A project report on

Parking Space Optimization and Management

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics

By

ANUJ SINGH CHAUHAN (20BRS1098)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April,2024

Parking Space Optimization and Management

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics

By

ANUJ SINGH CHAUHAN (20BRS1098)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April,2024

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled “Parking Space Optimization and Management” submitted by me, for the award of the degree of Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics, Vellore Institute of Technology, Chennai is a record of bonafide work carried out by me under the supervision of Dr. V Arunkumar.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: **Signature of the Candidate**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report This is to certify that the report entitled **"Parking Space Optimization and Management"** is prepared and submitted by **Anuj Singh Chauhan (20BRS1098)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Robotics** programme is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr. V Arunkumar

Date:

Signature of the Internal Examiner	Signature of the External Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department,

B.Tech. CSE with Specialization in Artificial Intelligence and Robotics

Name: Dr. Harini S

Date:

(Seal of SCOPE)

ABSTRACT

With the increasing challenges of urbanization, the demand for efficient parking solutions has become very necessary. This project introduces an Automated Parking System designed to optimize parking space utilization in response to the evolving needs of modern urban environments.

The project tackles the problem by first recording the number plate of the incoming vehicles, and also measuring the size of the vehicle, this allows the system to appropriately allot one of the two types, small or large, parking space to the car. The system also keeps a track of the empty parking spaces and manages them as the cars enter and exits the parking lot.

Utilizing technologies such as sensor network, robot vision and image processing, the proposed system streamlines the parking process. Real-time data collection, with the help of ultrasonic sensors under the parking lot, enables the Automated Parking System to assess parking space availability and dynamically allocate spaces. By implementing a measurement system that uses a Stereo Vision camera to determine the dimensions of the vehicle that helps us to allot an appropriate parking space to that vehicle, an automated entry and exit system, using Automatic Number Plate Detection, allows for an easier experience for entry and exit, urban planners and stakeholders can achieve a more sustainable and efficient use of parking resources, contributing to the ongoing discourse on smart urban infrastructure. The research presented here outlines a viable solution to challenges posed by conventional parking systems in the face of rapid urbanization.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. V Arunkumar, Assistant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for his/her constant guidance, continual encouragement, understanding; more than all, he/she taught me patience in my endeavor. My association with him / her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Mechatronics.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Harini S, Head of the Department, B.Tech. CSE with Specialization in Artificial Intelligence & Robotics and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: **Anuj Singh Chauhan**

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ANPR	Automatic number plate detection
APS	Automatic Parking System
IoT	Internet of Things
SPMS	Smart Parking Management System
RFID	Radio-Frequency Identification
MQTT	Message Queuing Telemetry Transport
JDBC	Java Database Connectivity
SQL	Structured Query Language
OCR	Optical character recognition
IoU	Intersection of Unions

Chapter 1

Introduction

1.1 OVERVIEW

The rapid pace of urbanization has led to numerous challenges, with one of the most prominent being the management of parking spaces in densely populated areas. As cities expand and vehicle ownership rises, the demand for efficient parking solutions has become increasingly critical. Traditional parking methods often result in inefficiencies, traffic congestion, and environmental issues. In response to these challenges, there is a growing interest in developing Automated Parking Systems that can optimize parking space utilization and enhance overall urban mobility.

1.2 BACKGROUND OF THE PROBLEM

The integration of stereo vision and ANPR in an APS represents a response to the pressing challenges faced by urban areas in managing parking resources efficiently. Traditional parking systems often rely on manual methods for ticketing and space allocation, leading to delays, inefficiencies, and frustrated drivers. As urbanization intensifies, there is a need for smarter, more responsive parking solutions that leverage technological advancements to streamline the parking process and enhance the overall urban mobility experience.

The problem of parking inefficiency is grown by the lack of real-time data and adaptive systems in traditional approaches. Without accurate information on available parking spaces and the dimensions of incoming vehicles, it becomes challenging to optimize the allocation of parking spots. Drivers may spend valuable time searching for suitable spaces, contributing to traffic congestion and increased emissions.

In response to these challenges, the integration of stereo vision and ANPR aims to change the way parking facilities operate. By accurately measuring the dimensions of each vehicle through stereo vision and enrolling them seamlessly with ANPR, the

APS can dynamically allocate parking spaces based on real-time demand and ensure an optimal fit for each vehicle. This not only enhances the efficiency of the parking process but also contributes to a reduction in traffic congestion, emissions, and overall environmental impact.

1.3 PROBLEM STATEMENT

This project aims to tackle the problem of inefficiencies of traditional parking methods, specifically the space optimization in order to help with the scarcity of space in densely populated areas.

1.4 CURRENT PARKING CHALLANGES

Urban areas are grappling with limited parking space availability, leading to increased traffic congestion as vehicles circle in search of suitable parking spots. This not only causes frustration for drivers but also contributes to environmental concerns due to heightened carbon emissions from idling vehicles. Moreover, the traditional approach to parking management lacks the flexibility needed to adapt to changing patterns of demand, resulting in underutilized or overcrowded parking facilities.

Chapter 2

Related Research Work

2.1 LITRATURE REVIEW

Parking management has emerged as a crucial facet of urban planning, grappling with challenges stemming from limited parking spaces, escalating traffic congestion, and escalating environmental concerns. The evolution of technology has given rise to intelligent parking systems, with an array of research papers shedding light on innovative solutions. The ensuing literature review synthesizes insights from a diverse selection of studies, encompassing automated systems, novel datasets, web-based frameworks, RFID applications, smart parking management for cities, Raspberry Pi-based solutions, machine learning approaches, and futuristic perspectives on parking. Each of these papers contributes distinct perspectives and technological applications to tackle the complex issues surrounding contemporary parking management.

2.1.1 SMART PARKING USING TRADITIONAL METHODS

In the paper by Elsonbaty and Shams (2020) [14] they introduces a System called Smart Parking Management Systems (SPMS) .This system uses Arduino microcontrollers, Android applications, and Internet of Things (IoT) technologies to optimize the utilization of parking spaces. They used IR sensors for real-time detection of parking space availability and transmitted data wirelessly to servers, SPMS enables users to access parking information conveniently through an Android applications. The application provides an interface for users to check availability, reserve spots, and receive real-time updates on parking status. They also included cost calculation based on entry and exit times enhance user experience and convenience, contributing to improved urban mobility. Furthermore, SPMS aligns with the broader vision of building smart cities by providing real-time data on parking availability and usage patterns, thereby facilitating informed decision-making in urban planning and development.

Nath et al. (2020) [7], showcase the development of smart efficiency parking systems highlights the critical importance of addressing parking space utilization challenges in increasingly crowded urban environments. As vehicle ownership continues to rise, traditional parking systems often result in wasted space between parked vehicles due to drivers' comfort requirements, traffic congestion and safety risks. This project focuses on eliminating wasted space in parking by implementing vehicle conveyors using bi-directional DC motors, conveyor belts, and NFC tagging for driver authorization. They use DC motors to control conveyor belt movement in parking slots and NFC cards for driver authorization, the system aims to optimize parking space utilization while enhancing driver convenience and safety. The Arduino system serves as the central coordination system, controlling all the activities within the project, while the L298 Motor Driver precisely control DC motor operations. In future, the project can implement technologies such as IoT, cloud computing, and AI to further enhance the overall performance of smart parking solutions.

Kannadasan et al. (2016) [10], pointed out that current parking systems suffer from inefficiencies, including the need for manual issuance of tokens, time-consuming processes, and high operational costs. To tackle these problems they have developed RFID-based automatic parking systems, using microcontrollers, sensing circuits, and IR sensors to monitor vehicle entry and exit. The system requires valid RFID tags for vehicle entry, with a central database maintaining accounts for each tag to facilitate automatic authorization and payment deduction. IR sensors are used to monitor parking slot occupancy in real time and LED lights to indicate slot status. The centralized database allows for analysis of parking demand patterns, offering insights for future planning and optimization of parking resources. Challenges that the system faces are the limited range of passive RFID tags and potential risks to data privacy exist. Future research can focus on enhancing system features, improving cost-benefit ratios, and addressing data privacy concerns to further optimize the overall performance of RFID-based automatic parking systems in urban environments.

Ebin et al. (2018) [12], proposed Android application that offers a comprehensive solution by automating the parking process, enabling users to quickly find available parking spots and complete payments seamlessly, thereby saving time and reducing inconvenience. The application allows users to rent out their parking spaces, harnessing unused resources and potentially generating additional income. Security measures are integrated to safeguard registered vehicles in the parking area, ensuring peace of mind for users. Through user registration and vehicle number scanning, the application tracks entry and exit, while cameras capture photos for enhanced monitoring. The Security protocol based on XOR-PRNG and Binary ECC enhances the system's security and reliability. RFID tags installed in vehicles helps in parking allocation, streamlining the process for users and administrator. Experimental results validate the functionality and effectiveness of the application system.

Melnyk et al. (2019) [13], introduce a Smart Parking Management System (SPMS) as a solution to tackle the challenges associated with parking congestion and inefficient space utilization. This system utilizes a sensing platform and a mobile application to enable real-time interaction for drivers, thereby reducing the time spent searching for parking spots and simplifying car localization. To validate the system's efficiency across various scenarios, small-scale test-beds have been developed, showcasing its effectiveness in optimizing parking operations. The system employs the Message Queuing Telemetry Transport (MQTT) protocol for data collection and processing, with data stored in SQL databases accessible via JDBC controllers and Apache Tomcat servers.

Azshwanth et al. (2019) [8] focuses on the need for solutions to alleviate the challenges of parking congestion and safety risks in urban environments. By using sensors in both vehicles and parking areas, the system offer real-time execution and pre-booking capabilities, allowing users to secure parking spaces in advance and minimize the time spent searching for parking. Mobile application facilitate user interaction, booking and access to parking information. The proposed solutions aim to provide a completely safe and automated parking experience, minimizing human interaction while ensuring efficient space allocation and utilization. Challenges in implementing this system consists of hard system scalability, integration with existing infrastructure, and user acceptance.

2.1.2 SMART PARKING USING MACHINE LEARNING

Thakur et al. (2024) [1] use YOLOv3 for Automatic Number Plate Recognition (ANPR). By eliminating errors associated with manual entry of vehicle registration information, the system utilizes ANPR system to accurately recognize vehicle number plates by processing the images and passing them through Optical Character Recognition (OCR) model. The Data produced contains vehicle registration details with its entry/exit timestamps and fees, facilitating efficient parking space utilization. They faced challenges such as lighting variations, weather conditions, and image distortion. The limitation consists of complexities in ANPR systems and variations in plate characteristics. Despite achieving a great accuracy score of 98% accuracy. But the model consists of the inherent complexities of ANPR systems; however, advancements in bounding box accuracy enable precise extraction of license plate numbers.

