framework for object detection, making it a popular choice for applications where real-time processing is essential.

Points to consider:

- YOLO can handle multiple objects within an image in a single pass, making it efficient for real-time applications.
- The algorithm predicts bounding boxes, class probabilities, and confidence scores for detected objects.
- YOLO architecture has undergone several versions (e.g., YOLOv1, YOLOv2, etc.), each with improvements in speed and accuracy.
- It's commonly used in various fields beyond self-driving vehicles, such as surveillance, robotics, and image recognition.

These features collectively contribute to YOLO's effectiveness in detecting and classifying objects swiftly, making it suitable for real-world applications like self-driving vehicles.

- **NumPy:**

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

Key Features:

- Supports n-dimensional arrays to enable vectorization, indexing, and broadcasting operations.
- Supports Fourier transforms mathematical functions, linear algebra methods, and random number generators.
- Implementable on different computing platforms, including distributed and GPU computing.
- Easy-to-use high-level syntax with the optimized Python code to provide high speed and flexibility.
- In addition to that, NumPy enables the numerical operations of plenty of libraries associated with data science, data visualization, image processing, quantum computing, signal processing, geographic processing, bioinformatics, etc. So, it is one of the versatile machine learning libraries.

- **Scikit-learn:**

  Scikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool for someone who is starting out with ML.

  Key Features:

  - Easy to use for precise predictive data analysis
  - Simplifies solving complex ML problems like classification, pre-processing, clustering, regression, model selection, and dimensionality reduction
  - Plenty of inbuilt machine learning algorithms
  - Helps build a fundamental to advanced level ML model.

3.3 Our Approach

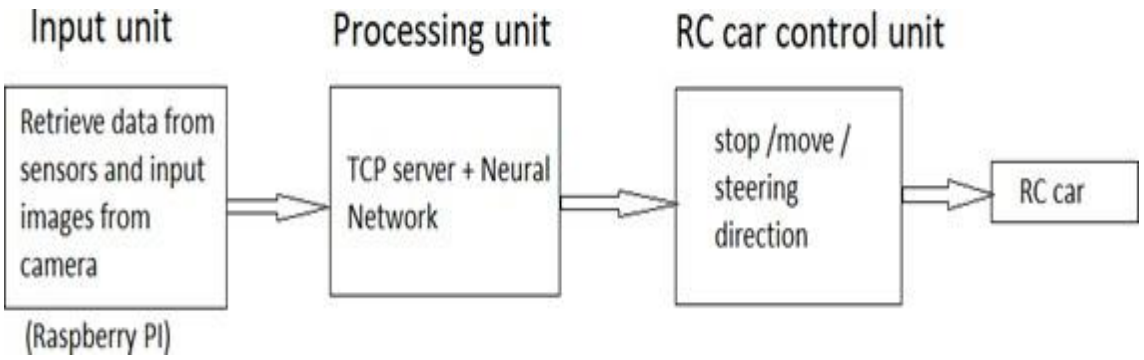

Figure 3.1 System Process

- **Raspberry Pi 4 B:** The Raspberry Pi 4 B, a powerhouse in a compact form, serves as the brain of our system. With its quad-core processor and enhanced memory, it orchestrates seamless communication and computation, laying the foundation for intelligent decision-making in our project.

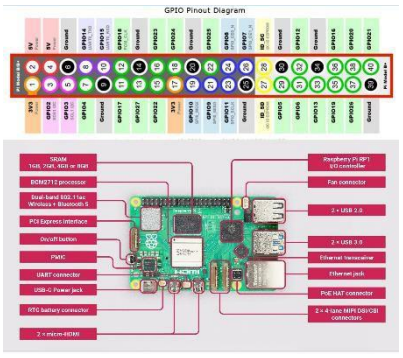

Figure 3.2 Raspberry Pi 4 B

- **Web Camera:** The Pi Camera, our eyes in the project, captures the world with precision. Its high- resolution imaging and adaptability make it the perfect tool for real-time visual data acquisition, enabling our system to perceive and respond to its surroundings effectively.

Figure 3.3 Pi Camera

- **Ultrasonic Sensor:** The Ultrasonic Sensor, a silent guardian in our setup, plays a pivotal role in environmental awareness. With accurate Ultrasonic index measurements, it ensures our project is not only responsive but also considers the well-being of its surroundings, demonstrating a commitment to safety and sustainability.

Figure 3.4 Ultrasonic Sensor

- **Battery**: Powering our project's mobility, the Battery is the lifeblood that fuels uninterrupted operations. Its high capacity and efficient output management guarantee prolonged autonomy, making our system reliable and resilient in diverse scenarios.

Figure 3.5 Battery

- **Motor driver (L293D):** The L293D-driven DC Motor is the powerhouse behind our project's physical movements. This high-performance motor driver ensures precise control and maneuverability, transforming our system from a passive observer to an active participant in its environment.

Figure 3.6 Motor driver

- **12v Motor:** 12-volt DC motors play a crucial role in the development of self-driving vehicles due to their compact size, efficiency, and compatibility with standard automotive electrical systems,

  contributing to the overall reliability and cost-effectiveness of autonomous vehicle components.

Figure 3.7 12v Motor

- **Buzzer:** In self-driving vehicles, buzzers serve as essential auditory components for communication and safety alerts. These compact devices are integrated to provide audible

  notifications, such as collision warnings, lane departure alerts, and system status updates, enhancing both user awareness and overall safety in autonomous driving scenarios.

Figure 3.8 Buzzer

3.5 Summary

In the methodology chapter, we meticulously outlined our approach, hardware, and software requirements for the project. Our hardware arsenal boasts essential components like the web Camera, Ultrasonic Sensors, Raspberry Pi 4B, DCL 293 motors, actuators, sensors, and a Wi-Fi module, synergizing seamlessly to form the backbone of our system. This hardware symphony collaborates flawlessly with the software ensemble, featuring powerhouse tools such as CNN, YOLO, Anaconda, and the Anaconda Environment. Each element was chosen with precision to ensure optimal performance and reliability.

Our approach intricately weaves together these hardware and software facets, creating a robust framework for our self-driving car project. From the nuanced integration of sensors for real-time data acquisition to the implementation of sophisticated algorithms like CNN and YOLO, our methodology is a testament to our commitment to cutting-edge technology. The inclusion of Raspberry Pi 5 adds a layer of versatility, enabling seamless communication and control within our system.

As we delve into the specifics of our methodology, the meticulous selection of components and the strategic interplay between hardware and software become evident. Our project unfolds as a symphony of technology, orchestrated with care to navigate the complexities of autonomous driving. The methodology chapter stands as a comprehensive guide, a result of thoughtful planning and meticulous execution, mirroring the essence of a project crafted with passion and precision.

CHAPTER 4

DESIGN AND DEVELOPMENT

4.1 System Design

System design is the process of the defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development. Object- oriented analysis and methods are becoming the most widely used methods for computer systems design. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user. The UML has become the standard language in object -oriented analysis and design.

System architecture is a conceptual model that defines the structure and behavior of the system. It comprises of the system components and the relationship describing how they work together to implement the overall system.

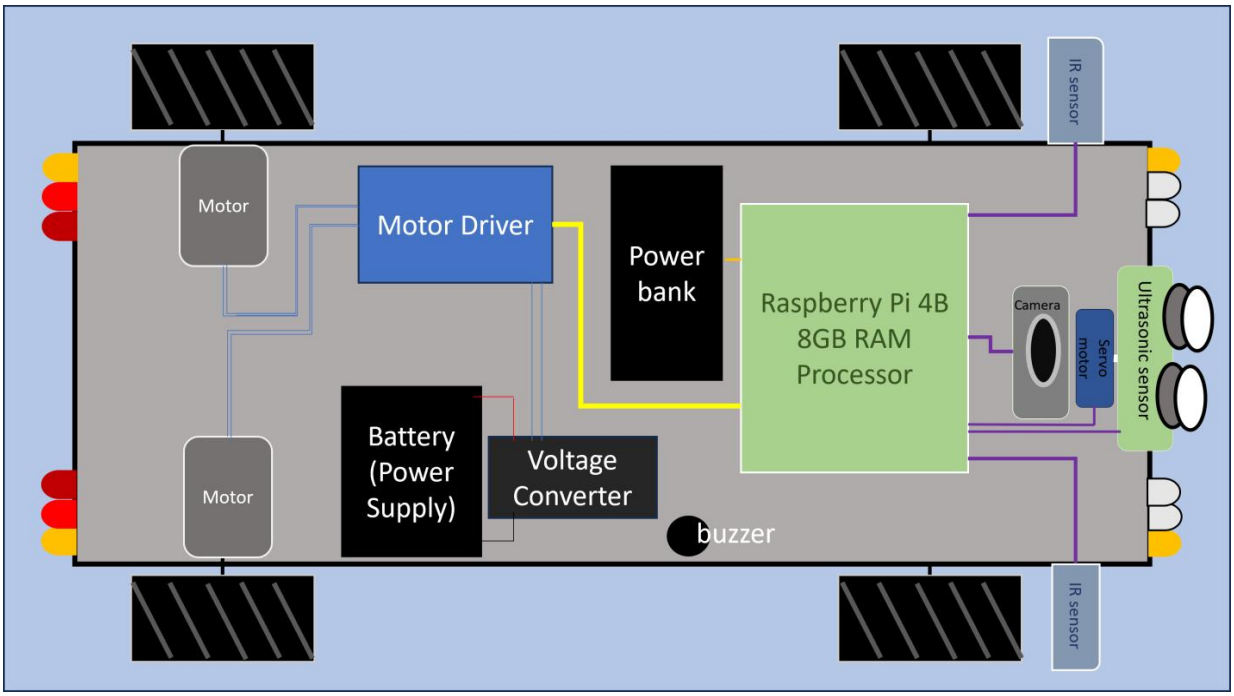

Figure 4.1 System Design

- **Servo motor:** Incorporating a Servo Motor at the forefront of our model, we elevate the capabilities of our ultrasonic sensor, enabling dynamic scanning of the environment. By mounting the ultrasonic sensor on the Servo Motor, we empower it to rotate and survey its surroundings with precision, enhancing the vehicle's perception and navigation.

