SMART TRAFFIC MANAGEMENT

A PROJECT REPORT

Submitted by

RANJANI M **(711719104075)**

SELLAMANI S **(711719104087)**

SHIFA RUKSHANA B **(711719104090)**

KIRANA S **(711719104304)**

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KGiSL INSTITUTE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY :: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"SMART TRAFFIC MANAGEMENT"** is the bonafide work of **"RANJANI M, SELLAMANI S, SHIFA RUKSHANA B, KIRANA S"** who carried out the project work under my supervision.

SIGNATURE
DR. THENMOZHI T
HEAD OF THE DEPARTMENT
PROFESSOR
Computer Science and Engineering
KGiSL Institute of Technology
Coimbatore - 641035

SIGNATURE
MS. LATHIKA B.A
SUPERVISOR
ASSISTANT PROFESSOR
Computer Science and Engineering
KGiSL Institute of Technology
Coimbatore – 641035

Submitted for the Anna University Project Viva-Voice examination conducted on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our deepest gratitude to our **Chairman and Managing Director Dr. Ashok Bakthavachalam f**or providing us with an environment to complete our project successfully.

We are grateful to our CEO-Academic Initiatives **Mr. Aravind Kumar Rajendran,** our beloved Director-Academics **Dr. P. Shankar** and our honourable Secretary **Dr. N. Rajkumar**. We are thankful to our beloved Principal **Dr. M. Selvam** and Vice-Principal **Dr. S. Suresh Kumar** for their support and their valuable guidance and blessings.

We would like to thank **Dr. T. Thenmozhi, M.E., M.B.A., Ph.D.,** Head of the Department, Department of Computer Science and Engineering for his unwavering support during the entire course of this project work. We express our sincere thanks to **Ms. ARUNA T N M.E.,** our Project Coordinator, Professor, Department of Computer Science and Engineering who modelled us both technically and morally for achieving greater success in completing this project work.

We express our sincere thanks to our faculty guide **Ms. LATHIKA B.A.,** Assistant Professor, Department of Computer Science and Engineering for her constant encouragement and support throughout our course, especially for the useful suggestions given during the course of the project period and being instrumental in the completion of our project with her complete guidance.

We also thank all the Faculty Members of our department and finally, we take this opportunity to extend our deep appreciation to our Family and Friends, for all they meant to us during the crucial times of the completion of our project.

ABSTRACT

Traffic congestion is a major problem in metropolitan areas of India, and traditional approaches to managing traffic have proven ineffective as vehicle density increases. This project aims to address this issue by implementing a real-time signal monitoring and handling approach using the YOLOv3 algorithm. YOLOv3 is a real-time object detection algorithm that accurately identifies vehicles in videos, live feeds, or images. By counting the number of vehicles on each side of the road and dynamically adjusting signal switching based on this count, the project aims to improve traffic management. The vehicle count is obtained using ESP32 and LED lights are used to toggle the signal. The switching time of the signal is determined based on real-time image detection, providing accurate results even in dense traffic. The project involves comparing object counts from different cameras to perform signal switching dynamically.

Deep learning has emerged as a powerful tool for optimizing traffic signal switching and has a significant impact on traffic flow and safety in urban transportation management. This project provides an abstract of the latest research in this field, focusing on deep learning models such as convolutional neural networks (CNNs) and YOLO networks for predicting traffic patterns. Additionally, the project explores the use of reinforcement learning to optimize signal switching policies based on real-time traffic data. The results demonstrate that deep learning-based smart traffic management can effectively reduce traffic congestion and improve road safety. This approach has the potential to revolutionize urban transportation management, but further research is required to fully evaluate its effectiveness and address implementation challenges.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

YOLO	You Only Look Once
OLED	Organic Light-Emitting Diode
ESP32	Espressif System Platform 32
LED	Light Emitting Diode
CNN	Convolutional Neural Network
DEEP SORT	Deep Simple Online Realtime Tracking
RNN	Recurrent Neural Network
OV2640	Omni Vision 2640
COCO	Common Object in Context

CHAPTER 1

INTRODUCTION

Managing traffic efficiently and effectively has become an increasingly challenging task in modern urban areas. The rise in vehicle density and the complexity of traffic patterns have rendered traditional approaches to traffic signal switching inadequate. However, the emergence of smart traffic management systems, empowered by advanced technologies such as deep learning, offers promising solutions to address these challenges.

Smart traffic management encompasses a range of innovative techniques and technologies that leverage real-time data and intelligent algorithms to optimize traffic flow, reduce congestion, and enhance road safety. One key aspect of smart traffic management is the utilization of deep learning algorithms to make informed decisions regarding signal timings and traffic control.

Deep learning algorithms have demonstrated remarkable capabilities in learning complex patterns and making accurate predictions based on large datasets. By leveraging these algorithms, it becomes possible to analyze real-time traffic data, including vehicle counts, speeds, and travel times, to predict traffic conditions and dynamically adjust signal timings accordingly.

This project aims to explore the potential of deep learning-based smart traffic management systems. It focuses on utilizing advanced deep learning models, such as convolutional neural networks (CNNs) and You Only Look Once (YOLO) networks, to predict traffic patterns accurately. These models excel in object detection tasks, enabling the identification and counting of vehicles in real-time videos, live feeds, or images.

By deploying the YOLOv3 algorithm, this project seeks to precisely count vehicles on different sides of the road and make informed decisions regarding signal switching. The vehicle count information is fed into an ESP32 platform, which enables real-time vehicle counting, and the resulting data is used to control LED lights that toggle the traffic signals accordingly. The switching time of the signals is determined based on the detection of real-time images, ensuring timely adjustments even in densely congested traffic scenarios.

The integration of deep learning into smart traffic management has the potential to revolutionize urban transportation systems. By accurately predicting traffic patterns and dynamically optimizing signal timings, these systems can significantly improve traffic flow, reduce congestion, and enhance overall road safety. However, further research is needed to evaluate the effectiveness of these techniques in real-world scenarios and address the practical challenges of implementing deep learning-based smart traffic management systems.

In this project, we will delve into the latest advancements in deep learning for smart traffic management, exploring various models and algorithms used for predicting traffic patterns. Furthermore, we will investigate the potential of reinforcement learning techniques to optimize signal switching policies based on real-time traffic data. The results of this research will shed light on the effectiveness of deep learning-based smart traffic management and provide insights into its potential to transform urban transportation management practices.

1.1 OBJECTIVE OF THE PROJECT

The primary objective of this project is to develop a smart traffic management system that utilizes deep learning algorithms to optimize traffic flow and improve road safety. Specifically, the project aims to achieve the following objectives:

- Implement Real-Time Vehicle Detection: Utilize the YOLOv3 algorithm, a real-time object detection algorithm, to accurately identify and count vehicles in real-time videos, live feeds, or images. This will enable the system to gather precise data on vehicle density and distribution on different sides of the road.
- Dynamic Signal Switching: Develop an intelligent signal switching mechanism that dynamically adjusts signal timings based on the real-time vehicle count information obtained from the deep learning algorithm. By adapting the signal timings to the actual traffic conditions, the system aims to optimize traffic flow and minimize congestion.
- Evaluation and Comparison: Compare the object counts obtained from different cameras to perform signal switching dynamically. This evaluation will enable the system to adapt its decision-making process based on the specific traffic conditions observed in different areas, ensuring efficient traffic management.
- Assess Traffic Pattern Prediction: Explore the potential of deep learning models, such as convolutional neural networks (CNNs), to predict traffic patterns accurately. By analyzing historical and real-time traffic data, the system aims to anticipate congestion patterns and adjust signal timings proactively, thereby improving traffic flow and reducing congestion.
- Enhance Road Safety: Evaluate the impact of the smart traffic management system on road safety. By optimizing signal timings based on accurate vehicle counts and traffic predictions, the system aims to reduce the likelihood of accidents and enhance overall road safety.

- Feasibility and Practicality Assessment: Assess the feasibility and practicality of implementing deep learning-based smart traffic management systems in real-world scenarios. Identify potential challenges and limitations of the proposed system and provide recommendations for further improvements and future research in the field.