Choudhaury et al. [3] Showcase Chaurah, it is a low-cost parking system based on Raspberry Pi and Automatic Number Plate Recognition (ANPR) technology, presents their approach to parking management and vehicle surveillance. They used Convolutional Neural Networks (CNNs) for license plate detection and Optical Character Recognition (OCR) for character extraction, Chaurah offers a two-stage process for accurate identification of vehicles entering and exiting parking facilities. They integrated a Flutter and Firebase application to facilitate database management and billing. Chaurah eliminates the need for additional hardware on vehicles, enhancing its accessibility and cost-effectiveness. The utilization of AI-based algorithms and a low-cost microcontroller underscores its economic viability and simplicity of use in public parking facilities. They implemented YOLOv4 for Detection and Keras OCR pipeline for digit extraction further enhances its capabilities, while future research directions aim to improve OCR pipelines and object detection models for optimized performance. Overall, Chaurah represents a significant advancement in smart parking systems, offering a scalable, efficient, and cost-effective solution for managing parking facilities and enhancing urban mobility.

Lokhande et al. (2021) [4] highlights the significance of Automatic Number Plate Recognition (ANPR) systems for efficient traffic monitoring and the advancement of smart transport networks. ANPR systems, using image processing techniques combined with neural networks, offer reliable methods for extracting and recognizing vehicle number plate characters from images. With the improvement in digital camera technology over the past decade, ANPR has gained substantial interest due to its versatility in applications such as speed enforcement, parking management, and security control. The presented ANPR systems typically operate in three key steps: vehicle image detection, number plate detection and extraction, and character recognition, with variations in software and hardware models discussed in the literature. Lightweight ANPR system are capable of recognizing specific standard number plate formats, such as Sindh standard plates. Additionally, advancements in object detection and neural networks enable the detection of side views, tilted images, and moving images, further enhancing ANPR system capabilities. Future research directions focus on improving number plate recognition accuracy through the integration of high-resolution cameras with increased frames, suggesting a promising area for enhancing ANPR system performance in real-world traffic surveillance scenarios. The literature underscores the critical role of ANPR systems in enhancing traffic management efficiency and lays the groundwork for continued innovation in smart transport networks.

Vashishtha et al. [2], highlights the increasing importance of using advanced technologies to address the challenges of urban parking congestion and safety. The approach proposed in their research involves the integration of cameras and ArUco markers for object measurement and number plate detection, alongside the use of ultrasonic sensors for distance measurement to allocate parking spaces. Colored LEDs are used to indicate the vacancy status of parking slots, providing real-time information to drivers. NodeMCU is used for system development, the proposed model detects vehicle number plates, and suggests optimal parking spaces, enhancing parking management efficiency and increasing vehicle safety. Using Automatic Number Plate Recognition (ANPR) and Optical Character Recognition (OCR), the system achieves number plate recognition and database updating in real-time. Challenges they faced were varied plate information and design discrepancies.

Mahmood et al. (2019) [6], highlights the potential of solutions to streamline parking operations and enhance security. Traditional parking systems often rely on manual ticket printing and scanning processes, which can be clumsy and time-consuming. To tackle this problem they have proposed automated systems with cameras at entrance and exit points, leveraging face detection and recognition technology for the identification of drivers. The system includes modules for image acquisition, vehicle and face detection, and feature extraction. The vehicle detection methodologies results in high accuracy while face recognition algorithms may require further refinement, improving accuracy under varied lighting conditions and face poses. Ad-hoc experiments conducted using the Dell Precision Tower 7810 provide insights into system feasibility and performance evaluation. Future researches can focus on algorithm enhancements and object detection improvements under adverse weather conditions. Continued research and development efforts are needed to address existing challenges, the scalability and effectiveness of these automated parking solutions in real-world deployment scenarios.

Alharbi et al. (2021) [9], in their paper propose smart car parking system that offer a solution for time and effort savings, reduction of traffic congestion, and mitigation of environmental pollution. To address these issues, they proposed a system for pre-reservation of parking spaces, utilizing web applications using OCR for automatic detection of license plates. This system have been successfully implemented in some parking lots of Saudi Arabia, aiming to reduce traffic congestion and improve transportation services. By providing web-based applications for booking parking spaces, automatic gates with cameras for registration plate capture, and using image processing for booking details, smart car parking systems makes parking operations easier and enhance user convenience. Moreover, by implementing online payments for parking charges and allowing cancellations up to a designated time before entry, the systems offer flexibility and ease of use for drivers. While the proposed systems shows benefits in terms of reducing fuel consumption costs, minimizing parking issues in illegal places, and providing additional economic and investment income, challenges such as specific conditions for number plate detection and the need for oversight by parking management personnel warrant further consideration.

Jabbar et al. (2021) [15], point out that with the rise of urbanization and the increasing demand for parking spaces, universities worldwide are turning to smart campus initiatives to improve efficiency, space utilization, and the overall campus community experience. The proposed system offers a novel solution by using Raspberry Pi, Pi camera modules, GPS sensors, and ultrasonic sensors to keep a track of filled parking spots in real time. The system uploads data to an IoT server and display it on a mobile app, due to this users can easily find available parking spots, saving time and reducing congestion. The effectiveness and applicability of the system in smart campus environments have been validated, highlighting its potential to enhance parking management and improve campus mobility. Ultrasonic sensors, LEDs, and Pi cameras help to make accurate detection and real-time monitoring of parking spot availability, use of TensorFlow framework helps in object detection and classification. The system's ability to provide accessibility, intelligence, and improved user experience underscores its significance in addressing parking challenges and promoting sustainable urban mobility.

Chai et al. (2019) [11], present a novel parking guidance system that integrates features such as changing the parking spot and real-time ability to guide traffic. This system empowers drivers to reduce travel expenses by selecting optimal parking destinations and routes. Through microscopic simulation utilizing SUMO and OmNet++, the proposed system illustrates a decrease in overall travel costs and achieves balanced parking garage occupancy, effectively addressing issues of resource over- or under-utilization. By tackling the fundamental challenge of urban parking availability and providing a dynamic solution adaptable to real-time traffic conditions and user preferences, the proposed system represents a significant advancement in improving urban mobility and enhancing the quality of life for both city residents and commuters.

Yaseen et al. (2019) [16], focuses on the development of robust automatic number plate detection and automatic number plate recognition systems relies heavily on the availability of diverse and realistic datasets for training and evaluation. The paper introduces a novel dataset named NI-VI comprising 1500 vehicle images captured in the northern region of Iraq. NI-VI is a collection of images that are rotated, scaled, and

translated, reflecting the variability encountered in real-world scenarios. These images were captured using handheld cameras in real-time, ensuring authenticity and relevance to the target environment. The paper critically evaluates existing ANPR datasets, highlighting limitations such as low resolutions, lack of detail, and inadequate coverage of weather conditions, pointing the need for datasets that capture the intricacies of diverse environments.

2.1.3 INTER VEHICLE DISTANCE

Zaarane et al. (2020) [5], in their paper particularly focuses on inter-vehicle distance measurement, and brings the attention to the significance of accurate and reliable techniques for ensuring safe and efficient operation in dynamic traffic environments. One of the approach proposed in recent research involves the utilization of stereo cameras and image processing techniques to detect vehicles and calculate inter-vehicle distances. This method typically employs template matching techniques for vehicle detection in one camera and subsequent matching in the other, leveraging stereo vision techniques for depth information extraction. By utilizing geometric derivations and technical data, the proposed systems enable precise calculation of inter-vehicle distances, providing higher accuracy compared to previous works. Experiments conducted on Hardware Processor Systems (HPS) validate the efficacy of the proposed method, highlighting its applicability in real-time autonomous driving systems. While mono vision methods may require extensive datasets for object recognition, stereo vision systems face challenges in object matching between cameras; however, the proposed approach overcomes these limitations, providing a practical solution for computing safety distances and measuring object distances in autonomous vehicle applications.

2.2 LITRATURE REVIEW SUMMARY

The reviewed papers collectively contribute to enhancing parking space efficiency through innovative technological solutions. These approaches range from the utilization of IoT and Raspberry Pi for real-time monitoring and occupancy display, to RFID-based systems for streamlined entry and exit processes. Machine learning models, such as YOLOv3 and VGG16, are employed for license plate detection and recognition, while dynamic systems integrate parking destination switching for optimal resource utilization. Various papers emphasize user-friendly interfaces, mobile applications, and web-based frameworks to facilitate parking space booking, reducing time wastage and improving traffic management. Overall, these diverse methodologies collectively contribute to making parking systems more efficient, convenient, and responsive to the dynamic demands of urban environments.

Chapter 3

Methodology

This chapter discusses about the details of each modules of the final system and the flow of each modules and how do they fit together in the final system.

The system is made up of 3 modules which are compiled to form the final product:

- Automatic Number Plate Recognition (ANPR) Module
- Measurement/Stereovision Module
- Parking Management Module

3.1 AUTOMATIC NUMBER PLATE RECOGNITION (ANPR) MODULE

Automatic Number Plate Recognition (ANPR), also known as Automatic License Plate Recognition (ALPR) in some regions, is a technology that uses optical character recognition (OCR) to automatically read and interpret the license plates on vehicles.

ANPR system typically consists of two modules:

- The number plate detection module
- Optical Character Recognition (OCR) module

3.1.1 NUMBER PLATE EXTRACTION

The number plate extraction module is tasked with isolating the number plate within a provided input image and providing the coordinates of the bounding box surrounding it. This enables the segmentation of the license plate from the image, allowing it to be passed as input to the OCR module for the extraction of the number embedded on the plate.

3.1.2 OPTICAL CHARACTER RECOGNITION MODULE

In Automatic Number Plate Recognition (ANPR) systems, Optical Character Recognition (OCR) serves as a pivotal component for identifying and extracting the alphanumeric characters found on vehicle license plates.

3.2 MEASUREMENT / STEREOVISION MODULE

The measurement module is tasked with determining the size of vehicles as they enter the parking lot, categorizing them as either large or small. This is achieved through the utilization of Stereovision Camera technology, which calculates the distance of the vehicle from the camera. Subsequently, the length of the vehicle is computed based on this calculated distance.

3.2.1 HOW DOES STEREOVISION CAMERA WORK

Stereovision, also known as stereo vision or stereoscopic vision, is a visual perception process that involves the simultaneous use of two eyes to perceive depth, distance, and three-dimensional (3D) structure of objects in the environment.

The human visual system has two eyes positioned slightly apart from each other, resulting in each eye receiving a slightly different view of the world. This phenomenon is known as binocular disparity. The brain processes these two slightly different views and combines them to create a single, coherent 3D perception of the scene. By analyzing the differences in the images received by the two eyes, the brain can perceive depth and estimate distances to objects in the environment. Objects that are closer to the observer will produce larger disparities between the images received by each eye, while objects that are farther away will produce smaller disparities.

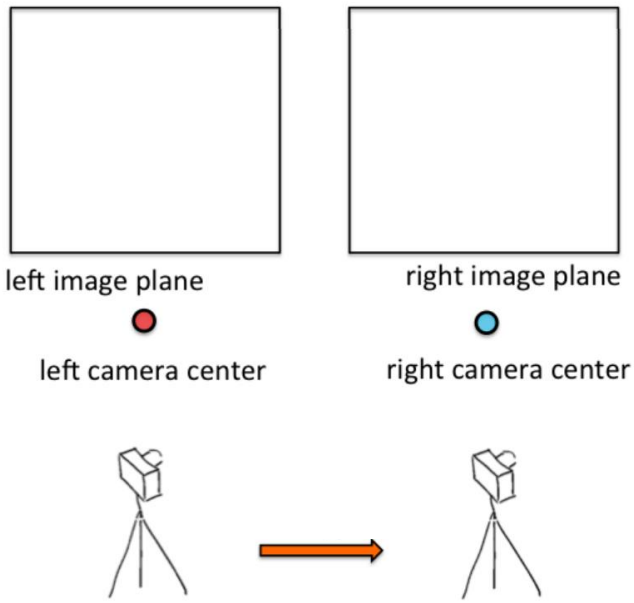

Fig. 3.1 Parallel configuration of stereo cameras

In fig. 3.1, assume that the cameras are calibrated to remove any distortion. The cameras are placed parallel to each other, i.e. the right camera will be a fixed distance to the right of the left camera. This distance between the cameras is called the baseline.