- **Camera:** The camera plays a vital role in enabling the vehicle to recognize key elements of its environment essential for safe navigation. It is instrumental in identifying traffic signs, lane markings, potholes, direction indicators, animals such as cows and dogs, and various traffic signals, including red lights. Bycapturing high-quality visual data, the camera al- lows thevehicle's onboard systems to analyze and interpret the surroundings accurately, aiding in efficient decision-making and ensuring the safety of the vehicle and its occupants.

- **Raspberry Pi 4B 8GB RAM Processor:** The Raspberry Pi 4B with 8GB RAM is chosen for its powerful computing capabilities, enabling efficient processing of sensor data and complex algorithms essential for real-time decision-making inautonomous driving scenarios. Its high memory capacity mooth multitasking and seamless execution of machine learning algorithms, contributing to enhanced navigation accuracy and overall safety of the vehicle

- **Motor Driver:** The motor driver controls the movement of the vehicle's motors, translating commands from the Rasp- berry Pi into physical motion. It ensures precise and reliable control of the vehicle's speed and direction, enabling smooth navigation and maneuvering in various driving conditions.

- **Voltage Converter:** The voltage converter regulates the powersupply to the vehicle's components, ensuring stable and consistent voltage levels. It protects sensitive electronics from voltage fluctuations and provides reliable power distribution throughout the vehicle's electrical system.

- **Battery:** The battery serves as the primary power source for the autonomous vehicle, supplying energy to all its components. It provides the necessary electrical power to drive the motors, operate the sensors, and run the onboard computer, enabling sustained operation without external power sources.

- **Buzzer:** The buzzer serves as an auditory alert system in our autonomous vehicle project, enhancing safety measures during navigation. When the vehicle detects an obstacle while in motion, the buzzer emits a distinct sound signal for a duration of 2 seconds, signaling the vehicle to come to a halt. This audible alert not only alerts nearby pedestrians and vehicles but also provides feedback to the vehicle's occupants. Once the warning period concludes, the vehicle proceeds to navigate around the obstacle, ensuring smooth and efficient operation while prioritizing safety.

- **PowerBank:** The Power Bank serves as a portable power source for our autonomous vehicle project, providing backup energy to ensure uninterrupted operation It serves as a reliable power supply, allowing the vehicle to function even in scenarios where the primary power source is unavailable or depleted. The PowerBank enhances the vehicle's versatility and reliability, making it suitable for extended periods of operation without external power sources.

- **LED:** LEDs are utilized in our project to provide illumination and visual indicators for various functions. Positioned strategically around the vehicle, LEDs enhance visibility in lowlight conditions and improve the vehicle's visibility to other roadusers. Additionally, LEDs serve as status indicators, providing feedback on the vehicle's operational state and assisting in diagnostics and troubleshooting.

- **Robot Frame and Wheels:** The robot frame serves as the structural foundation for our autonomous vehicle, providing support and stability for mounting all components. Constructed from durable materials, the robot frame ensures the integrity andlongevity of the vehicle, withstanding the rigors of regular operation. The wheels, integral to the vehicle's mobility, enable smooth movement and navigation across various sur- faces. Designed for optimal traction and maneuverability, the wheels facilitate precise control and ensure efficient traversal of diverse terrain, enhancing the vehicle's overall performance and versatility

4.2 Architecture:

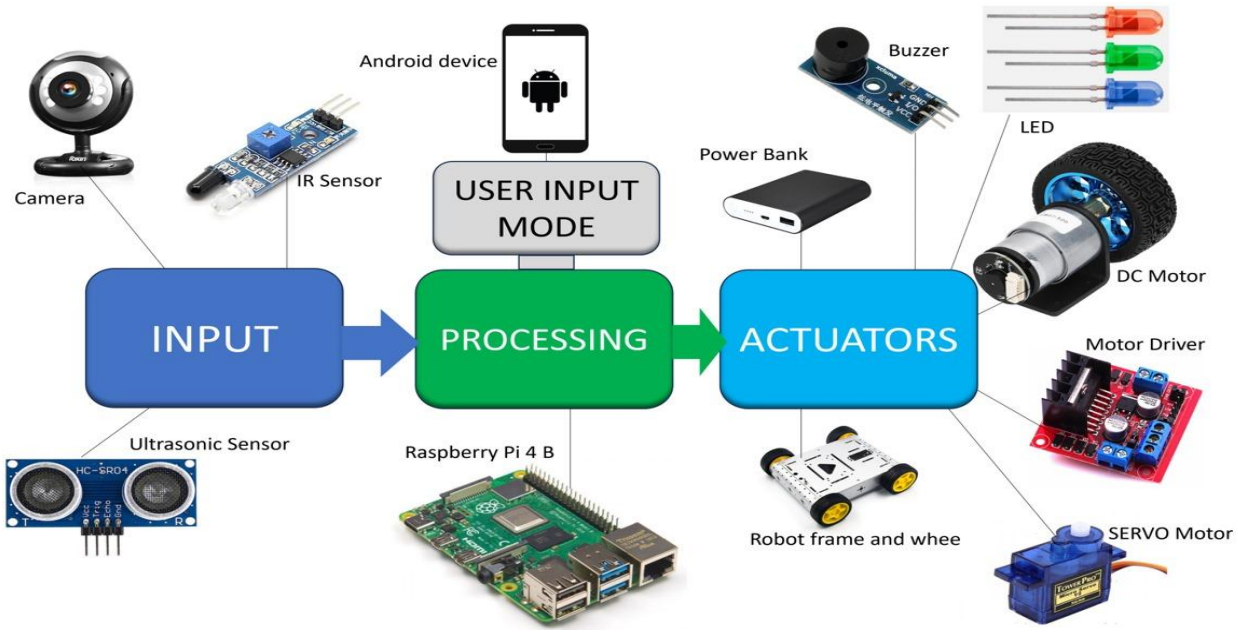

Figure 4.2 System architecture

The architecture data has two outlines crucial aspects of autonomous vehicle (AV) technology: mapping and external sensation/perception. Mapping serves as a foundational element for AV trips, offering vital information about permanent and semi-permanent structures, road networks, traffic laws, and more. Maps come in various formats, including 2D, 3D, and graph representations, with common attributes such as pedestrian crossings, traffic lights, road signs, and barriers. During AV trips, maps play key roles in localization, where sensed data is referenced to map data, and planning, influencing global routes and local plans. External sensation/perception involves sensors like cameras, each providing distinct data types such as high-resolution images, precise distance measurements, and speed measurements, respectively. Sensor data can be processed individually or jointly after fusion, with techniques including object/semantic detection and monitoring/tracking. This comprehensive integration of mapping and sensor perception forms the backbone of autonomous vehicle navigation, facilitating safe and efficient travel through complex environments.

4.3 Flowchart:

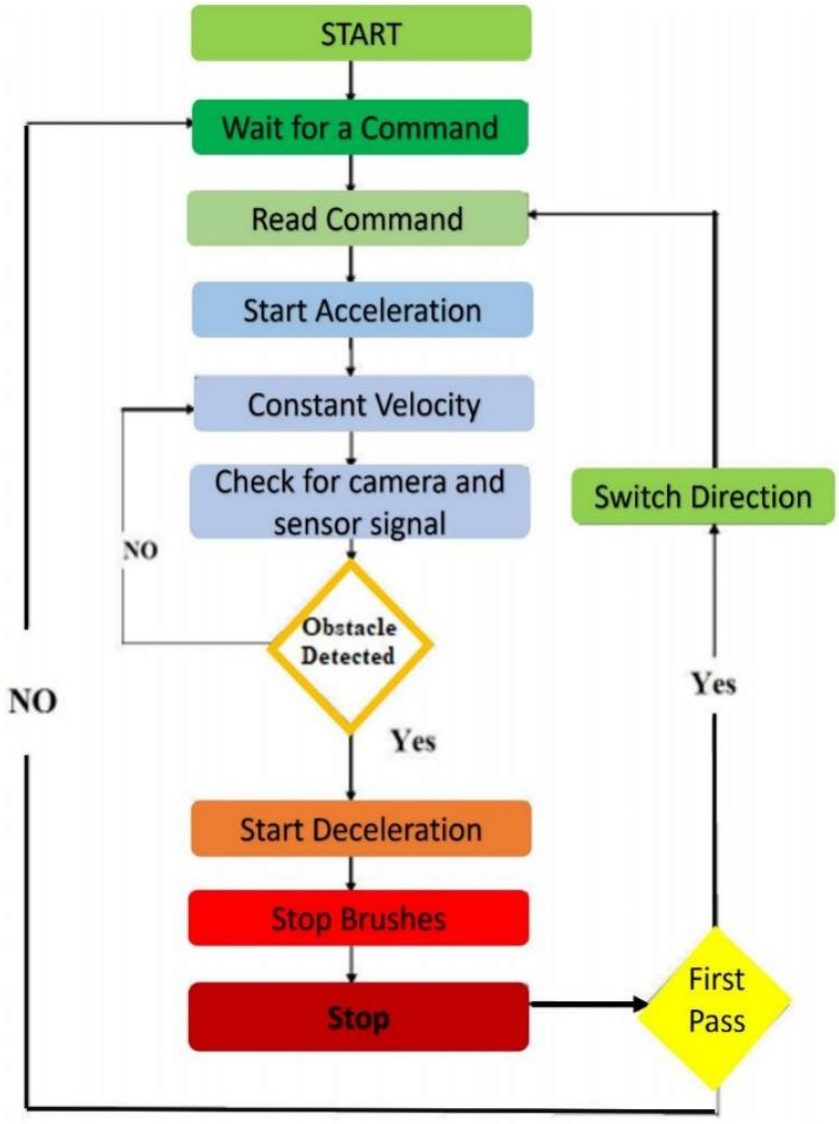

Figure 4.3 Working Flowchart

- **Start:** The process begins when the robot is powered on.
- **Wait for a Command:** The robot remains idle until it receives a command to start cleaning.
- **Read Command:** Upon receiving a command, the robot interprets the instructions.
- **Start Acceleration:** The robot begins to accelerate from a stationary position to its working speed.
- **Constant Velocity:** Once the desired speed is reached, the robot maintains a constant velocity for efficient cleaning.
- **Check for Camera and Sensor Signal:** The robot continuously checks its camera and sensors for any signals that might indicate obstacles or changes in the environment.
- **Switch Direction:** If no obstacle is detected, the robot may switch direction to cover more area or to follow a cleaning pattern.
- **Obstacle Detected:** If an obstacle is detected, the robot must decide how to respond.
- **Start Deceleration:** The robot starts to slow down to avoid a collision with the obstacle.
- **Stop Brushes:** The cleaning brushes are stopped to prevent damage or entanglement with the obstacle.
- **Stop:** The robot comes to a complete stop to assess the situation.