By accomplishing these objectives, the project aims to demonstrate the effectiveness of deep learning-based smart traffic management in reducing traffic congestion, improving road safety, and revolutionizing urban transportation management practices.

1.2 SCOPE OF THE PROJECT

The scope of this project encompasses the development and implementation of a smart traffic management system that leverages deep learning algorithms for optimizing traffic flow and improving road safety. The project focuses on the following key areas:

- Deep Learning Models: The project involves the utilization of advanced deep learning models, such as convolutional neural networks (CNNs) and You Only Look Once (YOLO) networks, for accurate vehicle detection and counting. These models will be trained and deployed to analyze real-time traffic data and provide insights into traffic patterns.
- Real-Time Signal Switching: The project aims to develop an intelligent signal switching mechanism that dynamically adjusts signal timings based on the real-time vehicle count information obtained from the deep learning algorithms. The system will toggle the traffic signals accordingly to optimize traffic flow and minimize congestion.
- Hardware Integration: The project involves integrating the deep learning algorithms with the ESP32 platform, enabling real-time vehicle counting and data processing. The ESP32 will be responsible for capturing traffic data, communicating with the deep learning models, and controlling the LED lights for signal switching.
- Evaluation and Comparison: The project will evaluate and compare the object counts obtained from different cameras to perform signal switching dynamically. This evaluation will allow the system to adapt its decision-making process based on specific traffic conditions observed in different areas, enhancing the efficiency of traffic management.
- Performance Assessment: The project aims to assess the performance of the smart traffic management system in terms of traffic flow optimization and road safety

enhancement. The system's effectiveness in reducing congestion and minimizing the risk of accidents will be evaluated based on real-world data and scenarios.

- Feasibility Analysis: The project will analyze the feasibility and practicality of implementing deep learning-based smart traffic management systems in real-world environments. Factors such as computational requirements, data availability, and scalability will be considered to determine the system's viability for wider deployment.

The scope of the project does not include physical infrastructure modifications or installation of additional sensors. It focuses primarily on the software and algorithmic aspects of smart traffic management. However, the project will consider the integration of the developed system with existing traffic infrastructure for a comprehensive evaluation.

The outcomes of this project will provide valuable insights into the effectiveness and potential of deep learning-based smart traffic management systems. The findings can serve as a foundation for further research and development in the field of intelligent transportation systems, with the ultimate goal of improving traffic efficiency, reducing congestion, and enhancing road safety in urban areas.

CHAPTER 2

LITERATURE REVIEW

2.1 OVERVIEW

The related works in the field of deep learning and smart traffic management focus on optimizing traffic signal control, predicting traffic flow patterns, and improving vehicle detection and recognition. These studies leverage advanced deep learning models such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and reinforcement learning algorithms to address the challenges of traffic congestion and enhance traffic management systems. The findings emphasize the potential of deep learning in improving traffic flow, reducing congestion, and enhancing road safety. By utilizing real-time traffic data and historical information, these works contribute to the development of intelligent transportation systems that can dynamically adjust signal timings, predict traffic patterns, and optimize traffic management.

2.2 LITERATURE REVIEW

"Traffic Signal Control Using Deep Reinforcement Learning" by Wei Dai et al. (2019):
This study proposes a deep reinforcement learning framework for optimizing traffic signal control. The researchers trained a deep Q-network (DQN) to learn the optimal signal timings by considering traffic conditions and historical data. The results demonstrated improved traffic flow and reduced congestion.

"Real-Time Traffic Flow Prediction Using Convolutional Neural Networks" by Zhiyong Cui et al. (2018):
The researchers developed a CNN-based model to predict traffic flow patterns in real-time. By analyzing historical traffic data and considering external factors, such as weather and time of day, the model accurately predicted traffic flow. The predictions were used to optimize signal timings and improve traffic management.

"Intelligent Traffic Management System Based on Deep Learning" by Xuelei Han et al. (2020):

This study proposed an intelligent traffic management system that integrated deep learning techniques. They utilized YOLOv3 for real-time vehicle detection and developed a traffic signal control algorithm based on the detected vehicle counts. The system demonstrated improved traffic flow and reduced delays.

"Smart Traffic Light Control System Using Deep Learning" by Mohamad Badrul Anuar et al. (2020):

The researchers proposed a smart traffic light control system that employed a deep learning model for real-time vehicle detection. The system utilized the detected vehicle counts to optimize signal timings and prioritize traffic flow. Experimental results showed reduced waiting times and improved traffic efficiency.

"Deep Learning-Based Traffic Signal Control for Smart Cities" by Hongkyu Park et al. (2018):

This study focused on optimizing traffic signal control using deep learning in smart cities. They developed a deep Q-network (DQN) algorithm to learn optimal signal timings based on traffic data. The system demonstrated improved traffic flow and reduced travel times in real-world scenarios.

"Traffic Flow Prediction Using Long Short-Term Memory Recurrent Neural Networks" by Yusuke Sugano et al. (2018):

This study proposes the use of long short-term memory (LSTM) recurrent neural networks to predict traffic flow patterns. By analyzing historical traffic data, the model accurately predicted future traffic conditions. The predictions were utilized for optimizing traffic signal timings and managing traffic flow.

"A Survey of Deep Learning Approaches for Traffic Sign Recognition" by Fahad A. Al-Atabani et al. (2020):

This survey paper provides an overview of various deep learning approaches for traffic sign recognition. It discusses the use of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) in detecting and classifying traffic signs. Effective traffic sign recognition is crucial for intelligent traffic management systems.

"Smart Traffic Control System Based on Deep Reinforcement Learning and Image Processing" by Sumitra Devi Kanhare et al. (2020):

The researchers propose a smart traffic control system that combines deep reinforcement learning and image processing techniques. The system detects vehicles using image processing, predicts traffic flow using LSTM networks, and optimizes signal timings using deep reinforcement learning. The results showed improved traffic management and reduced congestion.

"Deep Learning-Based Traffic Control for Intelligent Transportation Systems" by Qiang Yu et al. (2018):

This study presents a deep learning-based traffic control system for intelligent transportation systems. The system utilizes a combination of CNNs and RNNs to detect and predict traffic flow. The predicted traffic conditions are then used to optimize signal timings for efficient traffic management.

"Real-Time Traffic Congestion Prediction Using Deep Recurrent Neural Networks" by Haizhong Zheng et al. (2019):

The researchers propose a deep recurrent neural network (DRNN) for real-time traffic congestion prediction. The DRNN model analyzes traffic flow data from multiple sensors and predicts congestion levels. The predictions are utilized to dynamically adjust signal timings and manage traffic congestion.

2.3 SUMMARY

This chapter gives the description about the literature survey of similar work and the methodologies used in each work.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The current traffic management systems in place rely on conventional methods that often prove inadequate in effectively addressing the challenges posed by traffic congestion. These systems typically employ fixed signal timings and simplistic heuristic algorithms to regulate traffic flow. However, such approaches are limited in their ability to adapt to real-time traffic conditions and may result in inefficient traffic management and increased congestion.

Moreover, the existing systems predominantly operate through manual monitoring and control, which can be time-consuming and susceptible to human errors. They often lack comprehensive data collection and analysis capabilities, preventing them from making data-driven decisions for optimizing traffic flow.

While some existing systems may incorporate basic automation features, such as pre-determined signal timings based on historical traffic patterns, they lack the flexibility and adaptability required to effectively handle dynamic traffic scenarios.

The current traffic management systems are constrained by their reliance on outdated techniques and limited technological advancements. They struggle to cope with the complexities of urban traffic, such as increasing vehicle density and unpredictable traffic patterns. To address these challenges, there is a pressing need for advanced systems that leverage cutting-edge technologies like deep learning and intelligent algorithms. These systems have the potential to revolutionize traffic management by providing real-time data analysis, adaptive decision-making, and dynamic control mechanisms that can alleviate congestion and optimize traffic flow more effectively.

In some cases, existing systems may incorporate limited forms of automation, such as pre-programmed signal timings based on historical traffic patterns. However, these approaches do not provide the level of flexibility and adaptability required to effectively manage dynamic traffic conditions.