In fig. 3.2, shows the simplified diagram of the configuration shown in fig. 3.1, for ease of understanding and calculation of depth.

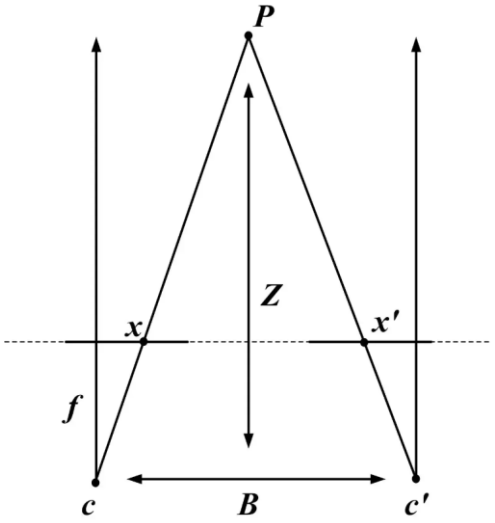

Fig. 3.2 Simplified Diagram for Depth calculation

From fig. 3.2 we have

P: a point in real world,

Z: the distance between P and the cameras,

C and C': represents the two parallel cameras,

B: distance between the two cameras

f: focal length of the cameras

x and x' are the image planes of the cameras C and C' respectively.

Now by triangulation, we can compute the depth Z according to formula (1):

$$Z = \frac{f.B}{(x - x')} \qquad \text{- (1)}$$

Once we have the distance Z, we can calculate the size of the object by calculating α_1 and α_2 which are the angles the vehicle's front and end projects on the centre of camera C, and using the values in formula (2):

$$S = \tan^{-1}\alpha_1 + \tan^{-1}\alpha_2 \qquad \text{- (2)}$$

Now this size can be used to appropriately allot the vehicle parking space.

3.3 PARKING MANAGEMENT MODULE

This module is responsible to keep the log of entry and exit of the vehicles, keep a track of occupied and unoccupied parking lots of both small and large kind.

3.4 PROPOSED SYSTEM

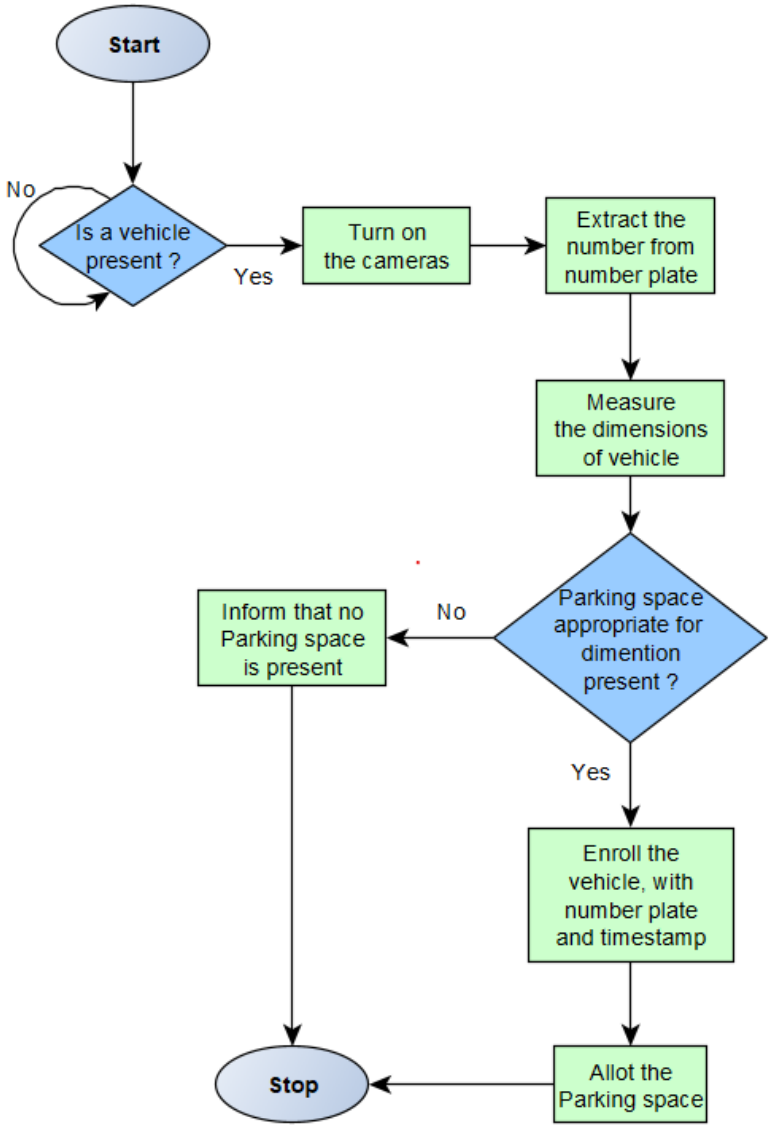

Fig. 3.3 Proposed Parking System Diagram

The system works as represented in the flowchart in Fig. 3.3, an ultrasonic sensor checks whether a vehicle is waiting or not, if the vehicle is waiting the cameras turn on. The images taken from camera is used to first extract the number plate and if the number plate is valid only then measure the dimensions from the images. Once both of these data points are extracted, the system checks whether an appropriate parking space is available according to the size of the vehicle.

There are 3 possible cases:

Case 1: The appropriate parking space is available and the vehicle is given the entry.

Case 2: The vehicle is large and there is no large parking space available, the vehicle will be informed that there is no parking space available.

Case 3: The vehicle is small and there is only large parking spaces available, the vehicle will be allowed to enter and park in the large parking space.

Further details of Implementing these systems and modules are present in the chapter 4.

Chapter 4

Implementation

This chapter will further explain the modules and system defined in the previous chapter and explain the way they were implemented in this project.

4.1 ANPR USING MOBILENETV2 AND EASY OCR

4.1.1 NUMBER PLATE DETECTION USING MOBILENETV2

This project implements MobileNetV2 trained on coco17 dataset to detect the number plate and segment it from the image.

MOBILENET ARCHITECTURE

MobileNet is a convolutional neural network architecture designed for efficient mobile and embedded vision applications. MobileNet is specifically designed to have a small memory footprint and be computationally efficient while still maintaining reasonable accuracy for tasks like image classification, object detection, and semantic segmentation. This makes it suitable for deployment on devices with limited computational resources, such as smartphones, tablets, and embedded systems.

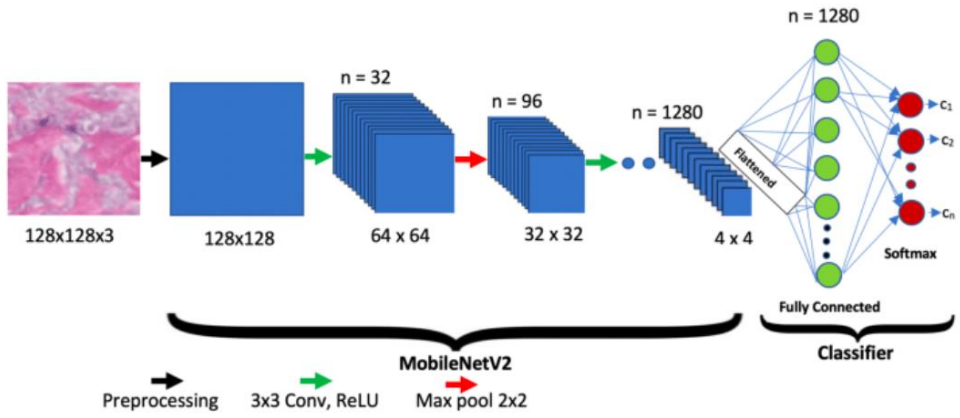

Fig. 4.1 MobileNetV2 architecture overview

The architecture visible in fig. 4.1 comprises several components:

Depthwise Separable Convolution: In MobileNetV2, depthwise separable convolutions are employed, which decompose the traditional convolution into depthwise and pointwise convolutions. This strategy notably decreases computational complexity while maintaining effectiveness.

Inverted Residuals: Inverted residuals incorporate a bottleneck structure that first expands channel dimensions prior to applying depthwise separable convolutions. This expansion amplifies the model's ability to capture complex features, ultimately leading to enhanced accuracy.

Bottleneck Design: MobileNetV2 incorporates a bottleneck design featuring 1x1 convolutions to reduce channel dimensions before depthwise separable convolutions. This design optimizes the trade-off between model size and accuracy.

Linear Bottlenecks: To address information loss during the bottleneck process, linear bottlenecks are introduced. These bottlenecks utilize linear activations instead of non-linear ones, thereby enhancing information preservation and resulting in improved capture of details.

Squeeze-and-Excitation (SE) Blocks: MobileNetV2 incorporates SE blocks to improve feature representation. These blocks dynamically modulate channel-wise feature responses, enabling the model to prioritize informative features while attenuating less pertinent ones.

The MobileNetV2 is trained on the dataset mentioned in the previous chapter, to detect and extract the number plate image from the whole image.

4.1.2 LICENCE PLATE CHARACTER EXTRACTION

The image of license plate extracted is then given as input to the EasyOCR model to extract the number from the license plate.

EASYOCR (OPTICAL CHARACTER RECOGNITION)

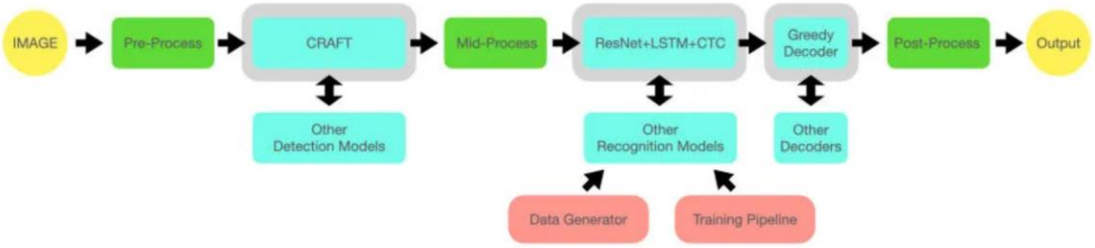

Fig. 4.2 EasyOCR Framework

Fig. 4.2 provides a rough flow diagram of how the system processes an image and produces a text as output.

EasyOCR is a Python library for Optical Character Recognition (OCR). It's designed to be simple to use while still providing powerful text extraction capabilities from images and scanned documents. With EasyOCR, developers can integrate OCR functionality into their Python applications with ease, enabling tasks such as data entry automation, image analysis, and text extraction from images.