4.4 Sensor Based Control:

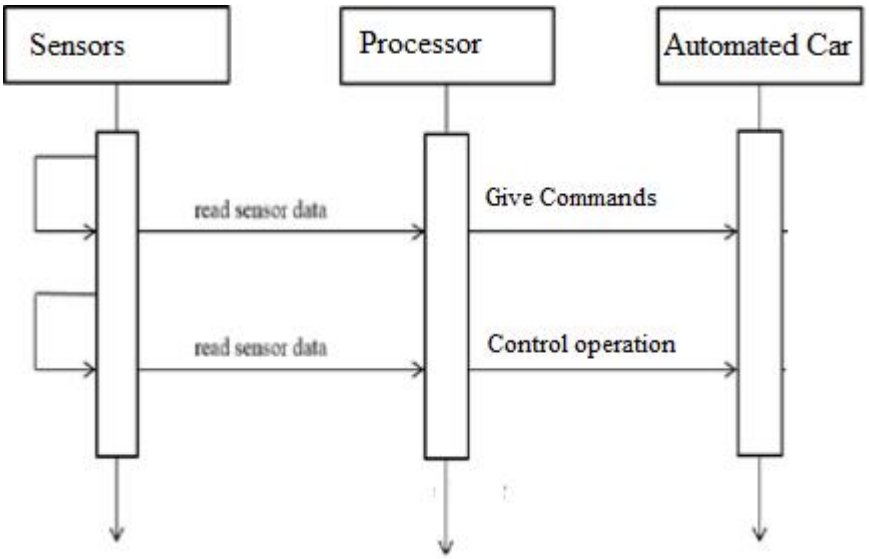

Figure 4.4 Sensor based control

The Sensor based control provides a comprehensive overview of the operational process underlying an automated car, delineating the interplay between its fundamental components. At the core of this system are the sensors, strategically positioned throughout the vehicle to capture and relay essential data from the surrounding environment. These sensors serve as the sensory organs of the automated car, detecting and interpreting a myriad of environmental cues such as

nearby objects, road conditions, and potential obstacles. The collected data is then seamlessly transmitted to the processor, a sophisticated computational unit tasked with the intricate task of data analysis and decision-making. Upon receiving the sensor data, the processor embarks on a multifaceted process of data processing, employing advanced algorithms and machine learning techniques to extract valuable insights and discern patterns within the incoming information. This analytical prowess enables the processor to generate precise commands and directives tailored to the real-time demands of the driving scenario. Finally, these meticulously crafted commands are relayed to the automated car, serving as the guiding force behind its navigation and operational functions. From steering and acceleration to braking and collision avoidance, the automated car diligently executes the instructions received from the processor, orchestrating a harmonious interaction between technology and environment. In essence, the flowchart encapsulates the intricate choreography of sensors, processor, and automated car, underscoring the seamless integration of cutting-edge technology to navigate the complexities of the modern roadways with unparalleled precision and efficiency.

4.5 Image Based Control (Animal / Lane / Signal):

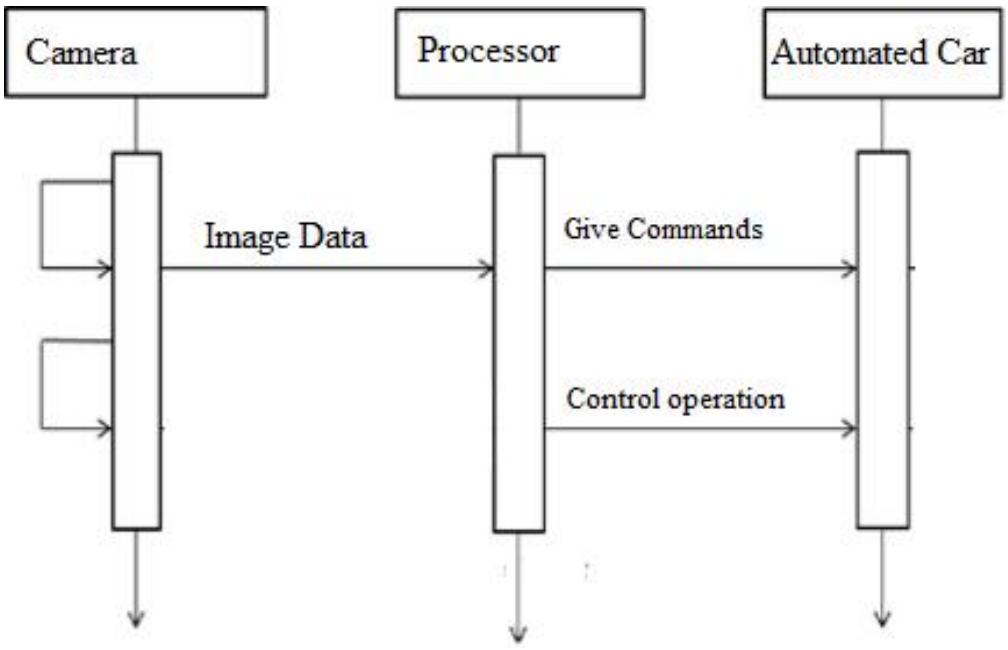

Figure 4.5 Image Based Control

The flow diagram provides a detailed insight into the operational dynamics of an automated car, delineating the pivotal role played by its three main components: the camera, processor, and the automated car itself. At the forefront of this system is the camera, acting as the primary sensory apparatus responsible for capturing image data from the car's surroundings. Positioned strategically within the vehicle, the camera serves as the eyes of the automated car, continuously

scanning the environment and relaying visual information to the onboard processor. The processor, a sophisticated computational unit, assumes the critical responsibility of interpreting the captured image data and transforming it into actionable insights. Employing a myriad of algorithms and analytical techniques, the processor meticulously analyzes the visual input to discern pertinent details such as road conditions, traffic patterns, and the presence of obstacles. Based on this comprehensive analysis, the processor formulates precise commands and directives tailored to the real-time demands of the driving scenario. These commands are then seamlessly transmitted to the automated car, which acts as the physical manifestation of the processor's decision-making capabilities. From steering and acceleration to braking and collision avoidance, the automated car diligently executes the instructions received from the processor, orchestrating a seamless interaction between technology and environment. In essence, this elaborate process underscores the symbiotic relationship between cameras, processors, and automated cars, underscoring their collective role in enabling safe, efficient, and autonomous navigation on the roads of the future.

4.6 Use Case Diagram:

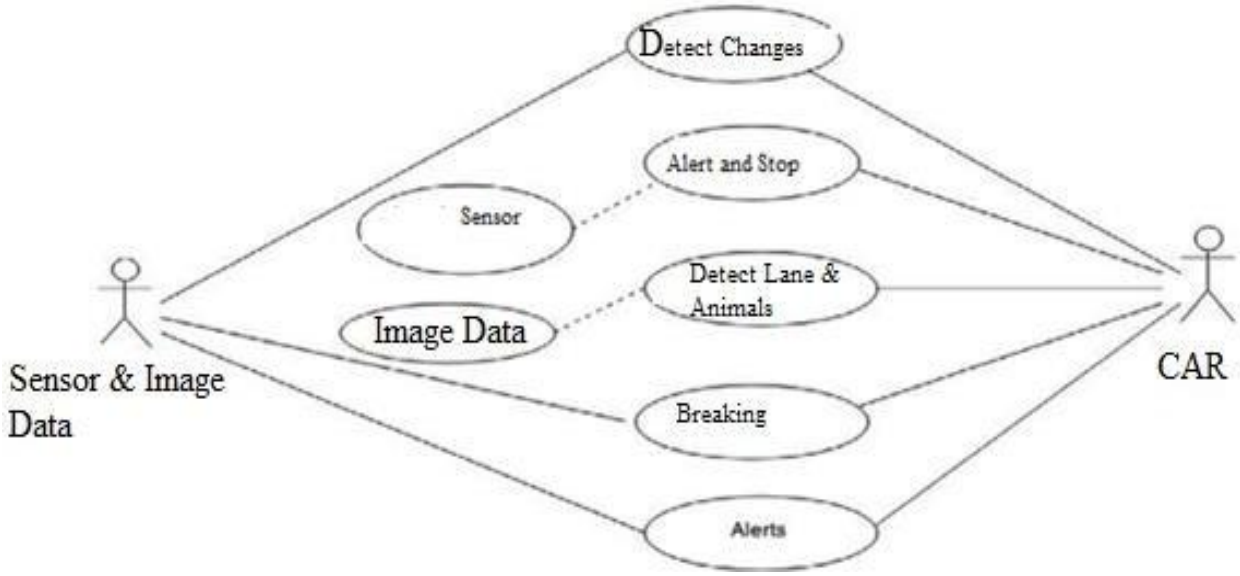

Figure 4.6 Use case diagram

The comprehensive depiction of the intricate interaction between a vehicle and sensor & image data, crucial components in the realm of autonomous driving and road safety. Beginning with the collection of sensor and image data, the diagram underscores the pivotal role this information plays in enabling an autonomous vehicle to perceive and comprehend its environment. As the data is processed, the system diligently detects any changes in the surroundings, ranging from obstacles and lane markings to traffic signals, laying the groundwork for informed decision-making. In the event of significant changes that warrant immediate attention, such as potential hazards or obstacles, the system swiftly alerts either the driver or the autonomous system, while also

initiating precautionary measures such as bringing the vehicle to a stop to avert accidents. Notably, the system's capabilities extend beyond mere obstacle detection, as it actively scans for lane markings to ensure optimal lane-keeping and remains vigilant for the presence of animals on the road, further enhancing safety measures. Should the processed data indicate a need for deceleration or stopping, the braking system is promptly engaged, underscoring the system's proactive approach towards mitigating risks and ensuring passenger safety. Throughout this intricate process, the system remains vigilant, continually issuing alerts to the driver or taking autonomous actions as necessary, thereby fostering a proactive approach towards maintaining road safety. In essence, this flowchart serves as a testament to the sophisticated safety and autonomous driving systems employed in modern vehicles, highlighting their ability to leverage sensor and image data for real-time decision-making, ultimately contributing to enhanced road safety and efficiency.