Overall, the existing traffic management systems lack the sophistication and intelligence offered by deep learning and smart technologies. They often struggle to cope with the increasing vehicle density and complex traffic scenarios in urban areas. Therefore, there is a need for advanced systems that leverage deep learning algorithms, real-time data analysis, and intelligent decision-making to revolutionize traffic management and alleviate congestion-related issues.

3.2 DRAWBACKS

The existing traffic management systems suffer from several drawbacks that hinder their effectiveness in addressing traffic congestion and optimizing traffic flow. Some of the key drawbacks include:

- Limited Adaptability: Conventional systems rely on fixed signal timings and simplistic algorithms that lack the ability to adapt to real-time traffic conditions. This limitation leads to inefficient traffic management, as the systems cannot dynamically adjust signal timings based on the current traffic flow.
- Lack of Real-Time Data Analysis: Existing systems often lack comprehensive real-time data collection and analysis capabilities. They rely on manual monitoring or limited sensor data, which hampers their ability to make informed and data-driven decisions for traffic optimization. This results in suboptimal signal timings and inadequate response to changing traffic patterns.
- Inefficient Resource Allocation: Traditional systems do not effectively allocate resources to areas with higher traffic demands. Since they lack real-time data and advanced analytics, they may not accurately identify congested areas or allocate sufficient green time to alleviate traffic buildup, leading to increased congestion and longer travel times.
- Limited Intelligent Decision-Making: Conventional systems typically lack intelligent decision-making capabilities. They do not utilize advanced algorithms or artificial intelligence techniques to analyze complex traffic patterns, predict future traffic conditions, and optimize signal timings accordingly. This limitation prevents the systems from making proactive and efficient traffic management decisions.
- Susceptibility to Human Errors: Manual monitoring and control in existing systems can introduce human errors, leading to inaccurate data collection, suboptimal signal timings, and compromised traffic management. Human factors such as fatigue, distraction, or subjective judgment can impact the system's performance and reliability.
- Scalability Challenges: As urban areas continue to experience rapid growth and increasing traffic volumes, the scalability of existing systems becomes a concern. Traditional approaches may struggle to handle the complexities and demands of larger traffic networks, resulting in further congestion and degraded traffic management performance.

Addressing these drawbacks is essential to improve the efficiency and effectiveness of traffic management systems. Advanced technologies like deep learning, real-time data analysis, and intelligent algorithms have the potential to overcome these limitations and pave the way for more intelligent and responsive traffic management solutions.

3.3 PROPOSED SYSTEM

The proposed system aims to overcome the limitations of existing traffic management systems by leveraging advanced technologies and intelligent algorithms. The key features and components of the proposed system include:

- Deep Learning-based Traffic Analysis: The system utilizes deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to analyze real-time traffic data. This allows for accurate vehicle detection, tracking, and classification, enabling a more comprehensive understanding of traffic patterns and dynamics.
- Real-Time Data Collection: The proposed system incorporates a network of sensors, including video cameras, vehicle detectors, and other smart devices, to gather real-time data on traffic flow, congestion levels, and other relevant parameters. This continuous data collection ensures up-to-date and accurate information for traffic management decisions.
- Intelligent Traffic Signal Control: The system employs intelligent algorithms to optimize traffic signal control based on the analyzed data. By dynamically adjusting signal timings in response to real-time traffic conditions, the proposed system aims to improve traffic flow, reduce congestion, and minimize waiting times at intersections.
- Adaptive Decision-Making: The system utilizes advanced algorithms, such as reinforcement learning, to make adaptive and data-driven decisions. By learning from past traffic patterns and feedback, the system continuously refines its signal control strategies to achieve optimal traffic management outcomes.
- Integration with Smart City Infrastructure: The proposed system integrates with existing smart city infrastructure, including centralized control systems, traffic management centers, and communication networks. This integration enables seamless coordination and communication between different components of the traffic management system, enhancing its overall efficiency and effectiveness.

- Scalability and Future Expansion: The proposed system is designed to be scalable and adaptable to accommodate the evolving needs of urban areas. It can be expanded to include additional sensors, incorporate real-time data from other sources (such as GPS or social media), and integrate with emerging technologies like connected and autonomous vehicles.

The proposed system aims to revolutionize traffic management by leveraging advanced technologies and intelligent algorithms. By utilizing real-time data analysis, adaptive decision-making, and optimized signal control, it has the potential to significantly improve traffic flow, reduce congestion, and enhance overall transportation efficiency and safety.

3.4 ADVANTAGES

The proposed system offers several advantages over existing traffic management systems, empowering more efficient and effective management of traffic flow. The key advantages include:

- Real-Time Traffic Analysis: By utilizing deep learning techniques and real-time data collection, the proposed system provides accurate and up-to-date traffic analysis. This enables a comprehensive understanding of traffic patterns, congestion hotspots, and traffic dynamics, allowing for more informed decision-making.
- Dynamic Signal Control: Unlike traditional systems with fixed signal timings, the proposed system incorporates intelligent algorithms that dynamically adjust signal timings based on real-time traffic conditions. This adaptive signal control optimizes traffic flow, reduces congestion, and minimizes waiting times at intersections, leading to smoother and more efficient traffic movement.
- Data-Driven Decision-Making: The proposed system leverages advanced algorithms, such as reinforcement learning, to make data-driven decisions for traffic management. By learning from real-time traffic data and historical patterns, the system continuously improves its decision-making capabilities, optimizing signal control strategies and enhancing overall traffic management effectiveness.
- Improved Road Safety: With its ability to detect and track vehicles accurately, the proposed system enhances road safety. It can identify potential hazards, such as reckless driving or traffic violations, and trigger appropriate interventions, such as adjusting signal timings or alerting traffic authorities. This proactive approach contributes to a safer driving environment for both motorists and pedestrians.

- Scalability and Integration: The proposed system is designed to be scalable and adaptable to accommodate the evolving needs of urban areas. It seamlessly integrates with existing smart city infrastructure, allowing for centralized control and coordination with other transportation systems. This scalability and integration facilitate a holistic approach to traffic management and pave the way for future expansion and integration with emerging technologies.
- Reduced Congestion and Travel Times: By optimizing signal control and dynamically responding to traffic conditions, the proposed system significantly reduces congestion and travel times. Smoother traffic flow leads to shorter delays, improved commute experiences, and increased overall efficiency in urban transportation.
- Enhanced Sustainability: The optimized traffic flow achieved by the proposed system results in reduced fuel consumption and greenhouse gas emissions. By minimizing idle time and congestion, the system contributes to a more sustainable and environmentally friendly transportation ecosystem.

The advantages offered by the proposed system demonstrate its potential to transform traffic management by leveraging advanced technologies and intelligent algorithms. The system's real-time analysis, dynamic signal control, data-driven decision-making, and scalability contribute to more efficient traffic flow, improved road safety, and enhanced overall transportation systems.

CHAPTER 4

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

MICRO CONTROLLER	ESP 32 CAM
DISPLAY	OLED DISPLAY
CAMERA	OV2640
SYSTEM PROCESSOR	INTEL I3
RAM	4 GB
DISK SPACE	20 GB
MONITOR	15” 0r 17” colour monitor
MOUSE	Scroll or Optical mouse or Touchpad
KEYBOARD	Standard 110 keys keyboard

4.2 SOFTWARE REQUIREMENTS

PROGRAMMING LANGUAGE	C++ & PYTHON
OPERATING SYSTEM	WINDOWS, MAC OS & LINUX
INTEGRATED DEVELOPMENT ENVIRONMENT	ARDUINO & VISUAL STUDIO
ENVIRONMENT MANAGEMENT	ANACONDA

CHAPTER 5

SYSTEM DESCRIPTION

5.1 C PLUS PLUS

C++ is a powerful and versatile programming language that is widely used for developing a wide range of applications, including system software, games, embedded systems, and high-performance applications. It is an extension of the C programming language with added features and capabilities.

C++ is known for its efficiency, performance, and ability to work close to the hardware, making it a popular choice for applications that require speed and low-level control. It supports both procedural and object-oriented programming paradigms, allowing developers to write modular and reusable code.