EasyOCR is composed of three main components: feature extraction, sequence labeling, and decoding, which collectively enable the extraction of text from images effectively.

1 Feature Extraction (ResNet and VGG):

Feature extraction is the initial step in the OCR process, where input images are converted into a set of features necessary for further analysis. EasyOCR employs ResNet and VGG, two deep learning models, for this purpose.

- **ResNet**: also referred to as Residual Networks, represents a convolutional neural network (CNN) variant specifically crafted to tackle the complexities associated with training deep networks. This is accomplished through the integration of skip connections, allowing the network to attain greater depth without compromising its trainability.

- **VGG**: Visual Geometry Group (VGG) is another CNN architecture known for its uniform design and simplicity. It utilizes small convolution filters stacked in deep layers, facilitating network understanding and modification.

2 Sequence Labeling (LSTM):

After extracting features, the next step involves sequence labeling to interpret the sequential context of these features. EasyOCR employs Long Short-Term Memory (LSTM) networks for this purpose.

LSTM networks belong to the category of recurrent neural networks (RNNs) and are tailored to capture temporal dependencies across prolonged sequences. Their unique architecture, which includes memory cells and gates, facilitates efficient sequence labeling by effectively managing and retaining information over time.

3 Decoding (CTC):

Decoding is the final step in the recognition process, where labeled sequences are transcribed into recognized text. EasyOCR employs the Connectionist Temporal Classification (CTC) algorithm for decoding.

CTC is a loss function suitable for sequence tasks where the alignment between labels and input data is uncertain. It handles variable-length predictions by appending a blank label and computing loss over all possible alignments, making it suitable for OCR decoding tasks.

EasyOCR's training pipeline is based on an adapted version of the deep-text-recognition-benchmark framework. This framework enhances text recognition in images and provides a robust foundation for OCR execution, allowing training on various datasets and ensuring versatility and effectiveness in text extraction tasks.

4.2 STEREOVISION CAMERA

A stereo camera is a type of camera that contains two or more lenses positioned to mimic human binocular vision. By having multiple lenses or sensors separated by a specific distance, stereo cameras are capable of capturing two slightly offset images simultaneously, much like our eyes perceive the world. These offset images, known as a stereo pair, are then processed to create a three-dimensional (3D) representation of the scene.

4.2.1 BUILDING THE STEREOVISION CAMERA

To build the stereo camera, we need:

- Two ESP32-CAM boards,
- A Breadboard,
- FTDI programmer (or Alternatively Arduino or ESP)
- Jumper cable

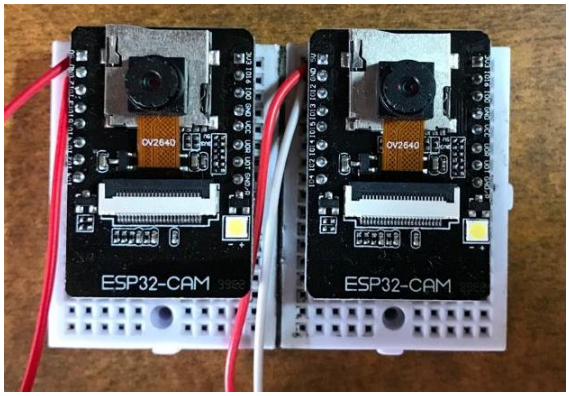

Fig. 4.3 Stereo camera

The Breadboard acts as a stable surface to attach both the ESP32-CAM modules as shown in Fig. 4.3, as the distance between both the lenses should be fixed and not change during the calibration and while in use.

Upload the code for the server and the client on the ESP32-CAMs, you can look at the code in appendix 1, once the code is uploaded the ESP32-CAMs will stream the video and images to their respective IP addresses.

4.2.2 DEPTH AND SIZE ESTIMATION

This process involves comparing the differences between the two images from the two calibrated cameras as visible in Fig. 4.4 and using principles of triangulation to determine the distance to objects in the scene.

Fig. 4.4 Stereo pair (left frame and right frame respectively)

Calibrate the ESP32-CAMs to remove any distortion, in individual cameras, using these calibrated cameras get a stereo pair which contains a vehicle, use a YOLO model to detect the vehicle in both the images. Once you have detected the vehicle, we need a point in both images which correlates to the same point in the real world. So to get these points take the mid points of both the vehicles, as the cameras are parallel the mid of both vehicles will correlate to the same point in real world.

Now using the disparity between these points calculate the distance between the vehicle and the cameras, this distance then can be used to calculate the length of the vehicle using the formula mentioned in chapter 3.

4.3 PARKING MANAGEMENT SYSTEM

The parking management system consists of the following hardware:

- ESP8266
- Ultrasonic sensors

The primary function of the parking management system is to monitor and manage the availability of parking spaces, including spaces of various sizes. It aims to keep track of the number of vacant parking spots, categorized by their size, within a parking facility. Additionally, the system evaluates whether a vehicle can be accommodated within the available parking spaces based on its size and the remaining capacity of the parking lot.

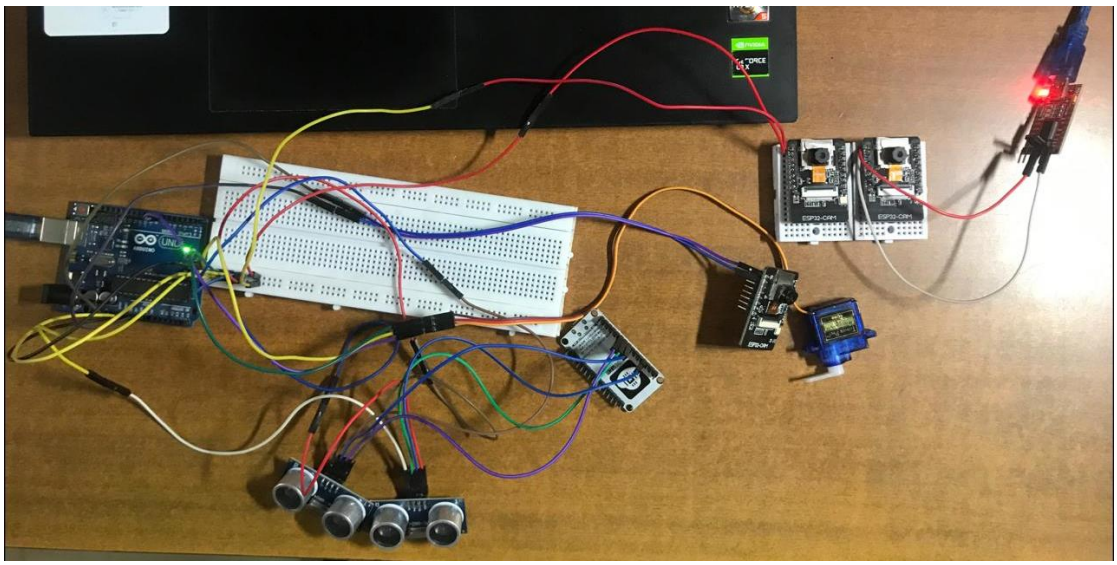

Fig. 4.5 Parking management hardware

As visible in Fig. 4.5, the Ultrasonic sensor are responsible to detect whether any vehicles are parked on top of it or not and the servo is responsible to open and close the barrier.

The Parking Management system consists of a server and the sensors fitted under the parking lot to determine whether the parking space is occupied or not. The server is responsible for the decision making and processing the data transmitted from the ESP8266. The ultrasonic sensors are used to detect the occupancy of the parking lot. The code for ESP8266 can be referred to in Appendix 1.

Chapter 5

Results and Discussion

The Results and Discussion is divided into each module and then the final system.

5.1 MOBILENETV2 ANPR SYSTEM

The MobileNewV2 was trained on our custom data was trained to have the following metrics:

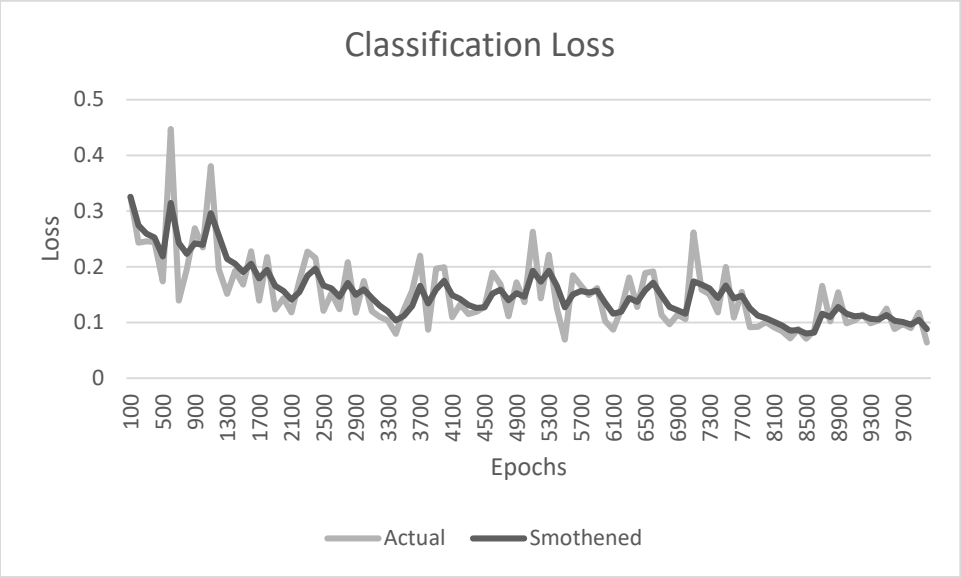

Fig. 5.1 Classification loss

Classification loss, as illustrated in Figure 5.1, concerns the accurate classification of objects within an image. In tasks like object detection, this involves predicting the class labels assigned to objects identified within bounding boxes. The loss function utilized for classification loss is binary cross-entropy loss, which penalizes the model for differences between predicted class probabilities and the actual class labels in the ground truth. Classification loss makes sure the model effectively categorizes the objects depicted in the image, irrespective of their specific spatial placement.

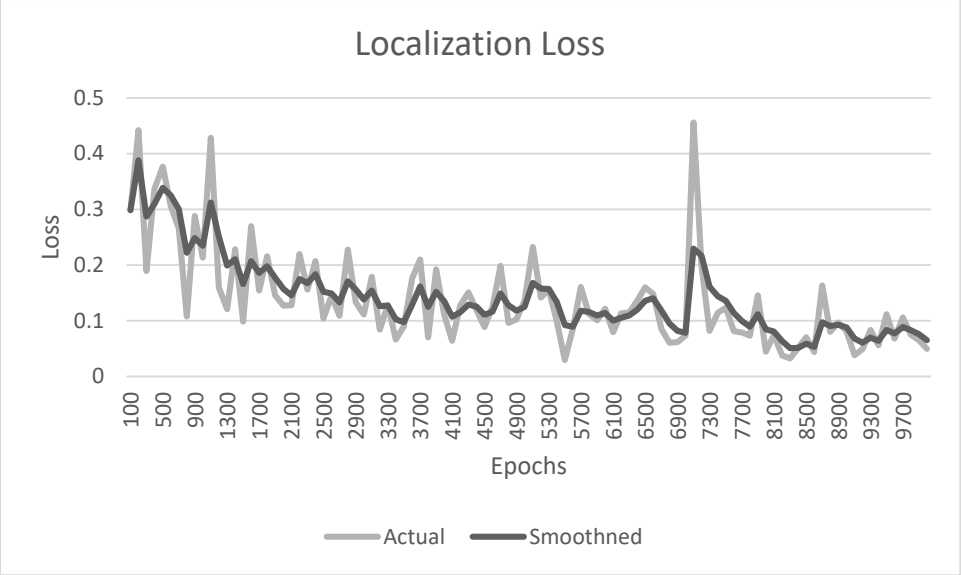

Fig. 5.2 Localization loss

Localization loss, depicted in Figure 5.2, pertains to the task of accurately determining the position of objects within an image. In scenarios such as object detection, this entails predicting bounding boxes around target objects, such as a license plate. The loss function employed for localization loss is the smooth L1 loss, which penalizes the model for differences between predicted bounding box coordinates and the actual coordinates of the bounding boxes in the ground truth. The aim of localization loss is to ensure the model precisely identifies the location of objects in the image by making accurate predictions of bounding box coordinates.