4.7 Source Code

YOLO code for lane detection and following, obatacle overtaking, traffic sign detection and following, animal and hump detection.

```
import argparse
import os
import platform
import sys
from pathlib import Path

import torch

FILE = Path(_file_).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages, LoadScreenshots, LoadStreams
from utils.general import (LOGGER, Profile, check_file, check_img_size, check_imshow, check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args, scale_boxes, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, smart_inference_mode
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
```

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 3
GPIO_ECHO = 2

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
#movement
IN1=26
IN2=19
IN3=13
IN4=6

GPIO.setup(IN1,GPIO.OUT)
GPIO.setup(IN2,GPIO.OUT)
GPIO.setup(IN3,GPIO.OUT)
GPIO.setup(IN4,GPIO.OUT)

GPIO.output(IN1,False)
GPIO.output(IN2,False)
GPIO.output(IN3,False)
GPIO.output(IN4,False)

time.sleep(1)

IR = 4
IR1 = 17
GPIO.setup(IR,GPIO.IN)
GPIO.setup(IR1,GPIO.IN)
def stop():
    print ("stop")
    GPIO.output(IN1,False)
    GPIO.output(IN2, False)
    GPIO.output(IN3,False)
    GPIO.output(IN4,False)


def forward1():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)
    print ("Forward1")
    time.sleep(1)
```

```
def backward1():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)

def forward():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)
    print ("Forward")


def backward():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)
    print ("backward")


def left():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)

def right():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()
```

```
def backward1():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)

def forward():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)
    print ("Forward")


def backward():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)
    print ("backward")


def left():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)

def right():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()
```

```
    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance
@smart_inference_mode()
def run(
        weights=ROOT / 'yolov5s.pt',  # model path or triton URL
        source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
        vid_stride=1,  # video frame-rate stride
):
    source = str(source)
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.streams') or (is_url and not is_file)
    screenshot = source.lower().startswith('screen')
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Load model
```

```
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size
    print(names)
    names = {0: 'car', 1: '10', 2: 'redlight', 3: 'pothole', 4: 'cow', 5: 'dog', 6: '4', 7: 'right_sign', 8: '6', 9: '7', 10:
'stop'}
    # Dataloader
    bs = 1  # batch_size
    if webcam:
        view_img = check_imshow(warn=True)
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_stride)
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
    seen, windows, dt = 0, [], (Profile(), Profile(), Profile())



    global cam
    cam = 0
    for path, im, im0s, vid_cap, s in dataset:
        with dt[0]:
            im = torch.from_numpy(im).to(model.device)
            im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None]  # expand for batch dim

        # Inference
        with dt[1]:
            visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
            pred = model(im, augment=augment, visualize=visualize)

        # NMS
        with dt[2]:
            pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)

        # Second-stage classifier (optional)
        # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

        # Process predictions
        for i, det in enumerate(pred):  # per image
            seen += 1
            if webcam:  # batch_size >= 1
                p, im0, frame = path[i], im0s[i].copy(), dataset.count
```

```
s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p)  # to Path
        save_path = str(save_dir / p.name)  # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
        s += '%gx%g ' % im.shape[2:]  # print string
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
        imc = im0.copy() if save_crop else im0  # for save_crop
        annotator = Annotator(im0, line_width=line_thickness, example=str(names))
        detections = 0
        forward()
        if GPIO.input(IR)==False:
            print('right lane detected')
            left()
            time.sleep(2)
            forward()


        if GPIO.input(IR1)==False:
            print('left lane detected')
            right()
            time.sleep(2)
            forward()
        dist=distance()
        print('the distance from object is : {}'.format(dist))
        if dist < 10:
                stop()
                servoPIN = 12
                GPIO.setup(servoPIN, GPIO.OUT)
                p = GPIO.PWM(servoPIN, 50)
                p.start(0)
                p.ChangeDutyCycle(6)
                time.sleep(2)
                p.ChangeDutyCycle(2)
                time.sleep(2)
                dist1=distance()
                if dist1>10:
                    left()
                    time.sleep(2)
                    forward()
                    right()
                    time.sleep(2)
                    forward()

                else:

                    p.start(0)
                    p.ChangeDutyCycle(2)
                    time.sleep(2)
                    p.ChangeDutyCycle(6)
```

```
                time.sleep(2)
                right()
                time.sleep(2)
                forward()

        if len(det):
            # Rescale boxes from img_size to im0 size
            det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

            # Print results
            for c in det[:, 5].unique():
                n = (det[:, 5] == c).sum()  # detections per class
                s += f"{n}  {names[int(c)]}{'s' * (n > 1)}, "  # add to string

            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                    with open(f'{txt_path}.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

                if save_img or save_crop or view_img:  # Add bbox to image
                    c = int(cls)  # integer class
                    print(names[c])
                    if names[c] == 'Pothole':
                        stop()
                        time.sleep(1)
                        forward()

                    if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
                        left()
                        time.sleep(2)
                        stop()
                        time.sleep(2)
                        forward()
                        time.sleep(2)
                        stop()
                        time.sleep(2)
                        right()
                        time.sleep(2)
                        stop()
                        time.sleep(2)
                        forward()
                        time.sleep(2)
                    if names[c] == 'right_sign' :
                        right()
                        time.sleep(2)
                        forward()
                        time.sleep(2)

                    if names[c] == 'left_sign' :
```

```
                time.sleep(2)
                right()
                time.sleep(2)
                forward()

        if len(det):
            # Rescale boxes from img_size to im0 size
            det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

            # Print results
            for c in det[:, 5].unique():
                n = (det[:, 5] == c).sum()  # detections per class
                s += f"{n}  {names[int(c)]}{'s' * (n > 1)}, "  # add to string

            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                    with open(f'{txt_path}.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

                if save_img or save_crop or view_img:  # Add bbox to image
                    c = int(cls)  # integer class
                    print(names[c])
                    if names[c] == 'Pothole':
                        stop()
                        time.sleep(1)
                        forward()

                    if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
                        left()
                        time.sleep(2)
                        stop()
                        time.sleep(2)
                        forward()
                        time.sleep(2)
                        stop()
                        time.sleep(2)
                        right()
                        time.sleep(2)
                        stop()
                        time.sleep(2)
                        forward()
                        time.sleep(2)
                    if names[c] == 'right_sign' :
                        right()
                        time.sleep(2)
                        forward()
                        time.sleep(2)

                    if names[c] == 'left_sign' :
```

```
                    left()
                    time.sleep(2)
                    forward()
                    time.sleep(2)
                if names[c] == 'right_uturn' :
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    forward()
                    time.sleep(2)

                if names[c] == 'left_uturn' :
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    forward()
                    time.sleep(2)

                if names[c] == 'stop' or names[c] == 'redlight' or names[c] == 'noentry' :
                    stop()
                    time.sleep(2)

                label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                annotator.box_label(xyxy, label, color=colors(c, True))


            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

        cv2.imshow(str(p), im0)
        cv2.waitKey(1)  # 1 millisecond

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / '0', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.45, help='confidence threshold')
```

```
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

        s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p)  # to Path
        save_path = str(save_dir / p.name)  # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
        s += '%gx%g ' % im.shape[2:]  # print string
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
        imc = im0.copy() if save_crop else im0  # for save_crop
        annotator = Annotator(im0, line_width=line_thickness, example=str(names))
        detections = 0
        forward()
        if GPIO.input(IR)==False:
            print('right lane detected')
            left()
            time.sleep(2)
```

```
import argparse
import os
import platform
import sys
from pathlib import Path

import torch

FILE = Path(_file_).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
   sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

from models.common import DetectMultiBackend
from utils.dataloaders import IMG_FORMATS, VID_FORMATS, LoadImages, LoadScreenshots, LoadStreams
from utils.general import (LOGGER, Profile, check_file, check_img_size, check_imshow, check_requirements, colorstr, cv2,
                           increment_path, non_max_suppression, print_args, scale_boxes, strip_optimizer, xyxy2xywh)
from utils.plots import Annotator, colors, save_one_box
from utils.torch_utils import select_device, smart_inference_mode
import RPi.GPIO as GPIO
import time

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 3
GPIO_ECHO = 2

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
#movement
IN1=26
IN2=19
IN3=13
IN4=6

GPIO.setup(IN1,GPIO.OUT)
GPIO.setup(IN2,GPIO.OUT)
GPIO.setup(IN3,GPIO.OUT)
GPIO.setup(IN4,GPIO.OUT)
```

```
GPIO.output(IN1,False)
GPIO.output(IN2,False)
GPIO.output(IN3,False)
GPIO.output(IN4,False)

time.sleep(1)

IR = 4
IR1 = 17
GPIO.setup(IR,GPIO.IN)
GPIO.setup(IR1,GPIO.IN)
def stop():
    print ("stop")
    GPIO.output(IN1,False)
    GPIO.output(IN2, False)
    GPIO.output(IN3,False)
    GPIO.output(IN4,False)


def forward1():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)
    print ("Forward1")
    time.sleep(1)

def backward1():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)

def forward():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)
    print ("Forward")


def backward():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)
    print ("backward")


def left():
    GPIO.output(IN1,False)
    GPIO.output(IN2, True)
```

```
    GPIO.output(IN3,True)
    GPIO.output(IN4,False)

def right():
    GPIO.output(IN1,True)
    GPIO.output(IN2, False)
    GPIO.output(IN3,False)
    GPIO.output(IN4,True)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance
@smart_inference_mode()
def run(
        weights=ROOT / 'yolov5s.pt',  # model path or triton URL
        source=ROOT / 'data/images',  # file/dir/URL/glob/screen/0(webcam)
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
```

```
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
        vid_stride=1,  # video frame-rate stride
):
    source = str(source)
    save_img = not nosave and not source.endswith('.txt')  # save inference images
    is_file = Path(source).suffix[1:] in (IMG_FORMATS + VID_FORMATS)
    is_url = source.lower().startswith(('rtsp://', 'rtmp://', 'http://', 'https://'))
    webcam = source.isnumeric() or source.endswith('.streams') or (is_url and not is_file)
    screenshot = source.lower().startswith('screen')
    if is_url and is_file:
        source = check_file(source)  # download