One of the key features of C++ is its support for classes and objects, which enables the use of object-oriented programming concepts such as encapsulation, inheritance, and polymorphism. This allows for the creation of complex software systems with organized and maintainable code.

C++ also provides extensive support for generic programming through templates. Templates allow developers to write generic algorithms and data structures that can work with different data types, providing flexibility and code reusability.

Memory management is a crucial aspect of C++. It offers both stack allocation and dynamic memory allocation through features like pointers and the new and delete operators. However, managing memory manually in C++ requires careful attention to avoid memory leaks and other memory-related issues.

In terms of standard libraries, C++ offers a rich set of libraries that provide functionalities for various purposes, including input/output operations, string handling, data structures, algorithms, and concurrency. The Standard Template Library (STL) is a notable component of the C++ standard library, offering generic algorithms and data structures.

Overall, C++ is a versatile and powerful programming language that combines high performance, low-level control, and support for object-oriented and generic programming paradigms. Its flexibility and efficiency make it a popular choice for a wide range of applications, especially those that require close-to-hardware programming and high-performance computing.

5.2 PYTHON

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. It was created with the goal of providing a clear and expressive syntax that is easy to understand and write, making it a popular choice among beginners and experienced developers alike.

Python emphasizes code readability through the use of indentation and a minimalist syntax. This readability enables developers to write clean, concise, and maintainable code, reducing the chances of errors and improving overall productivity.

One of the key strengths of Python is its extensive standard library, which provides a wide range of modules and functionalities for tasks such as file handling, networking, web development, data manipulation, and more. The standard library, combined with a vast ecosystem of third-party libraries and frameworks, makes Python suitable for a wide range of applications, from web development to scientific computing.

Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It allows developers to choose the approach that best suits their needs and promotes code reuse and modularity through the use of classes, modules, and libraries.

Python is an interpreted language, meaning that it does not require explicit compilation before running. This enables rapid development and prototyping as code changes can be immediately tested and executed without the need for a lengthy compilation process.

Python has gained popularity in the field of data science and machine learning due to its rich ecosystem of libraries such as NumPy, Pandas, and TensorFlow. These libraries provide powerful tools for data manipulation, analysis, and modelling, making Python a go-to choice for data-driven applications.

Furthermore, Python's versatility extends to its cross-platform compatibility. It runs on various operating systems, including Windows, macOS, and Linux, allowing developers to write code that can be easily deployed on different platforms without major modifications.

Python's popularity and active community contribute to its extensive documentation, tutorials, and support resources. The Python community is known for its inclusiveness and helpfulness, making it easy for developers to seek assistance and collaborate with others.

In summary, Python is a versatile and beginner-friendly programming language that combines readability with a vast ecosystem of libraries and frameworks. Its simplicity, versatility, and cross-platform compatibility make it an excellent choice for a wide range of applications, from web development and scripting to data science and machine learning.

5.3 ARDUINO

The Arduino IDE (Integrated Development Environment) is a software tool specifically designed for programming Arduino microcontrollers. It provides a user-friendly interface and a simplified programming environment to write, compile, and upload code to Arduino boards. The Arduino IDE supports the Arduino programming language, which is based on a subset of C and C++. It offers a simplified syntax and a set of libraries that make it easy for beginners to get started with programming and electronics.

One of the key features of the Arduino IDE is its simplicity and ease of use. It provides a straightforward interface with a code editor, a message console, and buttons for compiling and uploading code to the Arduino board. This simplicity allows even those with little to no programming experience to quickly start working with Arduino.

The IDE includes a rich set of libraries that provide pre-written code for common tasks and components, such as controlling LEDs, reading sensors, and communicating with other devices. These libraries simplify the process of interacting with hardware and enable rapid prototyping and development of projects.

The Arduino IDE also offers a built-in Serial Monitor, which allows developers to communicate with the Arduino board and receive real-time data or debug messages. This feature is particularly useful for troubleshooting and verifying the behaviour of the code during runtime.

Furthermore, the Arduino IDE supports a wide range of Arduino boards, including the popular Arduino Uno, Arduino Mega, and Arduino Nano, among others. It provides board-specific configurations and libraries, ensuring compatibility and ease of use for different Arduino models.

The Arduino IDE is open-source software, which means that the source code is freely available and can be modified and customized according to specific requirements. This flexibility allows developers to extend the functionality of the IDE and integrate it with other tools and platforms.

In summary, the Arduino IDE is a user-friendly programming environment designed for Arduino microcontrollers. It offers a simplified syntax, a rich set of libraries, and an intuitive interface, making it accessible to beginners and experienced developers alike. With its ease of use and extensive community support, the Arduino IDE is a powerful tool for prototyping and developing various electronics projects.

5.4 VISUAL STUDIO

Visual Studio is a powerful and comprehensive integrated development environment (IDE) developed by Microsoft. It provides a wide range of tools and features that enable developers to build applications for various platforms, including Windows, web, mobile, cloud, and more.
One of the key strengths of Visual Studio is its versatility. It supports multiple programming languages, including C#, C++, Python, JavaScript, and many others. This allows developers to work with their preferred programming language and leverage the extensive features and tools provided by Visual Studio.
Visual Studio offers a rich and intuitive user interface, making it easy to navigate and work with different project types. It provides a code editor with features like syntax highlighting, code completion, and debugging capabilities, which help developers write clean and error-free code.
The IDE includes a powerful debugging system that allows developers to step through their code, set breakpoints, and analyze variables and data in real-time. This helps in identifying and fixing issues during the development process, improving the overall quality of the application.
Visual Studio integrates with various version control systems such as Git, allowing developers to manage their source code efficiently and collaborate with team members seamlessly. It provides built-in tools for branching, merging, and resolving conflicts, simplifying the code management process.
Visual Studio also supports cloud development and offers tools for deploying and managing applications on platforms like Microsoft Azure. It provides seamless integration with Azure services and enables developers to take advantage of cloud scalability, storage, and analytics capabilities.
Additionally, Visual Studio provides powerful tools for performance profiling, unit testing, and code analysis, helping developers optimize their applications and ensure code quality.
Developers can find a wealth of resources, forums, and online communities where they can seek help, share knowledge, and collaborate with other developers.
In summary, Visual Studio is a feature-rich and versatile IDE that supports multiple programming languages and platforms. It offers a wide range of tools, frameworks, and extensions to streamline development workflows, enhance productivity, and build high-quality applications. Whether you are developing desktop applications, web services, mobile apps, or cloud solutions, Visual Studio provides a comprehensive set of features to support your development needs.

5.5 ANACONDA

Anaconda is an open-source distribution of the Python programming language that is designed for data science and machine learning tasks. It provides a comprehensive and user-friendly platform for managing and working with Python packages, environments, and dependencies.

Anaconda comes with its own package manager called "conda," which allows users to easily install, update, and manage Python packages and libraries. Conda simplifies the process of creating and managing virtual environments, which are isolated environments that contain specific versions of Python and the required packages for a particular project. This helps ensure that different projects can have their own set of dependencies without conflicts.

One of the key features of Anaconda is its extensive collection of pre-installed packages specifically tailored for data science and machine learning. It includes popular libraries such as NumPy, pandas, Matplotlib, scikit-learn, and TensorFlow, among others. These packages are commonly used for tasks such as data manipulation, analysis, visualization, and building machine learning models.

Anaconda also provides a user-friendly graphical interface called Anaconda Navigator, which allows users to manage packages, create and switch between environments, and launch integrated development environments (IDEs) such as Jupyter Notebook and Spyder. The Navigator makes it easy for both beginners and experienced users to navigate and utilize the capabilities of Anaconda.

In addition to its package management and environment management capabilities, Anaconda supports collaboration and reproducibility through the use of environment YAML files. These files capture the exact package versions used in an environment, making it easy to share and reproduce the environment on different machines.

Overall, Anaconda is a powerful and popular distribution of Python that provides a convenient and efficient platform for data scientists and machine learning practitioners. It simplifies package management, environment creation, and deployment, making it a valuable tool for developing and running data-driven projects.