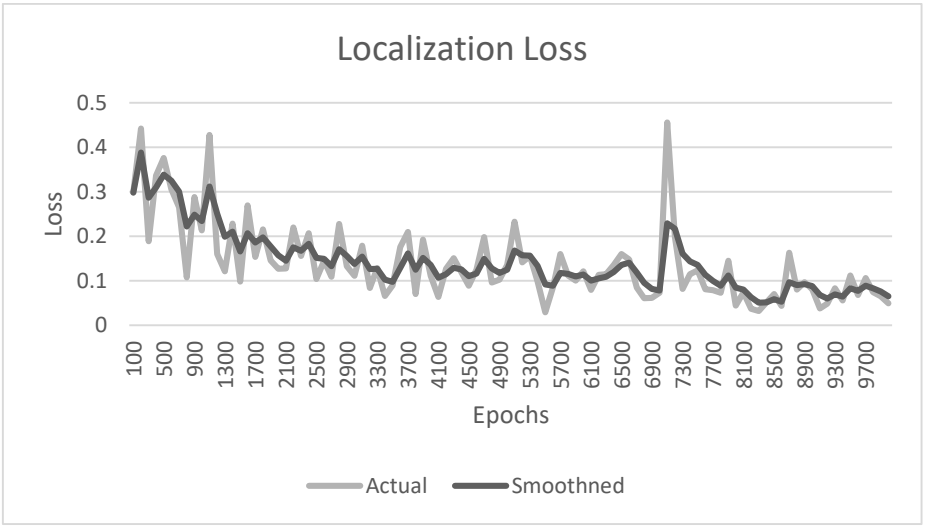

Fig. 5.3 Regularization loss

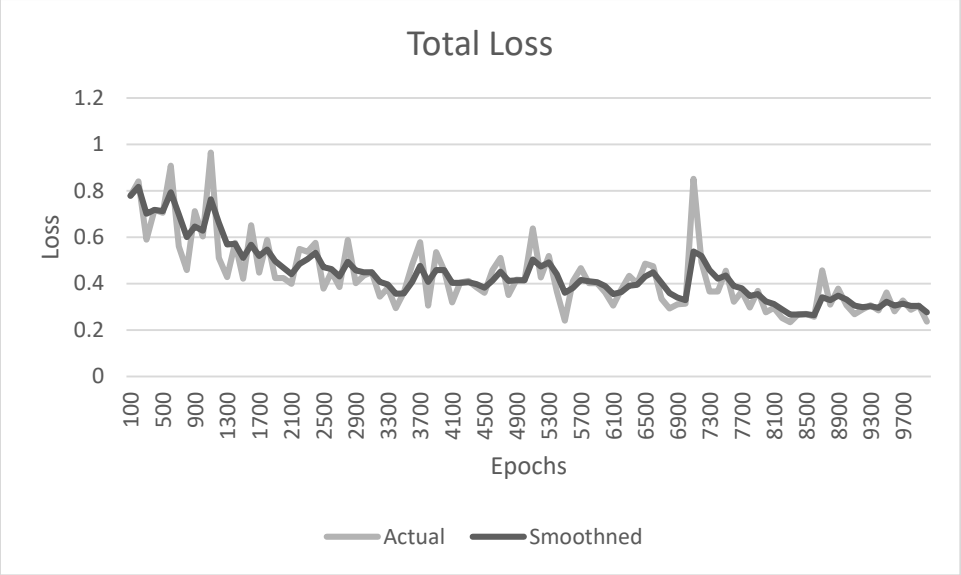

Fig. 5.4 Total loss

The total loss, fig. 5.4, is combination of both the localization loss and the classification loss. This combined loss is often referred to as the "total loss" or "composite loss." The total loss is calculated by summing or averaging the individual losses from both localization and classification components.

TYPE OF LOSS	LOSS VALUE
CLASSIFICATION LOSS	0.06359
LOCALIZATION LOSS	0.04944
TOTAL LOSS	0.2358

Table 1: loss metrics

AVERAGE PRECISION FOR DIFFERENT IOUS	PRECISION VALUES
IOU=0.50:0.95	0.542
IOU=0.50	0.748

Table 2: Precision Values of different IoUs

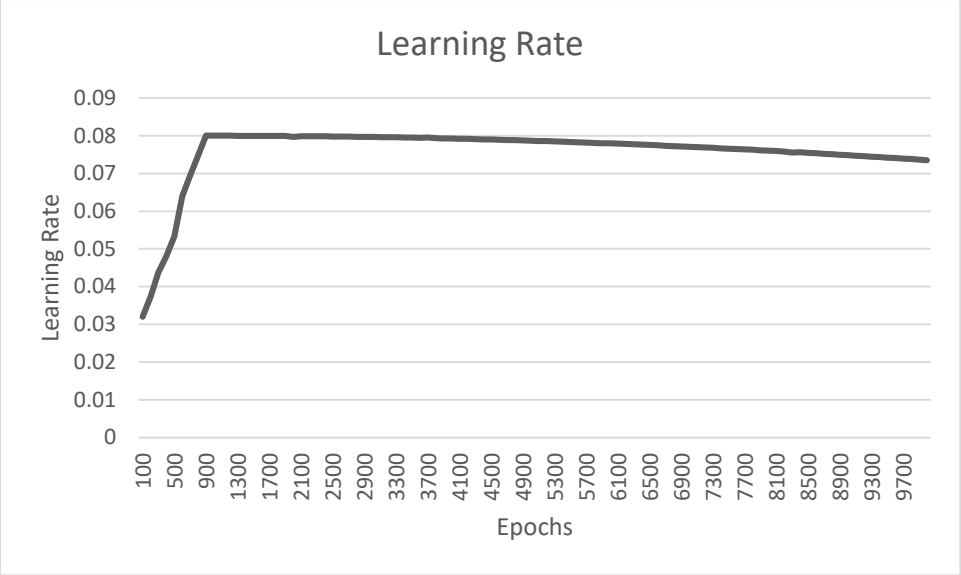

Fig. 5.5 Learning rate

The results on an image from testing dataset and live detection are as follows:

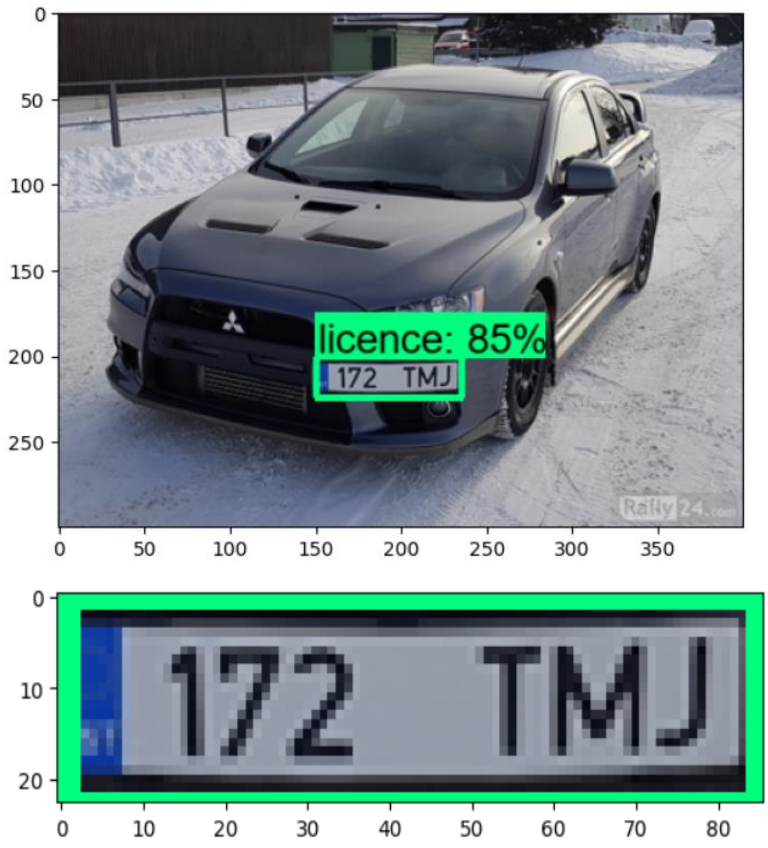

Fig. 5.6 License plate extraction from testing image

Fig. 5.7 License plate extraction from live dataset

As visible from fig. 5.6 and fig. 5.7, The system works better on a higher resolution image, the ESP 32 CAM is not able to produce that high resolution images, but still the system is able to extract the number from the licence plate with a decent accuracy.

5.2 STEREOVISION CAMERA

The stereo vision system works fine on live data. With an error of ±1cm while estimating the length of the vehicle.