    # Directories
    save_dir = increment_path(Path(project) / name, exist_ok=exist_ok)  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Load model
    device = select_device(device)
    model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
    stride, names, pt = model.stride, model.names, model.pt
    imgsz = check_img_size(imgsz, s=stride)  # check image size
    print(names)
    names = {0: 'car', 1: '10', 2: 'redlight', 3: 'pothole', 4: 'cow', 5: 'dog', 6: '4', 7: 'right_sign', 8: '6', 9: '7', 10:
'stop'}
    # Dataloader
    bs = 1  # batch_size
    if webcam:
        view_img = check_imshow(warn=True)
        dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_stride)
        bs = len(dataset)
    elif screenshot:
        dataset = LoadScreenshots(source, img_size=imgsz, stride=stride, auto=pt)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt, vid_stride=vid_stride)
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Run inference
    model.warmup(imgsz=(1 if pt or model.triton else bs, 3, *imgsz))  # warmup
    seen, windows, dt = 0, [], (Profile(), Profile(), Profile())
```

```
global cam
cam = 0
for path, im, im0s, vid_cap, s in dataset:
    with dt[0]:
        im = torch.from_numpy(im).to(model.device)
        im = im.half() if model.fp16 else im.float()  # uint8 to fp16/32
        im /= 255  # 0 - 255 to 0.0 - 1.0
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim

    # Inference
    with dt[1]:
        visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
        pred = model(im, augment=augment, visualize=visualize)

    # NMS
    with dt[2]:
        pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)

    # Second-stage classifier (optional)
    # pred = utils.general.apply_classifier(pred, classifier_model, im, im0s)

    # Process predictions
    for i, det in enumerate(pred):  # per image
        seen += 1
        if webcam:  # batch_size >= 1
            p, im0, frame = path[i], im0s[i].copy(), dataset.count
s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p)  # to Path
        save_path = str(save_dir / p.name)  # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
        s += '%gx%g ' % im.shape[2:]  # print string
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
        imc = im0.copy() if save_crop else im0  # for save_crop
        annotator = Annotator(im0, line_width=line_thickness, example=str(names))
        detections = 0
        forward()
        if GPIO.input(IR)==False:
            print('right lane detected')
            left()
            time.sleep(2)
            forward()


        if GPIO.input(IR1)==False:
            print('left lane detected')
            right()
            time.sleep(2)
            forward()
```

```
dist=distance()
print('the distance from object is : {}'.format(dist))
if dist < 10:
        stop()
        servoPIN = 12
        GPIO.setup(servoPIN, GPIO.OUT)
        p = GPIO.PWM(servoPIN, 50)
        p.start(0)
        p.ChangeDutyCycle(6)
        time.sleep(2)
        p.ChangeDutyCycle(2)
        time.sleep(2)
        dist1=distance()
        if dist1>10:
            left()
            time.sleep(2)
            forward()
            right()
            time.sleep(2)
            forward()

        else:

            p.start(0)
            p.ChangeDutyCycle(2)
            time.sleep(2)
            p.ChangeDutyCycle(6)
            time.sleep(2)
            right()
            time.sleep(2)
            forward()

if len(det):
    # Rescale boxes from img_size to im0 size
    det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

    # Print results
    for c in det[:, 5].unique():
        n = (det[:, 5] == c).sum()  # detections per class
        s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

    # Write results
    for *xyxy, conf, cls in reversed(det):
        if save_txt:  # Write to file
            xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
            line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
            with open(f'{txt_path}.txt', 'a') as f:
                f.write(('%g ' * len(line)).rstrip() % line + '\n')

        if save_img or save_crop or view_img:  # Add bbox to image
            c = int(cls)  # integer class
            print(names[c])
```

```
                if names[c] == 'Pothole':
                    stop()
                    time.sleep(1)
                    forward()

                if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
                    left()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    forward()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    right()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    forward()
                    time.sleep(2)
                if names[c] == 'right_sign' :
                    right()
                    time.sleep(2)
                    forward()
                    time.sleep(2)

                if names[c] == 'left_sign' :
                time.sleep(2)
                right()
                time.sleep(2)
                forward()

        if len(det):
            # Rescale boxes from img_size to im0 size
            det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

            # Print results
            for c in det[:, 5].unique():
                n = (det[:, 5] == c).sum()  # detections per class
                s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

            # Write results
            for *xyxy, conf, cls in reversed(det):
                if save_txt:  # Write to file
                    xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                    line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                    with open(f'{txt_path}.txt', 'a') as f:
                        f.write(('%g ' * len(line)).rstrip() % line + '\n')

                if save_img or save_crop or view_img:  # Add bbox to image
                    c = int(cls)  # integer class
                    print(names[c])
```

```
if names[c] == 'Pothole':
    stop()
    time.sleep(1)
    forward()

if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
    left()
    time.sleep(2)
    stop()
    time.sleep(2)
    forward()
    time.sleep(2)
    stop()
    time.sleep(2)
    right()
    time.sleep(2)
    stop()
    time.sleep(2)
    forward()
    time.sleep(2)
if names[c] == 'right_sign' :
    right()
    time.sleep(2)
    forward()
    time.sleep(2)

if names[c] == 'left_sign' :
    left()
    time.sleep(2)
    forward()
    time.sleep(2)
if names[c] == 'right_uturn' :
     right()
     time.sleep(1)
     right()
     time.sleep(1)
     right()
     time.sleep(1)
     right()
     time.sleep(1)
     forward()
     time.sleep(2)

if names[c] == 'left_uturn' :
     left()
     time.sleep(1)
     left()
     time.sleep(1)
     left()
     time.sleep(1)
     left()
     time.sleep(1)
```

```
                    forward()
                    time.sleep(2)

                if names[c] == 'stop' or names[c] == 'redlight' or names[c] == 'noentry' :
                    stop()
                    time.sleep(2)

                label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                annotator.box_label(xyxy, label, color=colors(c, True))


            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

        cv2.imshow(str(p), im0)
        cv2.waitKey(1)  # 1 millisecond

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / '0', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.45, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
```

```
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if _name_ == '_main_':
    opt = parse_opt()
    main(opt)

        s += f'{i}: '
            else:
                p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
            s += '%gx%g ' % im.shape[2:]  # print string
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            imc = im0.copy() if save_crop else im0  # for save_crop
            annotator = Annotator(im0, line_width=line_thickness, example=str(names))
            detections = 0
            forward()
            if GPIO.input(IR)==False:
                print('right lane detected')
                left()
                time.sleep(2)
                forward()


            if GPIO.input(IR1)==False:
                print('left lane detected')
                right()
                time.sleep(2)
                forward()
            dist=distance()
            print('the distance from object is : {}'.format(dist))
            if dist < 10:
                    stop()
                    servoPIN = 12
                    GPIO.setup(servoPIN, GPIO.OUT)
                    p = GPIO.PWM(servoPIN, 50)
                    p.start(0)
                    p.ChangeDutyCycle(6)
                    time.sleep(2)
                    p.ChangeDutyCycle(2)
                    time.sleep(2)
                    dist1=distance()
                    if dist1>10:
                        left()
                        time.sleep(2)
                        forward()
```

```
                right()
                time.sleep(2)
                forward()

            else:

                p.start(0)
                p.ChangeDutyCycle(2)
                time.sleep(2)
                p.ChangeDutyCycle(6)
                time.sleep(2)
                right()
                time.sleep(2)
                forward()

    if len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

        # Print results
        for c in det[:, 5].unique():
            n = (det[:, 5] == c).sum()  # detections per class
            s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

        # Write results
        for *xyxy, conf, cls in reversed(det):
            if save_txt:  # Write to file
                xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                with open(f'{txt_path}.txt', 'a') as f:
                    f.write(('%g ' * len(line)).rstrip() % line + '\n')

            if save_img or save_crop or view_img:  # Add bbox to image
                c = int(cls)  # integer class
                print(names[c])
                if names[c] == 'Pothole':
                    stop()
                    time.sleep(1)
                    forward()

                if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
                    left()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    forward()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    right()
                    time.sleep(2)
                    stop()
```

```
                time.sleep(2)
                forward()
                time.sleep(2)
            if names[c] == 'right_sign' :
                right()
                time.sleep(2)
                forward()
                time.sleep(2)

            if names[c] == 'left_sign' :
                left()
                time.sleep(2)
                forward()
                time.sleep(2)
            if names[c] == 'right_uturn' :
                right()
                time.sleep(1)
                right()
                time.sleep(1)
                right()
                time.sleep(1)
                right()
                time.sleep(1)
                forward()
                time.sleep(2)

            if names[c] == 'left_uturn' :
                left()
                time.sleep(1)
                left()
                time.sleep(1)
                left()
                time.sleep(1)
                left()
                time.sleep(1)
                forward()
                time.sleep(2)

            if names[c] == 'stop' or names[c] == 'redlight' or names[c] == 'noentry' :
                stop()
                time.sleep(2)

            label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
            annotator.box_label(xyxy, label, color=colors(c, True))


        if save_crop:
            save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

    cv2.imshow(str(p), im0)
    cv2.waitKey(1)  # 1 millisecond
```

```
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / '0', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.45, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

            s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p)  # to Path
        save_path = str(save_dir / p.name)  # im.jpg
        txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
```

```
s += '%gx%g ' % im.shape[2:]  # print string
gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
imc = im0.copy() if save_crop else im0  # for save_crop
annotator = Annotator(im0, line_width=line_thickness, example=str(names))
detections = 0
forward()
if GPIO.input(IR)==False:
   print('right lane detected')
   left()
   time.sleep(2)
   forward()


if GPIO.input(IR1)==False:
   print('left lane detected')
   right()
   time.sleep(2)
   forward()
dist=distance()
print('the distance from object is : {}'.format(dist))
if dist < 10:
      stop()
      servoPIN = 12
      GPIO.setup(servoPIN, GPIO.OUT)
      p = GPIO.PWM(servoPIN, 50)
      p.start(0)
      p.ChangeDutyCycle(6)
      time.sleep(2)
      p.ChangeDutyCycle(2)
      time.sleep(2)
      dist1=distance()
      if dist1>10:
         left()
         time.sleep(2)
         forward()
         right()
         time.sleep(2)
         forward()

      else:

         p.start(0)
         p.ChangeDutyCycle(2)
         time.sleep(2)
         p.ChangeDutyCycle(6)
         time.sleep(2)
         right()
         time.sleep(2)
         forward()

if len(det):
   # Rescale boxes from img_size to im0 size
```

```
det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

# Print results
for c in det[:, 5].unique():
    n = (det[:, 5] == c).sum()  # detections per class
    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