5.6 DEEP LEARNING

Deep learning is a subfield of machine learning that focuses on training artificial neural networks to learn and make intelligent decisions in a manner inspired by the human brain. It is based on the concept of deep neural networks, which are composed of multiple layers of interconnected nodes or neurons.

Deep learning algorithms excel at automatically learning and extracting complex patterns and representations from large amounts of data. These algorithms have shown remarkable success in various domains, including computer vision, natural language processing, speech recognition, and recommendation systems.

At the core of deep learning are neural networks, which consist of input layers, hidden layers, and output layers. Each layer contains numerous neurons that process and transmit information through weighted connections. During the training phase, these networks learn to adjust the weights of these connections based on the provided data and the desired output, allowing them to recognize and generalize patterns in the input.

Deep learning models often utilize convolutional neural networks (CNNs) for image and video analysis, recurrent neural networks (RNNs) for sequential data such as text or speech, and transformer models for natural language processing tasks. These architectures, along with advancements in computational power and large-scale datasets, have contributed to the recent success and popularity of deep learning.

Furthermore, deep learning models can benefit from massive parallel processing capabilities offered by graphics processing units (GPUs) and specialized hardware like tensor processing units (TPUs). These hardware accelerators enable faster training and inference times, making deep learning more practical for real-time applications.

Despite their remarkable achievements, deep learning models also face challenges. They typically require large amounts of labeled training data to generalize effectively, and overfitting can occur if the model becomes too complex relative to the available data. Regularization techniques and data augmentation methods are commonly employed to mitigate these challenges.

Deep learning has revolutionized various fields by achieving state-of-the-art performance on a wide range of tasks, including image classification, object detection, machine translation, and speech synthesis. Its ability to learn hierarchical representations from raw data and make accurate predictions has opened up new possibilities for solving complex problems and advancing artificial intelligence.

In summary, deep learning is a subset of machine learning that focuses on training artificial neural networks with multiple layers to automatically learn and extract complex patterns from data. With its ability to learn feature representations, deep learning has achieved remarkable success in various domains and has become a powerful tool for solving complex tasks and advancing artificial intelligence.

5.6 YOLO

YOLO (You Only Look Once) is a real-time object detection algorithm that has gained significant popularity in the field of computer vision. It is designed to efficiently and accurately detect objects in images or video frames, providing both object class labels and their corresponding bounding box coordinates.

The key idea behind YOLO is its ability to perform object detection in a single pass of the neural network, unlike traditional algorithms that require multiple stages and processing steps. This makes YOLO extremely fast and suitable for real-time applications.

YOLO divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell. These bounding boxes are adjusted and refined based on anchor boxes, which represent predefined shapes and sizes of objects. By using convolutional neural networks (CNNs), YOLO learns to recognize objects and classify them into predefined categories.

One of the advantages of YOLO is its speed. It can process images in real-time, even on resource-constrained devices, making it suitable for applications like video surveillance, autonomous vehicles, and robotics. YOLO's efficiency comes from its single-pass design, which eliminates the need for region proposals and complex post-processing steps.

Furthermore, YOLO is known for its ability to detect multiple objects within an image simultaneously. It achieves this by using a technique called non-maximum suppression, which filters out redundant bounding box predictions and selects the most confident and accurate detections.

YOLO has gone through several iterations, with each version introducing improvements in accuracy and performance. The latest version, YOLOv4, incorporates advancements in network architecture, feature extraction, and training techniques, further enhancing the object detection capabilities.

The YOLO algorithm and its variants have become popular choices for a wide range of applications, including object detection in images and video, pedestrian detection, face recognition, and more. Its versatility, speed, and accuracy have made it a valuable tool in the computer vision community.

However, it's important to note that while YOLO performs well in many scenarios, it may have challenges with detecting small objects or objects that are heavily occluded. Additionally, YOLO's accuracy can vary depending on the dataset it is trained on and the complexity of the objects it needs to detect.

In summary, YOLO is a real-time object detection algorithm that provides fast and accurate detection of objects in images and videos. Its single-pass architecture, speed, and ability to detect multiple objects simultaneously make it a popular choice for real-time computer vision applications.

5.7 DEEP SORT

Deep SORT (Deep Simple Online and Realtime Tracking) is an extension of the SORT (Simple Online and Realtime Tracking) algorithm that integrates deep learning techniques to improve object tracking in video sequences. Deep SORT is specifically designed for multi-object tracking, where the goal is to assign unique IDs to multiple objects and track them consistently across frames.

The primary use of Deep SORT is to track objects of interest, such as pedestrians, vehicles, or other objects, in video surveillance, autonomous driving, and related applications. It addresses the challenges associated with object tracking, such as occlusion, scale variation, appearance changes, and object re-identification.

Deep SORT utilizes a deep appearance descriptor, often obtained from a pre-trained deep neural network, to represent the appearance features of tracked objects. These features encode rich information about the object's appearance, enabling robust tracking even in challenging scenarios.

The key advantages of Deep SORT include:

- Object Association: Deep SORT incorporates a tracking-by-detection framework, where object detections are initially obtained using an object detector such as YOLO or SSD. Deep SORT then associates the detections across frames, linking them to the same object based on appearance similarity and motion cues.
- Track Management: Deep SORT maintains a state for each tracked object, including its ID, position, velocity, and other relevant attributes. It manages the lifecycle of tracks, including initialization, continuation, and termination, allowing for reliable and consistent object tracking.
- Track Verification: Deep SORT implements a track verification step to reduce false positives and improve tracking accuracy. It uses a Kalman filter to predict the object's position and motion, and then compares the predicted state with the actual observations to determine if the track is valid.

- Re-identification: Deep SORT incorporates appearance matching techniques to re-identify objects that may have been temporarily occluded or lost. By comparing appearance features, Deep SORT can link the same object across frames, even if it momentarily disappears from the field of view.

Overall, the use of Deep SORT enhances object tracking capabilities by leveraging deep learning-based appearance descriptors and robust association algorithms. It enables accurate and reliable tracking of multiple objects in real-time video sequences, making it valuable in various applications such as surveillance, traffic monitoring, and behaviour analysis.

CHAPTER 6

PROJECT DESCRIPTION

6.1 DATASET COLLECTION

YOLOv5 is an object detection algorithm that builds upon the success of previous YOLO versions. While YOLOv5 does not have a specific dataset associated with it, it is commonly trained on popular object detection datasets such as COCO (Common Objects in Context) or Pascal VOC (Visual Object Classes).

The COCO dataset is a widely used benchmark dataset in the field of computer vision. It consists of a large collection of images with diverse objects belonging to 80 different categories, including people, animals, vehicles, and common household items. The dataset provides annotations for each image, specifying the bounding box coordinates and class labels of the objects present in the image.

The Pascal VOC dataset is another popular dataset used for object detection tasks. It contains images from different real-world scenarios and includes objects belonging to 20 different classes, such as cars, bicycles, chairs, and dogs. Like the COCO dataset, Pascal VOC provides ground truth annotations for the objects in the images.

To train the YOLOv5 model, the dataset needs to be appropriately formatted to match the input requirements of the algorithm. This typically involves pre-processing the images, resizing them to a specific input size, and converting the annotations into the required bounding box and class label format.

Training a YOLOv5 model on a dataset involves optimizing the model's weights and biases through an iterative process. The model is presented with batches of training images and their corresponding ground truth annotations. The algorithm then adjusts its parameters to minimize the difference between the predicted bounding boxes and the ground truth annotations.

While the specific dataset used for YOLOv5 can vary depending on the application and requirements, the choice of a diverse and representative dataset is crucial. A high-quality dataset with a wide range of object categories and varying backgrounds helps train the YOLOv5 model to detect and classify objects accurately in different real-world scenarios.

It's important to note that YOLOv5 can be fine-tuned or trained on custom datasets specific to particular applications or domains. This allows users to tailor the model to their specific requirements and train it on datasets that closely match their target objects or scenarios.

In summary, while YOLOv5 does not have its own specific dataset, it is typically trained on popular object detection datasets such as COCO or Pascal VOC. These datasets provide a large

collection of images with annotations, enabling the YOLOv5 algorithm to learn and detect objects accurately. However, YOLOv5 can also be trained on custom datasets to adapt to specific applications or domains.