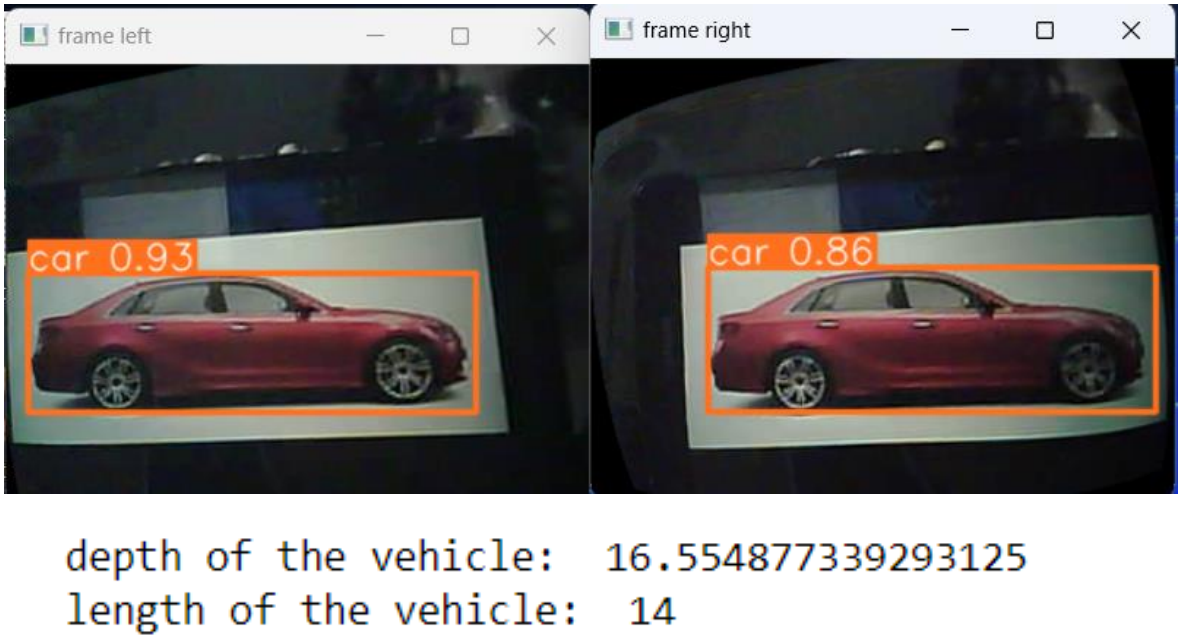

Fig. 5.8 Live length calculation for large vehicle

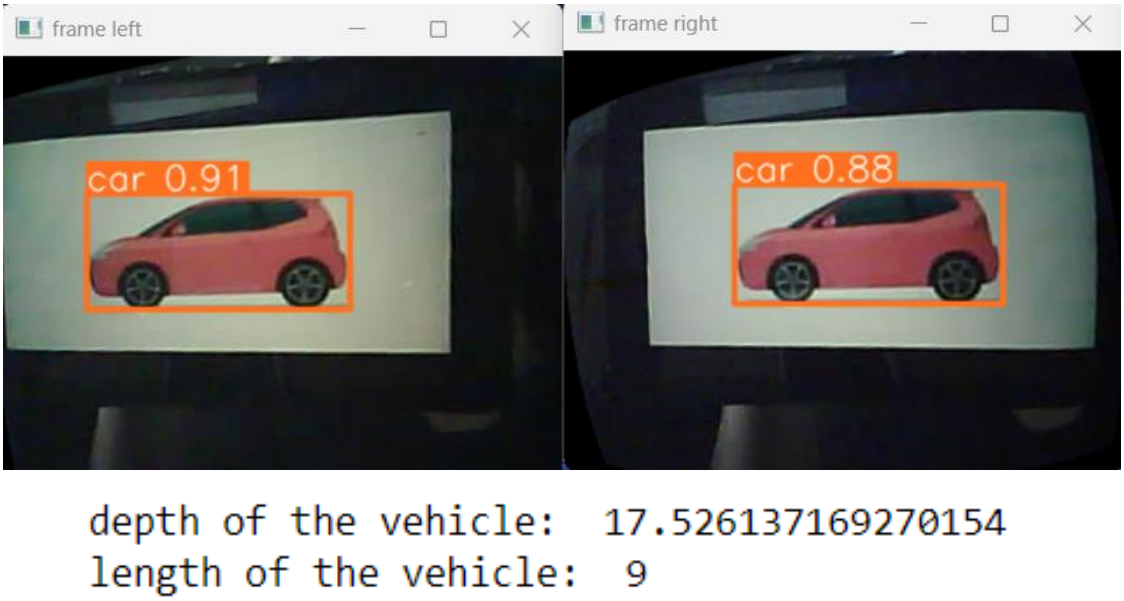

Fig. 5.9 Live calculation for small vehicle

From fig. 5.8 and fig. 5.9, the working of the stereovision system can be observed. The system first calibrates the left and right frames to remove any distortions and then does a object recognition of both of the frames to detect the vehicle in both the frames and hence calculating the distance and size.

The same problem is present in this process that ESP 32 CAM is not able to produce a high resolution image, due to the direct correlation of number of pixels and size, the system produces this error of ±1cm.

5.3 COMPLETE SYSTEM

The complete system works as explained in the system diagram in previous chapters. The ANPR system extracts the license plate from the input image, if a license plate is present, the stereovision system takes images to calculate the size of the vehicle, if the parking space is available then the vehicle is allowed to enter the parking lot, and the database is updated accordingly with license plate, vehicle size, parking space allotted and the timestamps.

The fig. 5.10 to fig. 5.12 are the output of the complete system:


```
['21 BH 2345 AA']
lenght of the vehicle is 14
please move to Slot: L1
```

Fig. 5.10 Appropriate Slot is available


```
['21 BH 2345 AA']
lenght of the vehicle is 14
No slots are available
```

Fig. 5.11 Large vehicle and no large slots


```
['HR26D05551']
lenght of the vehicle is 10
no small slots available, please move to Slot: L1
```

Fig. 5.12 Small vehicle and only large slots are available

As seen from fig. 5.10, 5.11 and 5.12, when there is appropriate slot available for the vehicle, the system assigns a slot for that vehicle but when there is no slot available for the large vehicle or the small vehicle the system prompts the user about it, if there is large slots available and a small vehicle has approached the parking lot, the system assigns the small vehicle large slot.

Chapter 6

Conclusion and Future Work

Overall, the project has demonstrated commendable performance, with the Automatic Number Plate Recognition (ANPR) system yielding satisfactory loss and precision metrics, and the stereovision system, made using ESP32-CAM, provided a surprisingly accuracy in estimating vehicle sizes with an error of ±1cm. The modular design of the parking system allows for easy scalability, facilitating potential expansion to larger deployments. Crucially, the project achieves its intended objectives, enhancing the efficiency of parking lot space utilization without imposing additional prerequisites on users. However, there remain areas for enhancement, particularly in refining the OCR and ANPR systems for improved accuracy, as well as fine-tuning the stereovision camera calibration for greater precision.

Looking ahead, several points for improvement present themselves. Firstly, enhancing the Optical Character Recognition (OCR) and ANPR systems could significantly boost accuracy, potentially through leveraging advanced machine learning algorithms or refining image preprocessing techniques. Optimizing the mounting and calibration of the stereovision camera on a more stable base could further enhance its accuracy in estimating vehicle sizes, potentially reducing the margin of error even further. Exploring innovative technologies, such as incorporating machine learning for real-time parking space allocation or integrating IoT sensors for dynamic parking management, could further elevate the efficiency and usability of the system. Overall, continued research and development efforts in these areas promise to further advance the effectiveness and scalability of the parking system, ultimately benefiting both operators and users alike.

REFERENCES

[1] Thakur, N., Islam, S. M., Neyaz, Z., Sadhwani, D., & Jain, R. (2022). Smart Parking System Using YOLOv3 Deep Learning Model. In *Applications of Artificial Intelligence, Big Data and Internet of Things in Sustainable Development* (pp. 49-63). CRC Press.

[2] Vashishtha, S., Joshi, S., Verma, R., Verma, S., & Kumar, V. Smart Parking System using Machine Learning. International Journal of Computer Applications, 975, 8887.

[3] Choudhaury, S. R., Narendra, A., Mishra, A., & Misra, I. (2023). Chaurah: A Smart Raspberry Pi based Parking System. arXiv preprint arXiv:2312.16894.

[4] Lokhande, S., Sahane, P., Katare, S., Tawhare, M. D., & Khandekar, M. S. (2021). Automatic number plate recognition System for Vehicle Identification using Machine Learning. International Journal of Creative Research Thoughts (IJCRT), 9(11).

[5] Zaarane, A., Slimani, I., Al Okaishi, W., Atouf, I., & Hamdoun, A. (2020). Distance measurement system for autonomous vehicles using stereo camera. Array, 5, 100016.

[6] Mahmood, Z., Haneef, O., Muhammad, N., & Khattak, S. (2019). Towards a fully automated car parking system. IET Intelligent Transport Systems, 13(2), 293-302.

[7] Nath, A., Konwar, H. N., Kumar, K., & Islam, M. R. (2020). Technology enabled smart efficiency parking system (TESEPS). In Trends in Communication, Cloud, and Big Data: Proceedings of 3rd National Conference on CCB, 2018 (pp. 141-150). Springer Singapore.

[8] Azshwanth, D., Koshy, M. T., & Balachander, M. T. (2019, November). Automated car parking system. In Journal of Physics: Conference Series (Vol. 1362, No. 1, p. 012059). IOP Publishing.

[9] Alharbi, A., Halikias, G., Yamin, M., & Abi Sen, A. A. (2021). Web-based framework for smart parking system. International Journal of Information Technology, 13(4), 1495-1502.

[10] Kannadasan, R., Krishnamoorthy, A., Prabakaran, N., Naresh, K., Vijayarajan, V., & Sivashanmugam, G. (2016). RFID based automatic parking system. Australian journal of basic and Applied Sciences, 10(2), 186-191.

[11] Chai, H., Ma, R., & Zhang, H. M. (2019). Search for parking: A dynamic parking and route guidance system for efficient parking and traffic management. Journal of Intelligent Transportation Systems, 23(6), 541-556.

[12] Ebin, P. M., Dev, P. A., Mishab, P., Sreejith, C., & Srudhil, U. K. (2018, July). An Andriod Application for Smart Parking With Efficient Space Management. In 2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR) (pp. 1-5). IEEE.

[13] Melnyk, P., Djahel, S., & Nait-Abdesselam, F. (2019, October). Towards a smart parking management system for smart cities. In 2019 IEEE International Smart Cities Conference (ISC2) (pp. 542-546). IEEE.

[14] Elsonbaty, A., & Shams, M. (2020). The smart parking management system. arXiv preprint arXiv:2009.13443.

[15] Jabbar, W. A., Wei, C. W., Azmi, N. A. A. M., & Haironnazli, N. A. (2021). An IoT Raspberry Pi-based parking management system for smart campus. Internet of Things, 14, 100387.

[16] Yaseen, N. O., Al-Ali, S. G. S., & Sengur, A. (2019, November). Development of new Anpr dataset for automatic number plate detection and recognition in north of Iraq. In 2019 1st International Informatics and Software Engineering Conference (UBMYK) (pp. 1-6). IEEE.