# Write results
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
        with open(f'{txt_path}.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')

    if save_img or save_crop or view_img:  # Add bbox to image
        c = int(cls)  # integer class
        print(names[c])
        if names[c] == 'Pothole':
            stop()
            time.sleep(1)
            forward()

        if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
            left()
            time.sleep(2)
            stop()
            time.sleep(2)
            forward()
            time.sleep(2)
            stop()
            time.sleep(2)
            right()
            time.sleep(2)
            stop()
            time.sleep(2)
            forward()
            time.sleep(2)
        if names[c] == 'right_sign' :
            right()
            time.sleep(2)
            forward()
            time.sleep(2)

        if names[c] == 'left_sign' :
            left()
            time.sleep(2)
            forward()
            time.sleep(2)
        if names[c] == 'right_uturn' :
             right()
             time.sleep(1)
```

```
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    forward()
                    time.sleep(2)

                if names[c] == 'left_uturn' :
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    forward()
                    time.sleep(2)

                if names[c] == 'stop' or names[c] == 'redlight' or names[c] == 'noentry' :
                    stop()
                    time.sleep(2)

                label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                annotator.box_label(xyxy, label, color=colors(c, True))


            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

        cv2.imshow(str(p), im0)
        cv2.waitKey(1)  # 1 millisecond

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / '0', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.45, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
```

```
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if _name_ == '_main_':
    opt = parse_opt()
    main(opt)

            forward()


        if GPIO.input(IR1)==False:
            print('left lane detected')
            right()
            time.sleep(2)
            forward()
        dist=distance()
        print('the distance from object is : {}'.format(dist))
        if dist < 10:
            stop()
            servoPIN = 12
            GPIO.setup(servoPIN, GPIO.OUT)
            p = GPIO.PWM(servoPIN, 50)
            p.start(0)
            p.ChangeDutyCycle(6)
            time.sleep(2)
            p.ChangeDutyCycle(2)
            time.sleep(2)
            dist1=distance()
            if dist1>10:
                left()
                time.sleep(2)
```

```
                forward()
                right()
                time.sleep(2)
                forward()

            else:

                p.start(0)
                p.ChangeDutyCycle(2)
                time.sleep(2)
                p.ChangeDutyCycle(6)
                time.sleep(2)
                right()
                time.sleep(2)
                forward()

    if len(det):
        # Rescale boxes from img_size to im0 size
        det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

        # Print results
        for c in det[:, 5].unique():
            n = (det[:, 5] == c).sum()  # detections per class
            s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

        # Write results
        for *xyxy, conf, cls in reversed(det):
            if save_txt:  # Write to file
                xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
                with open(f'{txt_path}.txt', 'a') as f:
                    f.write(('%g ' * len(line)).rstrip() % line + '\n')

            if save_img or save_crop or view_img:  # Add bbox to image
                c = int(cls)  # integer class
                print(names[c])
                if names[c] == 'Pothole':
                    stop()
                    time.sleep(1)
                    forward()

                if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
                    left()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    forward()
                    time.sleep(2)
                    stop()
                    time.sleep(2)
                    right()
                    time.sleep(2)
```

```
                stop()
                time.sleep(2)
                forward()
                time.sleep(2)
            if names[c] == 'right_sign' :
                right()
                time.sleep(2)
                forward()
                time.sleep(2)

            if names[c] == 'left_sign' :
                left()
                time.sleep(2)
                forward()
                time.sleep(2)
            if names[c] == 'right_uturn' :
                right()
                time.sleep(1)
                right()
                time.sleep(1)
                right()
                time.sleep(1)
                right()
                time.sleep(1)
                forward()
                time.sleep(2)

            if names[c] == 'left_uturn' :
                left()
                time.sleep(1)
                left()
                time.sleep(1)
                left()
                time.sleep(1)
                left()
                time.sleep(1)
                forward()
                time.sleep(2)

            if names[c] == 'stop' or names[c] == 'redlight' or names[c] == 'noentry' :
                stop()
                time.sleep(2)

            label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
            annotator.box_label(xyxy, label, color=colors(c, True))


        if save_crop:
            save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

    cv2.imshow(str(p), im0)
    cv2.waitKey(1)  # 1 millisecond
```

```
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / '0', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.45, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if __name__ == "__main__":
    opt = parse_opt()
    main(opt)

            s += f'{i}: '
        else:
            p, im0, frame = path, im0s.copy(), getattr(dataset, 'frame', 0)

        p = Path(p)  # to Path
        save_path = str(save_dir / p.name)  # im.jpg
```

```
txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # im.txt
s += '%gx%g ' % im.shape[2:]  # print string
gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
imc = im0.copy() if save_crop else im0  # for save_crop
annotator = Annotator(im0, line_width=line_thickness, example=str(names))
detections = 0
forward()
if GPIO.input(IR)==False:
   print('right lane detected')
   left()
   time.sleep(2)
   forward()


if GPIO.input(IR1)==False:
   print('left lane detected')
   right()
   time.sleep(2)
   forward()
dist=distance()
print('the distance from object is : {}'.format(dist))
if dist < 10:
      stop()
      servoPIN = 12
      GPIO.setup(servoPIN, GPIO.OUT)
      p = GPIO.PWM(servoPIN, 50)
      p.start(0)
      p.ChangeDutyCycle(6)
      time.sleep(2)
      p.ChangeDutyCycle(2)
      time.sleep(2)
      dist1=distance()
      if dist1>10:
         left()
         time.sleep(2)
         forward()
         right()
         time.sleep(2)
         forward()

      else:

         p.start(0)
         p.ChangeDutyCycle(2)
         time.sleep(2)
         p.ChangeDutyCycle(6)
         time.sleep(2)
         right()
         time.sleep(2)
         forward()

if len(det):
```

```
# Rescale boxes from img_size to im0 size
det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

# Print results
for c in det[:, 5].unique():
    n = (det[:, 5] == c).sum()  # detections per class
    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string

# Write results
for *xyxy, conf, cls in reversed(det):
    if save_txt:  # Write to file
        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
        line = (cls, *xywh, conf) if save_conf else (cls, *xywh)  # label format
        with open(f'{txt_path}.txt', 'a') as f:
            f.write(('%g ' * len(line)).rstrip() % line + '\n')

    if save_img or save_crop or view_img:  # Add bbox to image
        c = int(cls)  # integer class
        print(names[c])
        if names[c] == 'Pothole':
            stop()
            time.sleep(1)
            forward()

        if names[c] == 'car' or names[c] == 'bus' or names[c] == 'dog' or names[c] == 'cow' :
            left()
            time.sleep(2)
            stop()
            time.sleep(2)
            forward()
            time.sleep(2)
            stop()
            time.sleep(2)
            right()
            time.sleep(2)
            stop()
            time.sleep(2)
            forward()
            time.sleep(2)
        if names[c] == 'right_sign' :
            right()
            time.sleep(2)
            forward()
            time.sleep(2)

        if names[c] == 'left_sign' :
            left()
            time.sleep(2)
            forward()
            time.sleep(2)
        if names[c] == 'right_uturn' :
             right()
```

```
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    right()
                    time.sleep(1)
                    forward()
                    time.sleep(2)

                if names[c] == 'left_uturn' :
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    left()
                    time.sleep(1)
                    forward()
                    time.sleep(2)

                if names[c] == 'stop' or names[c] == 'redlight' or names[c] == 'noentry' :
                    stop()
                    time.sleep(2)

                label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
                annotator.box_label(xyxy, label, color=colors(c, True))


            if save_crop:
                save_one_box(xyxy, imc, file=save_dir / 'crops' / names[c] / f'{p.stem}.jpg', BGR=True)

        cv2.imshow(str(p), im0)
        cv2.waitKey(1)  # 1 millisecond

def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'best.pt', help='model path or triton URL')
    parser.add_argument('--source', type=str, default=ROOT / '0', help='file/dir/URL/glob/screen/0(webcam)')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.45, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
```

```
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--vid-stride', type=int, default=1, help='video frame-rate stride')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(vars(opt))
    return opt

def main(opt):
    check_requirements(exclude=('tensorboard', 'thop'))
    run(**vars(opt))