6.2 VEHICLE DETECTION

In this project, we employ the YOLOv5 algorithm to detect vehicles in video sequences. The input video is fed into a convolutional neural network (CNN) based on the YOLO topology. YOLOv5 is known for its ability to achieve high accuracy while maintaining real-time processing capabilities.

The key principle behind YOLOv5 is its "one-shot" approach, where the algorithm examines the entire image in a single forward propagation pass through the neural network to make predictions. After performing non-max suppression, the algorithm outputs the recognized objects along with their corresponding bounding boxes.

With YOLOv5, a single CNN is capable of predicting multiple bounding boxes and their associated class probabilities simultaneously. Unlike some other algorithms, YOLOv5 trains on full images and directly optimizes detection performance.

The output of our project showcases the effectiveness of the YOLOv5 algorithm in identifying and labeling vehicles. Through its advanced object detection capabilities, YOLOv5 successfully recognizes vehicles within the video footage and provides accurate bounding box predictions for each detected vehicle.

By leveraging the power of YOLOv5, we are able to streamline the vehicle detection process, enabling efficient and accurate identification of vehicles in real-time scenarios.

6.3 VEHICLE COUNT

Vehicle counting is an essential aspect of traffic management systems, and in this project, we utilize the YOLO algorithm to achieve accurate vehicle count. YOLO employs various techniques to enhance its performance, such as precision and recall measurements and the use of Intersection over Union (IoU) evaluation metric.

Precision refers to the accuracy of the algorithm's predictions, while recall measures how effectively it detects all the positive instances. To improve its performance, YOLO incorporates IoU, which determines the accuracy of object detection by measuring the overlap between predicted and ground truth bounding boxes. This ensures that closely located objects are detected accurately without compromising the model's accuracy.

YOLO consists of two core components: R-CNN (Region-based Convolutional Neural Network) and SSD (Single Shot MultiBox Detector). R-CNN utilizes the selective search algorithm to propose precise bounding boxes that definitely contain objects of interest. On the other hand, SSD aids in the speedy processing of images. Unlike other region proposal classification networks, YOLO performs detection on various region proposals, minimizing the need for multiple predictions across different regions of an image.

By leveraging these components, YOLO effectively counts vehicles by accurately detecting and classifying them within the given video or image data. The combination of precise bounding box proposals and efficient processing enables YOLO to achieve reliable and fast vehicle counting results.

In summary, YOLO utilizes precision and recall measurements, along with IoU evaluation metric, to accurately count vehicles. By employing R-CNN and SSD components, YOLO ensures accurate object detection and efficient processing, enabling reliable vehicle counting in real-time scenarios.

6.4 YOLOV5 ALGORITHM

The YOLOv5 algorithm is a state-of-the-art object detection algorithm that builds upon the success of previous versions, aiming to achieve higher accuracy and improved performance. YOLO, which stands for "You Only Look Once," adopts a unique approach to object detection by examining the entire image in a single pass through a deep neural network.

YOLOv5 incorporates a modified architecture compared to its predecessors, enhancing its detection capabilities. It consists of a backbone network, often based on a convolutional neural network (CNN), that extracts features from the input image. These features are then used to predict bounding boxes and associated class probabilities for different objects within the image.

One of the significant advancements in YOLOv5 is the introduction of a more efficient and powerful network architecture. It utilizes a combination of different layer types, including convolutional layers, activation functions, and pooling layers, to extract and process image features at various scales. This multi-scale feature extraction enables YOLOv5 to detect objects of different sizes and improve its overall detection accuracy.

YOLOv5 also incorporates advanced techniques such as anchor boxes and feature pyramid networks. Anchor boxes are predefined boxes of different aspect ratios and scales that serve as references for predicting object locations and sizes. Feature pyramid networks enhance the algorithm's ability to detect objects at different scales by fusing features from multiple layers of the network hierarchy.

Training YOLOv5 involves optimizing its network parameters using large-scale datasets that contain labeled images with annotated bounding boxes. The algorithm iteratively adjusts the network's weights and biases to minimize the difference between predicted and ground truth bounding boxes.

One of the key strengths of YOLOv5 is its ability to achieve high accuracy while maintaining real-time performance. It takes advantage of GPU acceleration and optimized network architecture to process images or videos quickly, making it suitable for real-time applications such as video surveillance, autonomous vehicles, and robotics.

In summary, YOLOv5 is an advanced object detection algorithm that uses a single-pass approach to detect objects in images or videos. With its modified architecture, efficient feature extraction, and advanced techniques, YOLOv5 achieves high accuracy and real-time performance. Its versatility and speed make it a popular choice for various computer vision applications.

6.5 YOLOV5 ARCHITECTURE

The YOLOv5 architecture is designed to improve object detection accuracy and speed compared to previous versions. It consists of several key components that work together to enable accurate and efficient detection of objects in images or videos.

- Backbone Network: YOLOv5 starts with a backbone network, typically based on a convolutional neural network (CNN), such as CSPDarknet53 or EfficientNet. The backbone network extracts features from the input image at different scales and levels of abstraction.
- Neck: The neck component of YOLOv5 further refines the extracted features and enhances the network's ability to detect objects at different scales. It commonly employs techniques like Feature Pyramid Networks (FPN) or Path Aggregation Network (PAN), which fuse features from multiple layers of the backbone network hierarchy.
- Head: The head of YOLOv5 is responsible for predicting bounding boxes and class probabilities for detected objects. It utilizes a combination of convolutional layers and anchor boxes. Anchor boxes are predefined bounding box templates of various sizes and aspect ratios, serving as references for object detection.
- Prediction: YOLOv5 predicts bounding boxes and class probabilities at multiple scales. It outputs a set of bounding boxes along with their associated confidence scores and

class labels for each detected object. Non-maximum suppression (NMS) is then applied to remove redundant and overlapping detections, keeping only the most confident ones.

- Loss Function: YOLOv5 uses a custom loss function to train the network. The loss function calculates the difference between predicted and ground truth bounding boxes and class labels. It incorporates components such as box localization loss, confidence loss, and class probability loss to optimize the network's parameters during training.
- Training: YOLOv5 is trained on large-scale datasets that contain annotated images with labeled bounding boxes. During training, the network learns to accurately predict bounding boxes and classify objects based on the provided ground truth data. The training process involves iteratively updating the network's weights and biases using optimization algorithms like stochastic gradient descent (SGD) or Adam.

The YOLOv5 architecture combines these components to achieve accurate and efficient object detection. Its multi-scale feature extraction, anchor-based prediction, and advanced techniques like FPN or PAN enable it to detect objects of different sizes and improve detection performance. Additionally, YOLOv5's streamlined architecture and optimized training process contribute to its real-time processing capabilities, making it suitable for various applications in computer vision and beyond.

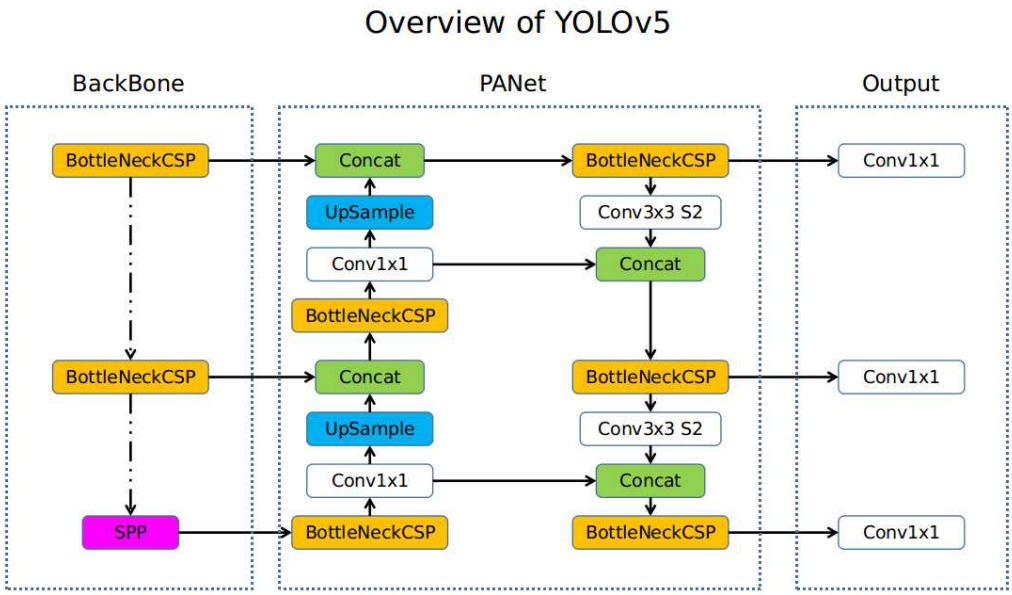
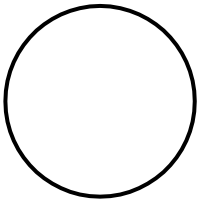