Appendices

Appendix 1

Hardware Code

1.1 ESP8266

```
#include "ESP_MICRO.h"


#define trigPin 13
#define t2 0
#define slot3 14
#define slot4 12


long duration;
int distance;

int S1 = 0, S2 = 0, S3=0, S4 = 0;
int flag1 = 0, flag2 = 0;
int slot = 4, total = 0;
int d1, d2, dis1, dis2;

int x, pos = 0;

void Read_Sensor(){
  S1 = 0, S2 = 0, S3=0, S4=0;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  d1 = pulseIn(slot3, HIGH);

  digitalWrite(t2, LOW);
  delayMicroseconds(2);
  digitalWrite(t2, HIGH);
  delayMicroseconds(10);
  digitalWrite(t2, LOW);
  d2 = pulseIn(slot4, HIGH);
  dis1 = d1*0.034/2;
  dis2 = d2*0.034/2;

  Serial.print(dis1+"\t");
```

```
  if(dis1 <=20){S3 = 1;}
  if(dis2 <=20){S4 = 1;}

}

void setup(){

  Serial.begin(115200);
  Serial1.begin(115200);
  start("ANUJ-PC 8708","abcd12345");


  pinMode(trigPin, OUTPUT);
  pinMode(t2, OUTPUT);
  pinMode(slot3, INPUT);
  pinMode(slot4, INPUT);


  delay(2000);
  Read_Sensor();
  total = S1+S2;
  slot = slot-total;
}

bool flg = true;
void loop(){
  waitUntilNewReq();
  String s = "";//Waits until a new request from python come
  if(1){
    Read_Sensor();
    delay(1000);

    if(S3 == 1){s = s+"S3: Fill,";}
    else{s = s+"S3: Empty,";}

    if(S4 == 1){s = s+"S4: Fill,";}
    else{s = s+"S4: Empty,";}

    if(S3 == 0 || S4 == 0){
      if(flg){
        Serial1.println("1");
        //Serial1.println("0");
        delay(5000);
        flg =!flg;
      }else{
        flg=!flg;
      }

      s = s+"Ok";
    }else{
```

```
    s = s+"No Slots Available";
   }
   returnThisStr(s);
   x=0;
  }

}
```

1.2 ESP32 CAM (Server Side)

```
#include "esp_camera.h"
#include <WiFi.h>

const char* wifiSSID = "abcd";
const char* wifiPassword = "abcd12345";

void startCameraServer();

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Serial.begin(115200);
  Serial.println();

  camera_config_t cameraConfig;
  cameraConfig.ledc_channel = LEDC_CHANNEL_0;
  cameraConfig.ledc_timer = LEDC_TIMER_0;
  cameraConfig.pin_d0 = Y2_GPIO_NUM;
  cameraConfig.pin_d1 = Y3_GPIO_NUM;
  cameraConfig.pin_d2 = Y4_GPIO_NUM;
  cameraConfig.pin_d3 = Y5_GPIO_NUM;
  cameraConfig.pin_d4 = Y6_GPIO_NUM;
  cameraConfig.pin_d5 = Y7_GPIO_NUM;
  cameraConfig.pin_d6 = Y8_GPIO_NUM;
  cameraConfig.pin_d7 = Y9_GPIO_NUM;
  cameraConfig.pin_xclk = XCLK_GPIO_NUM;
  cameraConfig.pin_pclk = PCLK_GPIO_NUM;
  cameraConfig.pin_vsync = VSYNC_GPIO_NUM;
  cameraConfig.pin_href = HREF_GPIO_NUM;
  cameraConfig.pin_sscb_sda = SIOD_GPIO_NUM;
  cameraConfig.pin_sscb_scl = SIOC_GPIO_NUM;
  cameraConfig.pin_pwdn = PWDN_GPIO_NUM;
  cameraConfig.pin_reset = RESET_GPIO_NUM;
  cameraConfig.xclk_freq_hz = 20000000;
  cameraConfig.pixel_format = PIXFORMAT_JPEG;

  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
```

```
  // for larger pre-allocated frame buffer.
  if(psramFound()){
    cameraConfig.frame_size = FRAMESIZE_UXGA;
    cameraConfig.jpeg_quality = 10;
    cameraConfig.fb_count = 2;
  } else {
    cameraConfig.frame_size = FRAMESIZE_SVGA;
    cameraConfig.jpeg_quality = 12;
    cameraConfig.fb_count = 1;
  }

  // camera init
  esp_err_t initError = esp_camera_init(&cameraConfig);
  if (initError != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", initError);
    return;
  }

  sensor_t * sensor = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (sensor->id.PID == OV3660_PID) {
    sensor->set_vflip(sensor, 1); // flip it back
    sensor->set_brightness(sensor, 1); // up the brightness just a bit
    sensor->set_saturation(sensor, -2); // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  sensor->set_framesize(sensor, FRAMESIZE_QVGA);

  WiFi.softAP(wifiSSID, wifiPassword);

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10000);
}
```

1.3 ESP32 CAM (Client Side)

```
#include "esp_camera.h"
#include <WiFi.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

// Select camera model
//#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
//#define CAMERA_MODEL_ESP_EYE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
//#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B
Has PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
//#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
#define CAMERA_MODEL_AI_THINKER // Has PSRAM
//#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM

#include "camera_pins.h"

const char* ssid = "abcd";
const char* password = "abcd12345";

void startCameraServer();

void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
```

```
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;

  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
  //                     for larger pre-allocated frame buffer.
  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1); // flip it back
    s->set_brightness(s, 1); // up the brightness just a bit
    s->set_saturation(s, -2); // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  s->set_framesize(s, FRAMESIZE_QVGA);

#if                     defined(CAMERA_MODEL_M5STACK_WIDE)                  ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif
  //WiFi.softAP(ssid, password);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10000);
}
```

1.4 Arduino

```
#include<SoftwareSerial.h>

#include <Servo.h>

Servo myservo;


void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  myservo.attach(9);//D4
  myservo.write(0);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(Serial.available() > 0){
    byte x = Serial.read();
    Serial.print(char(x));
    if(x == '1'){
      for(int pos = 0; pos<=90; pos+=1){
        myservo.write(pos);
```

```
      delay(15);
     }
     delay(3000);
     for(int pos = 90; pos>=0; pos-=1){
      myservo.write(pos);
      delay(15);
     }
     delay(5000);
    }
  }
}
```

Appendix 2

Number Plate Recognition Training and Deploying Code

```
import os
import cv2
import numpy as np
import easyocr
import urllib.request
import socket
from datetime import datetime
import tensorflow as tf
from object_detection.utils import config_util, label_map_util, visualization_utils
as viz_utils
from object_detection.builders import model_builder

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files[CONFIG'])
detection_model = model_builder.build(model_config=configs['model'],
is_training=False)
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT'], 'ckpt-11')).expect_partial()

# Function to perform object detection
@tf.function
def detect_objects(image):
    image, shapes = detection_model.preprocess(image)
    pred_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections

# Function to filter OCR text based on region threshold
def filter_text(region, ocr_result, region_threshold):
    rectangle_size = region.shape[0] * region.shape[1]
    plate = []
    for result in ocr_result:
        length = np.sum(np.subtract(result[0][1], result[0][0]))
        height = np.sum(np.subtract(result[0][2], result[0][1]))
        if length * height / rectangle_size > region_threshold:
            plate.append(result[1])
    return plate

# Function to perform OCR on detected regions
def ocr_process(image, detections, detection_threshold, region_threshold):
    scores = detections['detection_scores']
    boxes = detections['detection_boxes'][:len(scores)]
```

```
    width, height = image.shape[1], image.shape[0]
    for id, box in enumerate(boxes):
        roi = box * [height, width, height, width]
        region = image[int(roi[0]):int(roi[2]), int(roi[1]):int(roi[3])]
        reader = easyocr.Reader(['en'])
        ocr_result = reader.readtext(region)
        text = filter_text(region, ocr_result, region_threshold)
        return text, region

# Function to open socket connection and interact with server
def open_socket(ip, port):
    my_socket = socket.socket()
    my_socket.connect((ip, port))
    return my_socket

# Function to send request to server
def send_request(socket, msg):
    socket.send(msg.encode())

# Function to receive slot data from server
def receive_slot_data(socket):
    msg = '1'
    socket.send(msg.encode())
    msg = socket.recv(1024).decode()
    return msg.split()

# Function to check parking availability based on vehicle length
def check_availability(length):
    ToC = 0 if length <= 12 else 1
    slots = receive_slot_data(my_socket)
    if ToC == 0:
        if slots[0] == '1':
            print("Please park at S1")
            send_request(my_socket, '0')
            return True
        elif slots[1] == '1':
            print("No small slots available, please park at L1")
            send_request(my_socket, '0')
            return True
        else:
            print("No parking space available")
            return False
    else:
        if slots[1] == '1':
            print("Please park at L1")
            send_request(my_socket, '0')
            return True
        else:
            print("No parking space available")
            return False
```

```
# Function to find the length of the vehicle
def find_car_length():
    length_of_car = 0
    img_response = urllib.request.urlopen('http://192.168.4.1/capture?')
    frame = cv2.imdecode(np.array(bytearray(img_response.read()),
dtype=np.uint8), -1)
    return length_of_car

# Main loop for object detection and processing
while True:
    img_response = urllib.request.urlopen('http://192.168.4.4/capture?')
    frame = cv2.imdecode(np.array(bytearray(img_response.read()),
dtype=np.uint8), -1)
    detections = detect_objects(frame)

    try:
        now = datetime.now()
        text, region = ocr_process(frame, detections, detection_threshold,
region_threshold)
        length = find_car_length()
        print("Length of the vehicle is:", length)
        check_availability(length)
        save_results(text, region, now.strftime("%d/%m/%Y %H:%M:%S"),
'detection_results.csv', 'Detection_Images')
    except Exception as e:
        print("Error:", e)
        pass

    cv2.imshow('object detection', cv2.resize(frame, (800, 600)))

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cv2.destroyAllWindows()
        break
```

Appendix 3

Stereovision Code

3.1 Gather Calibration Images

```
import cv2
import urllib.request
import numpy as np

image_cnt = 0
while True:

    image_response1 = urllib.request.urlopen('http://192.168.4.1/capture?')
    frame1 = np.array(bytearray(image_response1.read()), dtype=np.uint8)
    img1 = cv2.imdecode(frame1, -1)

    image_response2 = urllib.request.urlopen('http://192.168.4.2/capture?')
    frame2 = np.array(bytearray(image_response2.read()), dtype=np.uint8)
    img2 = cv2.imdecode(frame2, -1)

    key = cv2.waitKey(5)

    if key == ‘q’:  # ESC key
        break
    elif key == ord('s'):
        cv2.imwrite('images/stL/imgL' + str(image_count) + '.png', img1)
        cv2.imwrite('images/stR/imgR' + str(image_count) + '.png', img2)
        image_count += 1

    cv2.imshow('Image 1', img1)
    cv2.imshow('Image 2', img2)
```