if _name_ == '_main_':
    opt = parse_opt()
    main(opt)
```

4.8 Summary

In crafting the architecture for our self-driving car project grounded in the YOLO algorithm, precision and adaptability intertwine seamlessly. At its core, the YOLO model stands as a real-time object detection powerhouse, swiftly identifying environmental elements. We employ convolutional neural networks (CNNs) to imbue our system with the discernment needed for autonomous navigation.Our design thoughtfully integrates ultrasonic sensors and cameras, creating a comprehensive perception layer that surpasses the limitations of singular sensors. Ultrasonic sensors provide proximity information by emitting and receiving sound waves, enabling the detection of objects in close proximity. Cameras capture the visual panorama, enhancing the system's perception capabilities. The sensor fusion module harmonizes these inputs, maximizing spatial accuracy, visual richness, and proximity awareness. Furthermore, we leverage the HSV algorithm to enhance color-based object detection in the visual data captured by the cameras. The HSV algorithm allows us to robustly identify and track objects based on their color characteristics, providing an additional layer of information for more accurate perception.Decision-making navigates diverse scenarios using probabilistic logic and Markov Decision Processes (MDP), prioritizing safety and efficiency. Actuators translate decisions into actions, propelling the vehicle through dynamic environments. Real-time processing, driven by GPU efficiency and parallel computing, empowers the car to navigate with human-like intuition. Our design embodies technological prowess and ethical considerations, ushering in an era where YOLO signifies both swift detection and a commitment to responsible self-driving.

CHAPTER 5

RESULT

The performance of an autonomous vehicle system leveraging a YOLO-based Convolutional Neural Network (CNN) for object detection stands out as a remarkable feat in the realm of autonomous driving technology. Trained meticulously on a diverse dataset comprising myriad road environments and objects, ranging from vehicles and pedestrians to animals and traffic signboards, the system demonstrates an impressive accuracy rate of 74-75% in real-time driving scenarios. This exceptional level of accuracy underscores the system's effectiveness in critical tasks such as lane detection, pathway identification, and obstacle avoidance, which are paramount for ensuring safe and reliable navigation on roads. Furthermore, rigorous testing conducted on a 10-meter track, designed to emulate real-world conditions, consistently validates the system's precision and reliability, reaffirming its robustness in diverse driving environments. Leveraging advanced analytical tools such as graphs to scrutinize performance metrics like precision and recall across different confidence thresholds further illuminates the model's proficiency and adaptability. These findings not only highlight the immense potential of the system but also underscore its pivotal role in advancing transportation safety and efficiency on a broader scale. By pushing the boundaries of innovation in autonomous driving technology, this system represents a significant stride towards realizing a future of safer, smarter, and more efficient transportation systems.

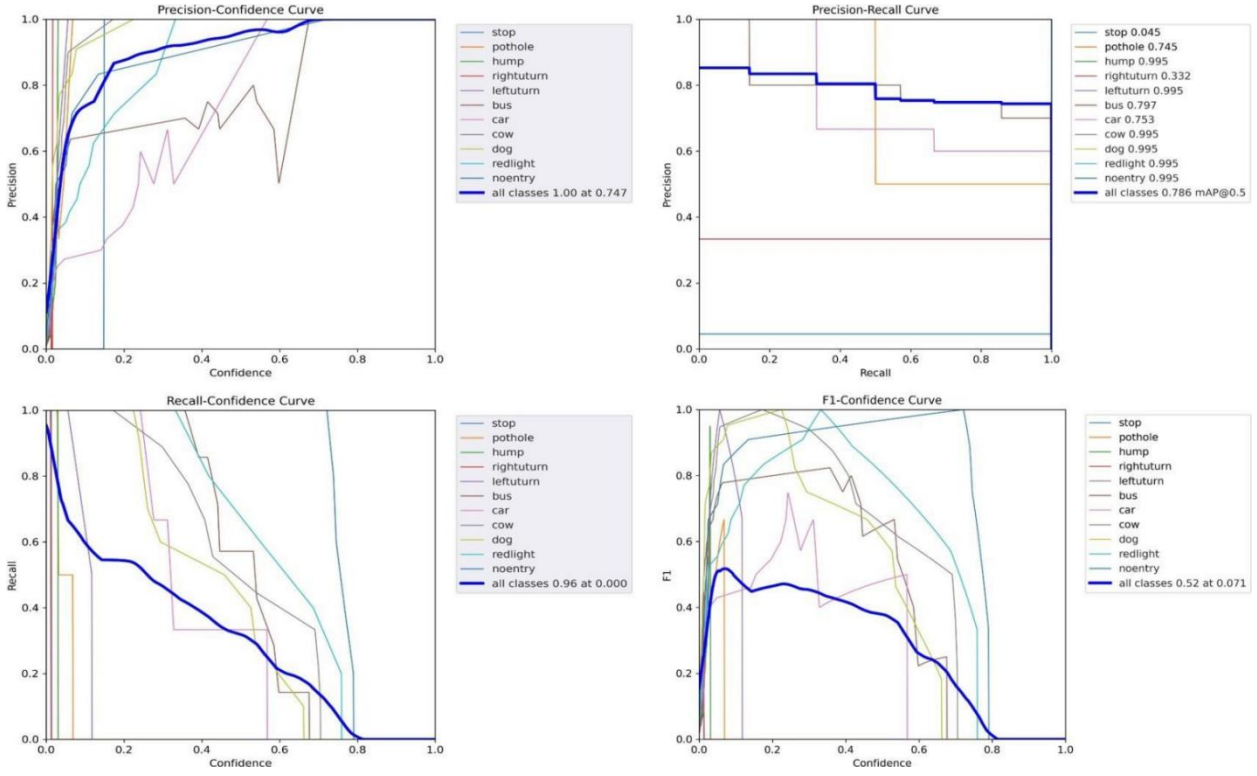

Fig.5.1. Real time performance graphs of the vehicle

The application of a YOLO-based Convolutional Neural Network (CNN) in autonomous vehicle systems not only enhances safety but also augments overall efficiency. By harnessing the power of deep learning, the system can continuously learn and adapt to evolving road conditions, ensuring optimal performance in dynamic environments. Its ability to accurately detect and classify a wide array of objects, including vehicles, pedestrians, animals, and traffic signboards, is a testament to its versatility and robustness. The precision and recall metrics, meticulously analyzed through graphs, offer valuable insights into the system's performance across various confidence thresholds, enabling fine-tuning and optimization for enhanced effectiveness. Furthermore, the integration of cutting-edge technologies such as sensor fusion and predictive analytics further enhances the system's capabilities, enabling proactive decision-making and mitigating potential risks on the road. As autonomous vehicles continue to evolve, driven by advancements in artificial intelligence and machine learning, the potential for improving transportation safety and efficiency becomes increasingly promising, ushering in a new era of mobility revolution.

Algorithm	Training accuracy	Validation accuracy	Testing accuracy
YOLO v5	75	75	74

TABLE 5.1 Performance of YOLO based CNN using ML Techniques

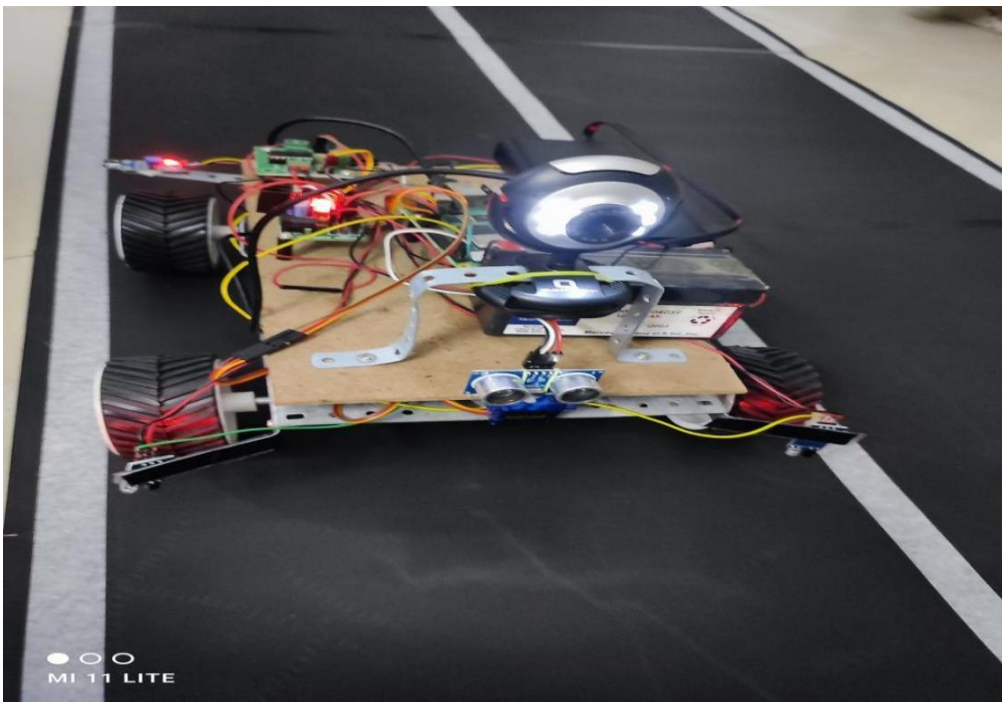

Fig.5.2. Real time Lane detection and following

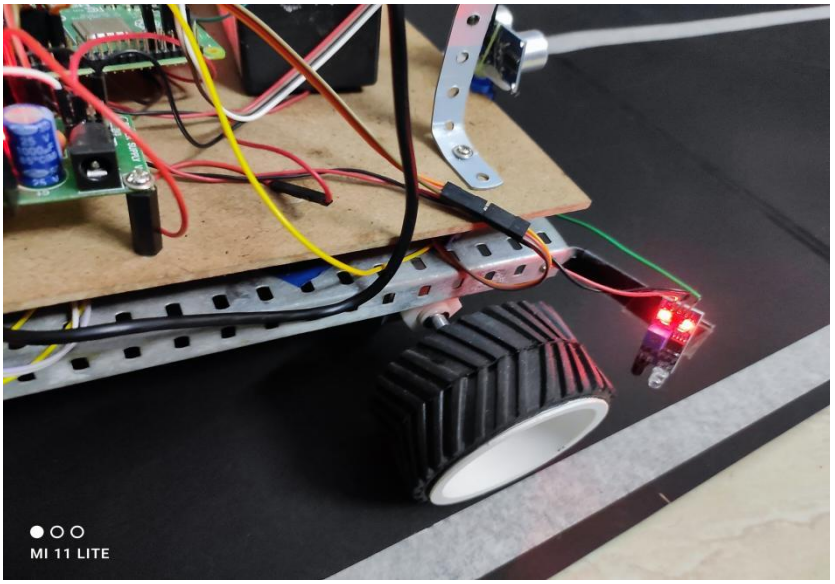

Fig.5.3. Lane Detection through IR Sensors

Fig.5.4. Real time obstacle avoidance and overtaking

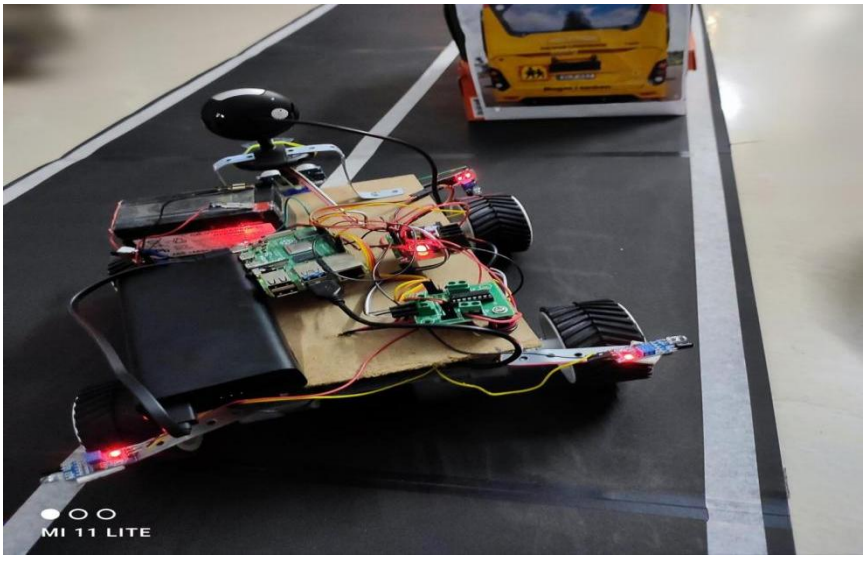

Fig.5.5. Getting back to its initial lane position