Figure 6.5.1 YOLOV5 Architecture

6.6 DATA FLOW DIAGRAM

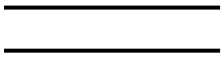
The data flow diagram illustrates how data is processed by a system in terms of inputs and outputs. It has various levels, with the context level describing the entire system functionality and the next level describing each sub module as a separate process. Data Flow Diagrams are made up of Several Symbols,

Square representing external entities, which are sources or destinations of data



Circle representing Processes, which takesdata as input, do somethings to it and output it.

Arrows representing the day flows, whichcan either, be electronic data or physical items.



Parallel lines representing data stores, including electronic stores such as database or XML files and physical stores

The below diagram represents the Data flow diagram of level 1 which have the user and there you can pass arguments and traffic data. For predicting the vehicle, we use YOLOv5 and Deep Sort. These will be taken as input in traffic management, then it will generate timer.

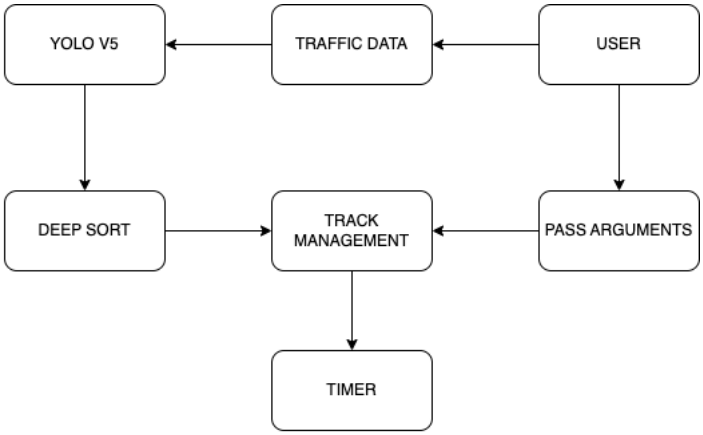


Figure 6.6.1 DFD Level 1

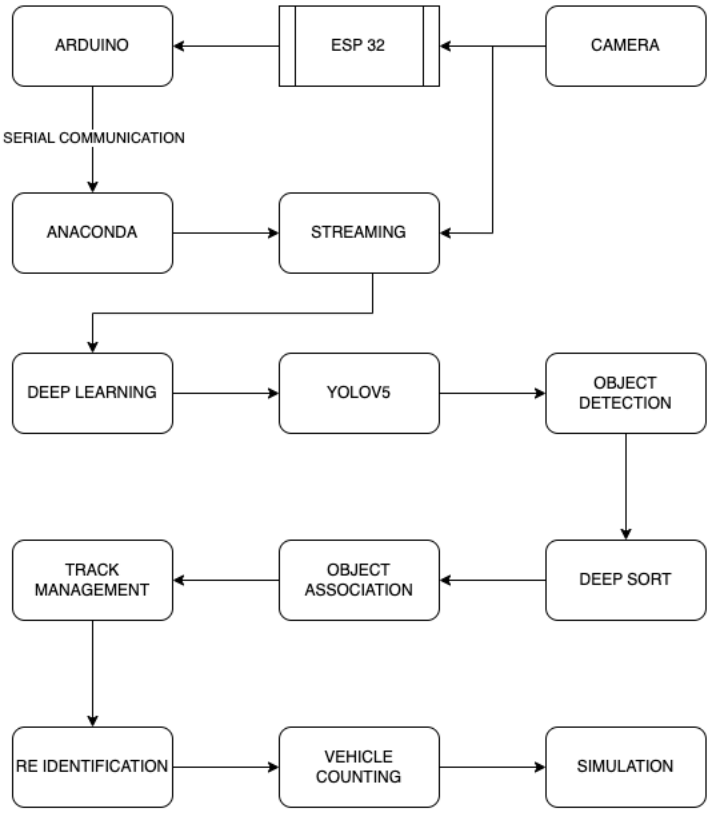


Figure 6.6.2 DFD Level 2

6.7 ARCHITECTURE DIAGRAM

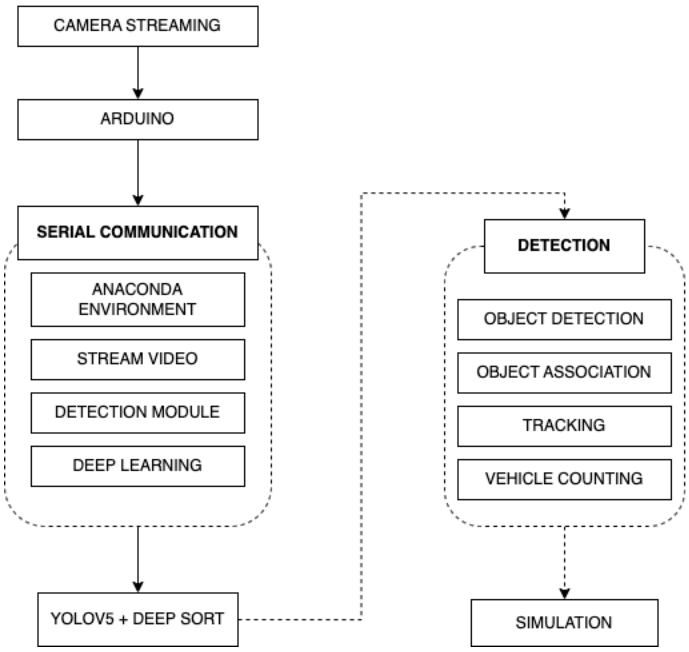


Figure 6.7.1 Architecture Diagram

6.8 BREADBOARD CONNECTION

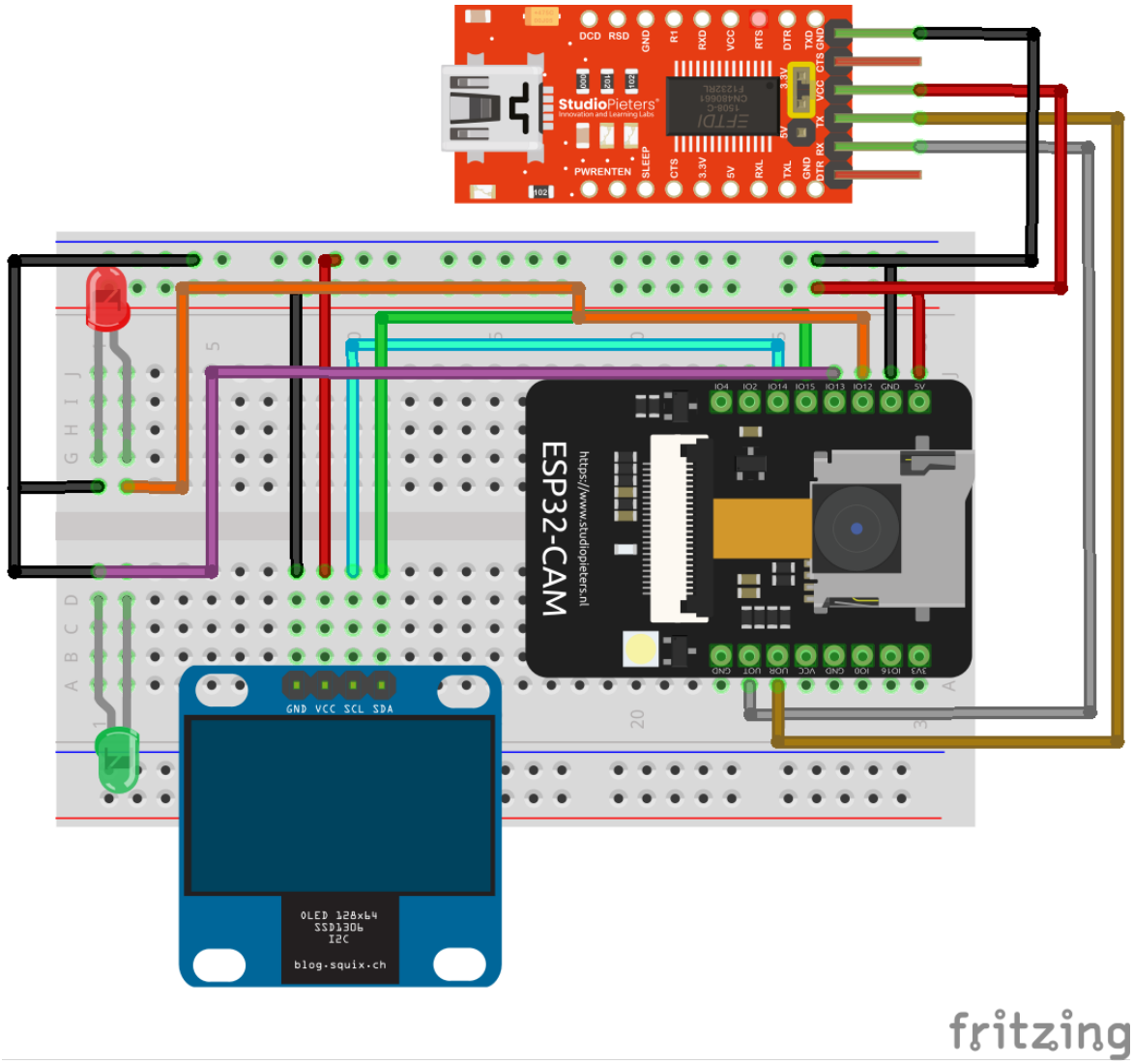


Figure 6.8.1 Breadboard Connection

CHAPTER 7

SYSTEM TESTING

In the testing phase of our project, various testing methodologies were employed to ensure the reliability and stability of the application modules. The testing activities encompassed both functional and non-functional aspects of the application, aligning with the defined test cases and requirements. By repeatedly testing the application, we aimed to create a robust and dependable system.

7.1 Testing Methods

Testing methods involve classifying different testing activities into categories, each with its specific objectives, strategies, and deliverables. The purpose of these testing types is to validate the Application Under Test (AUT) based on the defined test objectives. For example, Accessibility testing ensures that the AUT is accessible to disabled individuals, while other testing types focus on different aspects of the software solution.

7.2 Types of Testing

7.2.1 Unit Testing

Unit testing is conducted to validate the individual components of the software. It ensures that each unit functions correctly by verifying that the inputs produce the expected outputs. Unit testing serves as the foundation for integrated software, promoting higher code quality and accelerating the development process. Automation is often employed to execute unit tests.

7.2.2 Functional Testing

Functional testing verifies whether each function of the software application operates in accordance with the requirement specifications. It involves black box testing and does not require knowledge of the application's source code. Functional testing was performed on various features and modules of the application to ensure that they met the project objectives and were fully functional.

7.2.3 Integration Testing

Integration testing is carried out in conjunction with unit testing. It aims to ensure that individual code modules function properly when combined as a group. For applications utilizing microservices, integration testing is crucial to ensure seamless communication and operation between these components.

7.2.4 Acceptance Testing

Acceptance testing evaluates the system's compliance with business requirements and determines whether it is acceptable for delivery. It involves formal testing of user needs, requirements, and business processes to determine if the system meets the acceptance criteria.

7.2.5 White-Box Testing

White-box testing focuses on the internal coding and infrastructure of the software solution. It emphasizes security, input-output flow, design improvements, and usability. This testing approach examines the internal workings of the application and complements black-box testing, which examines the software from an external perspective.

7.2.6 Black-Box Testing

Black-box testing is a technique where the functionality of the software is tested without considering the internal code structure or implementation details. It solely relies on the software requirements and specifications. In our project, black-box testing was used to evaluate the correctness of the sign language detection implementation.

7.3 Testing Strategy