3.2 Calibrating the camera

```
import numpy as np
import cv2 as cv
import glob
import os

def find_corners(img, chessboard_size):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCorners(gray, chessboard_size, None)
    if ret:
        corners = cv.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
    return ret, corners

def calibrate_camera(objpoints, imgpoints, img_shape):
    ret, camera_matrix, dist_coeff, rvecs, tvecs = cv.calibrateCamera(objpoints,
imgpoints, img_shape, None, None)
    new_camera_matrix, _ = cv.getOptimalNewCameraMatrix(camera_matrix,
dist_coeff, img_shape[::-1], 1, img_shape[::-1])
    return ret, new_camera_matrix, dist_coeff

def stereo_calibrate(objpoints, imgpoints_l, imgpoints_r, img_shape,
criteria_stereo, flags):
    ret, new_camera_matrix_l, dist_l, new_camera_matrix_r, dist_r, rot, trans,
essential_matrix, fundamental_matrix = cv.stereoCalibrate(objpoints,
imgpoints_l, imgpoints_r, new_camera_matrix_l, dist_l, new_camera_matrix_r,
dist_r, img_shape, criteria_stereo, flags)
    return ret, new_camera_matrix_l, dist_l, new_camera_matrix_r, dist_r, rot,
trans

def stereo_rectify(camera_matrix_l, dist_l, camera_matrix_r, dist_r, img_shape,
rot, trans, rectify_scale):
    rect_l, rect_r, proj_matrix_l, proj_matrix_r, q, roi_l, roi_r =
cv.stereoRectify(camera_matrix_l, dist_l, camera_matrix_r, dist_r, img_shape,
rot, trans, rectify_scale, (0,0))
    return rect_l, rect_r, proj_matrix_l, proj_matrix_r

def save_stereo_map(stereo_map_l, stereo_map_r):
    cv_file = cv.FileStorage('stereoMap.xml', cv.FILE_STORAGE_WRITE)
    cv_file.write('stereoMapL_x', stereo_map_l[0])
    cv_file.write('stereoMapL_y', stereo_map_l[1])
    cv_file.write('stereoMapR_x', stereo_map_r[0])
    cv_file.write('stereoMapR_y', stereo_map_r[1])
    cv_file.release()

chessboard_size = (8, 6)
frame_size = (240, 320)
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER,
30, 0.001)
objp = np.zeros((chessboard_size[0] * chessboard_size[1], 3), np.float32)
```

```
objp[:, :2] = np.mgrid[0:chessboard_size[0], 0:chessboard_size[1]].T.reshape(-1,
2)
objpoints = []
imgpoints_l = []
imgpoints_r = []

images_left = glob.glob('images/stereoLeft/*.png')
images_right = glob.glob('images/stereoRight/*.png')

cnt = 0
for img_left, img_right in zip(images_left, images_right):
    img_l = cv.imread(img_left)
    img_r = cv.imread(img_right)
    ret_l, corners_l = find_chessboard_corners(img_l, chessboard_size)
    ret_r, corners_r = find_chessboard_corners(img_r, chessboard_size)
    if ret_l and ret_r:
        objpoints.append(objp)
        imgpoints_l.append(corners_l)
        imgpoints_r.append(corners_r)
        cv.drawChessboardCorners(img_l, chessboard_size, corners_l, ret_l)
        cv.imshow('img left', img_l)
        cv.drawChessboardCorners(img_r, chessboard_size, corners_r, ret_r)
        cv.imshow('img right', img_r)
        os.chdir('D:/ANPR1/stereoVision')
        cv.imwrite(str(cnt)+"L.png", img_l)
        cv.imwrite(str(cnt)+"R.png", img_r)
        os.chdir('D:/ANPR1/stereoVision')
        cnt += 1
        cv.waitKey(1000)

cv.destroyAllWindows()

ret_calibrate_l, new_camera_matrix_l, dist_l = calibrate_camera(objpoints,
imgpoints_l, img_l.shape[:2])
ret_calibrate_r, new_camera_matrix_r, dist_r = calibrate_camera(objpoints,
imgpoints_r, img_r.shape[:2])

flags = cv.CALIB_FIX_INTRINSIC
crit_stereo = (cv.TERM_CRITERIA_EPS +
cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
ret_stereo, new_camera_matrix_l, dist_l, new_camera_matrix_r, dist_r, rot, trans
= stereo_calibrate(objpoints, imgpoints_l, imgpoints_r, img_l.shape[:2],
criteria_stereo, flags)

rect_l, rect_r, proj_matrix_l, proj_matrix_r =
stereo_rectify(new_camera_matrix_l, dist_l, new_camera_matrix_r, dist_r,
img_l.shape[:2], rot, trans, rectify_scale=1)

stereo_map_l = cv.initUndistortRectifyMap(new_camera_matrix_l, dist_l, rect_l,
proj_matrix_l, img_l.shape[:2], cv.CV_16SC2)
```

```
stereo_map_r = cv.initUndistortRectifyMap(new_camera_matrix_r, dist_r,
rect_r, proj_matrix_r, img_r.shape[:2], cv.CV_16SC2)

print("Saving parameters!")
save_stereo_map(stereo_map_l, stereo_map_r)
```

3.3 Dimension Calculation

```
import sys
import cv2
import numpy as np
import time
import imutils
from matplotlib import pyplot as plt

import triangulation as tri
import calibration
import urllib.request
import torch
import math

from ultralytics import YOLO

model = YOLO('yolov8n.pt')

frame_rate = 120
B = 3.5 #distance between the cameras in cm
f = 4.8 #camera focal length mm
alpha = 65
pt1 = torch.empty(0, 4, dtype=torch.int64)



def giveLen(img, boundbox, angle, depth):
    print(img.shape)
    width = img.shape[1]
    height = img.shape[0]

    ratio = angle/width;

    centerOfFrame = width/2

    leftpix = centerOfFrame - boundbox[0]
    rightpix = boundbox[2] - centerOfFrame

#     print([boundbox[0], boundbox[1], centerOfFrame])

    leftang = ratio*leftpix
```

```
    rightang = ratio*rightpix

    leftlen = math.tan((leftang * math.pi)/180)*depth
    rightlen = math.tan((rightang * math.pi) / 180) * depth

#     if(boundbox[1] < centerOfFrame && boundbox[2] < centerOfFrame)

    return round(leftlen+rightlen)




def findCarLen():

    imgResponse = urllib.request.urlopen('http://192.168.4.1/capture?')
    frame = np.array(bytearray(imgResponse.read()), dtype = np.uint8)
    frame_left = cv2.imdecode(frame, -1)
    print(frame_left.shape)
    recFrame = frame_left

    imgResponse = urllib.request.urlopen('http://192.168.4.2/capture?')
    frame2 = np.array(bytearray(imgResponse.read()), dtype = np.uint8)
    frame_right = cv2.imdecode(frame2, -1)

    frame_right, frame_left = calibration.undistortRectify(frame_right, frame_left)


#     frame_right = cv2.cvtColor(frame_right, cv2.COLOR_BGR2RGB)
#     frame_left = cv2.cvtColor(frame_left, cv2.COLOR_BGR2RGB)


    results_right = model(frame_right, classes = 2)
    results_left = model(frame_left, classes = 2)


#     frame_right = cv2.cvtColor(frame_right, cv2.COLOR_RGB2BGR)
#     frame_left = cv2.cvtColor(frame_left, cv2.COLOR_RGB2BGR)


    center_right = 0
    center_left = 0

    if np.shape(results_right[0].boxes.xyxy) != np.shape(pt1):
        for result in results_right:
            frame_right = result.plot()
            h, w, c = frame_right.shape
            boundBox = result.boxes[0].xyxy[0].numpy()
            center_point_right = ((boundBox[0] + boundBox[2]) / 2, (boundBox[1] +
boundBox[3]) / 2)
```

```
    if np.shape(results_left[0].boxes.xyxy) != np.shape(pt1):
        for result in results_left:
            frame_left = result.plot()
            h, w, c = frame_left.shape
            boundBox = result.boxes[0].xyxy[0].numpy()
            center_point_left = ((boundBox[0] + boundBox[2]) / 2, (boundBox[1] + boundBox[3]) / 2)


    if np.shape(results_right[0].boxes.xyxy) != np.shape(pt1) and np.shape(results_left[0].boxes.xyxy) != np.shape(pt1):
        depth = tri.find_depth(center_point_right, center_point_left, frame_right, frame_left, B, f, alpha)
        x.append(depth)

        #getting the length of the car

        lenResults = model(recFrame, classes = 2)

        lengthOfCar = giveLen(recFrame, lenResults[0].boxes[0].xyxy[0].numpy(), alpha, depth)
        y.append(lengthOfCar)



    cv2.imshow("frame right", frame_right)
    cv2.imshow("frame left", frame_left)
```

Appendix 4

Final Parking System Code

```
import sys
import cv2
import numpy as np
import time
import imutils
from matplotlib import pyplot as plt
import math

import triangulation as tri
import calibration
import urllib.request
import torch

from ultralytics import YOLO

model = YOLO('yolov8n.pt')

x = []
y = []

frame_rate = 120
B = 3.5 #distance between the cameras in cm
f = 4.8 #camera focal length mm
alpha = 65
pt1 = torch.empty(0, 4, dtype=torch.int64)



def giveLen(img, boundbox, angle, depth):
    width = img.shape[1]
    height = img.shape[0]

    ratio = angle/width;

    centerOfFrame = width/2

    leftpix = centerOfFrame - boundbox[0]
    rightpix = boundbox[2] - centerOfFrame

    leftang = ratio*leftpix
    rightang = ratio*rightpix

    leftlen = math.tan((leftang * math.pi)/180)*depth
```

```
    rightlen = math.tan((rightang * math.pi) / 180) * depth

    return round(leftlen+rightlen)




def findCarLen():
    lengthOfCar = 0
    imgResponse = urllib.request.urlopen('http://192.168.4.1/capture?')
    frame = np.array(bytearray(imgResponse.read()), dtype = np.uint8)
    frame_left = cv2.imdecode(frame, -1)
    print(frame_left.shape)
    recFrame = frame_left

    imgResponse = urllib.request.urlopen('http://192.168.4.2/capture?')
    frame2 = np.array(bytearray(imgResponse.read()), dtype = np.uint8)
    frame_right = cv2.imdecode(frame2, -1)

    frame_right, frame_left = calibration.undistortRectify(frame_right, frame_left)


#     frame_right = cv2.cvtColor(frame_right, cv2.COLOR_BGR2RGB)
#     frame_left = cv2.cvtColor(frame_left, cv2.COLOR_BGR2RGB)


    results_right = model(frame_right, classes = 2, verbose=False)
    results_left = model(frame_left, classes = 2, verbose=False)


#     frame_right = cv2.cvtColor(frame_right, cv2.COLOR_RGB2BGR)
#     frame_left = cv2.cvtColor(frame_left, cv2.COLOR_RGB2BGR)


    center_right = 0
    center_left = 0

    if np.shape(results_right[0].boxes.xyxy) != np.shape(pt1):
        for result in results_right:
            frame_right = result.plot()
            h, w, c = frame_right.shape
            boundBox = result.boxes[0].xyxy[0].numpy()
            center_point_right = ((boundBox[0] + boundBox[2]) / 2, (boundBox[1] +
boundBox[3]) / 2)


    if np.shape(results_left[0].boxes.xyxy) != np.shape(pt1):
        for result in results_left:
            frame_left = result.plot()
```

```
        h, w, c = frame_left.shape
        boundBox = result.boxes[0].xyxy[0].numpy()
        center_point_left = ((boundBox[0] + boundBox[2]) / 2, (boundBox[1] + boundBox[3]) / 2)


    if np.shape(results_right[0].boxes.xyxy) != np.shape(pt1) and np.shape(results_left[0].boxes.xyxy) != np.shape(pt1):
        depth = tri.find_depth(center_point_right, center_point_left, frame_right, frame_left, B, f, alpha)
        x.append(depth)

        #getting the length of the car

        lenResults = model(recFrame, classes = 2, verbose=False)

        lengthOfCar = giveLen(recFrame, lenResults[0].boxes[0].xyxy[0].numpy(), alpha, depth)
        y.append(lengthOfCar)


    return lengthOfCar
#     cv2.imshow("frame right", frame_right)
#     cv2.imshow("frame left", frame_left)

while True:


    imgResponse = urllib.request.urlopen('http://192.168.4.3/capture?')
    frame = np.array(bytearray(imgResponse.read()), dtype = np.uint8)
    image_np = cv2.imdecode(frame, -1)


    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
```

```
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)


    try:
        now = datetime.now()
        text, region = ocr_it(image_np_with_detections, detections, detection_threshold, region_threshold)
        length = findCarLen()
        print(length)
        save_results(text, region, now.strftime("%d/%m/%Y %H:%M:%S") , 'detection_results.csv', 'Detection_Images')
        write_read()

    except:
        #print("hello")
        pass

    cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))



    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```