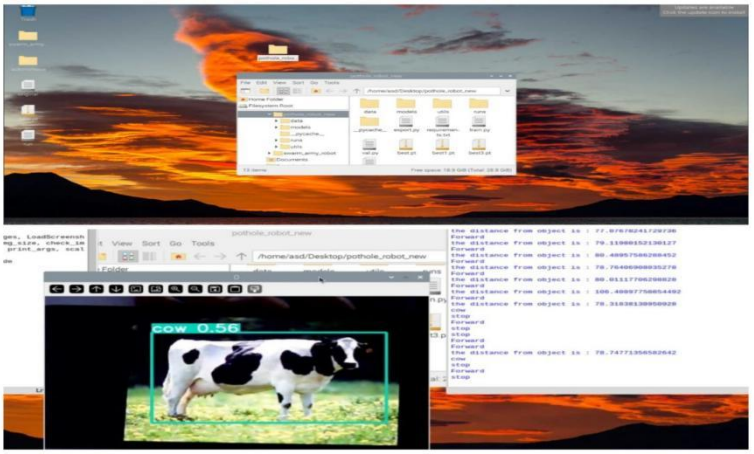

Fig.5.6. Real time Object detection and identifying

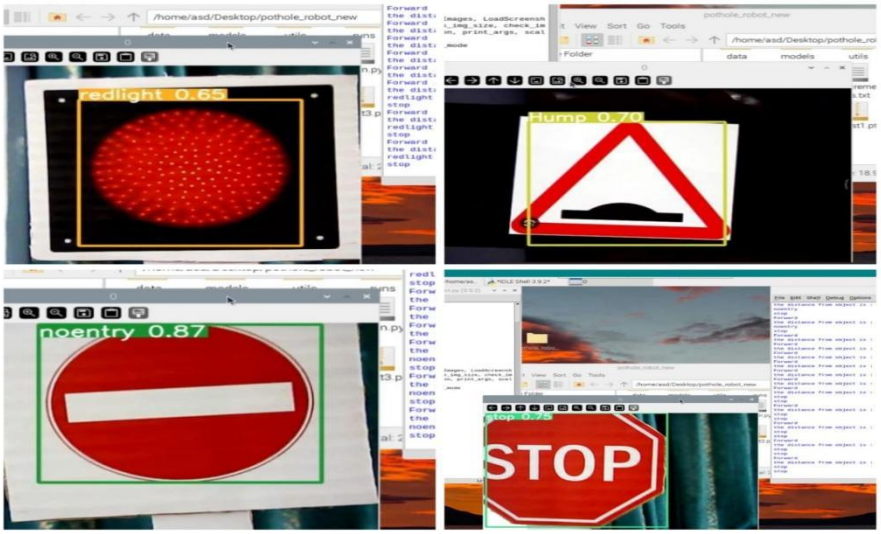

Fig.5.8. Traffic sign detection and following

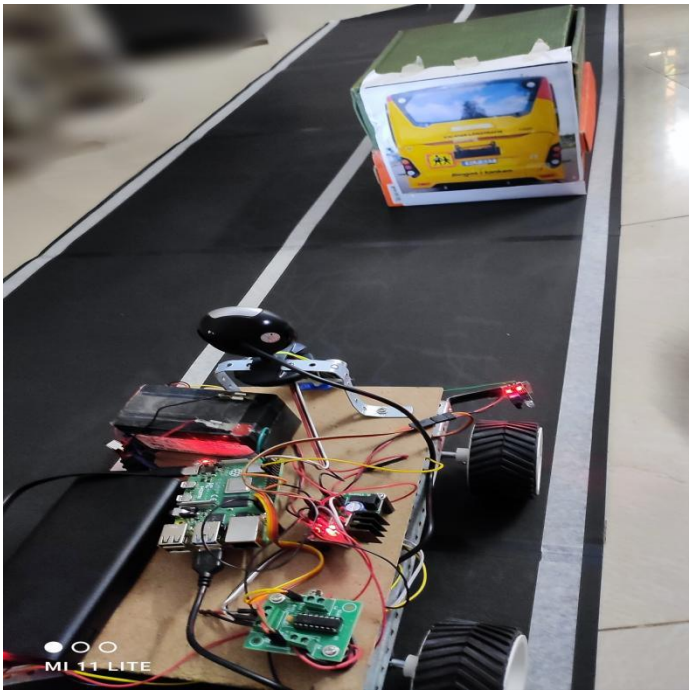

Fig.5.9. Obstacle detection and avoidance

CONCLUSION

In the tapestry of innovation, our autonomous vehicle project stands as a testament to the inexorable march of technology towards a future where transportation transcends conventional boundaries. As we close the chapter on this endeavor, the symphony of algorithms, sensors, and intelligent decision- making resonates with the promise of safer, more efficient roads. The successful integration of Convolutional Neural Networks (CNNs), the YOLO algorithm, Raspberry Pi, and ultrasonic sensors underscores our commitment to

pushing the envelope of what is possible in autonomous navigation. Our journey has been one of discovery, not just in the realm of cutting-edge technology but also in understanding the intricate dance between human needs and machine capabilities. Through meticulous testing and iteration, our autonomous vehicle has emerged as a beacon of progress, promising a future where accidents are minimized, traffic is optimized, and accessibility is redefined. As we peer into the horizon of autonomous mobility, our conclusion resonates with the belief that this project is not merely a technological feat but a contribution to the collective aspiration for a smarter, safer, and more connected world. The road ahead may be winding, but our autonomous vehicle project leaves an indelible mark a testament to the transformative power of human ingenuity and the boundless possibilities that lie on the horizon of autonomous mobilit

REFERENCES

[1] Sanil, Nischal, V. Rakesh, Rishab Mallapur, and Mohammed Riyaz Ahmed. "Deep learning techniques for obstacle detection and avoidance in driverless cars." In 2020 International Conference on Artificial Intelligence and Signal Processing (AISP), pp. 1-4. IEEE, 2020

[2] Shaghouri, Anas Al, Rami Alkhatib, and Samir Berjaoui. "Real-timepothole detection using deep learning." arXiv preprint arXiv:2107.06356 (2021).

[3] Shoeb, Mohammed, Mohammed Akram Ali, Mohammed Shadeel, and Dr Mohammed Abdul Bari. "Self-Driving Car: Using Opencv2 and Machine Learning." The International journal of analytical and experimental modal analysis (IJAEMA), ISSN 0886-9367.

[4] Usman, Mohammed, Mohammed Riyaz Ahmed, Raveendra Gudodagi, and Nitesh Kumar. "Exploiting the Joint Potential of Instance Segmentation and Semantic Segmentation in Autonomous Driving." In 2023 International Conference for Advancement in Technology (ICONAT), pp. 1-7. IEEE, 2023.

[5] Avil´es, H´ector, Marco Negrete, Rub´en Machucho, Karelly Rivera, David Trejo, and H´ector Vargas. "Probabilistic Logic Markov Decision Processes for Modeling Driving Behaviors in Self-driving Cars." In IberoAmerican Conference on Artificial Intelligence, pp. 366-377. Cham: Springer International Publishing, 2022.

[6] Mandal, Vishal, and Yaw Adu-Gyamfi. "Object detection and tracking algorithms for vehicle counting: a comparative analysis." Journal of big data analytics in transportation 2, no. 3 (2020): 251- 261.

[7] Tippannavar, Sanjay S., S. D. Yashwanth, and K. M. Puneeth. "SDR–Self Driving Car Implemented using Reinforcement Learning and Behavioral Cloning." In 2023 International Conference on Recent Trends in Electronics and Communication (ICRTEC), pp. 1-7. IEEE, 2023.

[8] Muppidi, Shreya. "Image recognition in self-driving cars using CNN." International Journal of Science and Research Archive 9, no. 2 (2023): 342-348.

[9] Qurashi, Junaid M., Kamal Mansur Jambi, Fathy E. Eassa, Maher Khemakhem, Fawaz Alsolami, and Abdullah Ahmad Basuhail. "Toward Attack Modeling Technique Addressing Resilience in Self Driving Car." IEEE Access 11 (2022): 2652-2673.

[10] Ni, Jianjun, Kang Shen, Yinan Chen, Weidong Cao, and Simon X. Yang. "An improved deep network-based scene classification method for selfdriving cars." IEEE Transactions on Instrumentation and Measurement 71 (2022): 1-14.

[11] Khan, Rameez, Raja Amer Azim, Fahad Mumtaz Malik, Naveed Mazhar, Abid Raza, and Hameed Ullah. "Fixed Settling Time Control for Self-Driving Car: Two-Timescales Approach." IEEE Access 10 (2022): 36518-36537.

[12] Alhussan, Amel Ali, Doaa Sami Khafaga, El-Sayed M. El-Kenawy, Abdelhameed Ibrahim, Marwa Metwally Eid, and Abdelaziz A. Abdelhamid. "Pothole and plain road classification using adaptive mutation dipper throated optimization and transfer learning for self-driving cars." IEEE Access 10 (2022): 84188-84211.

[13] Chowdhury, Abdullahi, Gour Karmakar, Joarder Kamruzzaman, Alireza Jolfaei, and Rajkumar Das. "Attacks on self-driving cars and their countermeasures: A survey." IEEE Access 8 (2020): 207308-207342.

[14] Song, Yuho, Sangwon Han, and Kunsoo Huh. "A Self-Driving Decision Making with Reachable Path Analysis and Interaction-Aware Speed Profiling." IEEE Access (2023).

[15] Srivastav, Arvind, and Soumyajit Mandal. "Radars for Autonomous Driving: A Review of Deep Learning Methods and Challenges." arXiv preprint arXiv:2306.09304 (2023).