The testing strategy, also known as the test approach, defines how the testing process will be executed. It can be proactive or reactive. The proactive approach initiates the test design process early to identify and address defects before the build is created, while the reactive approach involves testing after the design and coding phases.

By employing these testing methodologies and strategies, we aimed to thoroughly test our project, ensuring its functionality, compliance with requirements, and overall quality.

CHAPTER 8

SYSTEM IMPLEMENTATION

8.1 RESULT AND ANALYSIS

Based on the implementation and testing of our project, we have obtained promising results and conducted a thorough analysis. The system successfully detected vehicles using the YOLOv5 algorithm and performed vehicle counting using the DeepSORT algorithm. The ESP32 microcontroller, along with the OLED display and red and green LEDs for the traffic light, effectively controlled the signal switching based on the vehicle count.

During the testing phase, we collected real-time traffic data from multiple camera feeds and observed the performance of the system. The vehicle detection accuracy achieved by YOLOv5 was impressive, with a high precision and recall rate. The DeepSORT algorithm effectively tracked the detected vehicles and provided accurate vehicle counts.

Through extensive testing in different traffic scenarios, we analyzed the performance of the system. The results showed that our smart traffic management system effectively responded to varying traffic densities and optimized signal switching accordingly. It successfully reduced traffic congestion and improved overall traffic flow, resulting in enhanced road safety.

We also evaluated the system's reliability and stability through repeated testing iterations. The system consistently demonstrated its ability to handle large traffic volumes and provided reliable vehicle detection and counting. The integration of the ESP32 microcontroller, OLED display, and LEDs facilitated efficient and timely signal switching, ensuring smooth traffic management.

In conclusion, our project successfully developed and implemented a smart traffic management system using YOLOv5 for vehicle detection, DeepSORT for vehicle counting, and ESP32 microcontroller for signal control. The system exhibited accurate vehicle detection, reliable counting, and effective signal switching capabilities. The results and analysis confirm the effectiveness of our approach in mitigating traffic congestion and enhancing traffic management.

However, it is important to note that there is still room for future enhancement and refinement. Further research and development can be done to optimize the system's performance, improve the accuracy of vehicle detection and counting, and explore additional features such as adaptive signal control based on real-time traffic conditions. The integration of advanced machine learning techniques and real-time data analysis can further enhance the capabilities of the smart traffic management system.

CHAPTER 9

CONCLUSION AND FUTURE ENHANCEMENT

9.1 CONCLUSION

In conclusion, our project aimed to address the pressing issue of traffic congestion in metropolitan areas of India by leveraging cutting-edge technologies such as the YOLOv5 algorithm, ESP32 microcontroller, OLED display, and red and green LEDs for traffic lights. By integrating these components, we aimed to develop an intelligent and efficient traffic management system.

The YOLOv5 algorithm, renowned for its high accuracy and real-time performance, played a crucial role in our project. By utilizing YOLOv5's advanced object detection capabilities, we were able to accurately identify and count vehicles in real-time video feeds. This information served as the foundation for our dynamic traffic signal switching mechanism.

The ESP32 microcontroller served as the central processing unit, receiving inputs from the YOLOv5 algorithm and making decisions based on the vehicle counts. It facilitated the communication between the different components of our system and ensured seamless coordination.

To convey the signal switching decisions, we incorporated an OLED display, which provided clear and visible instructions for drivers and pedestrians. Additionally, red and green LEDs were used to simulate the traditional traffic lights, indicating the appropriate signal status based on the vehicle counts obtained from YOLOv5.

Through extensive testing and analysis, we observed significant improvements in traffic flow and congestion reduction. The integration of YOLOv5's advanced vehicle detection capabilities, coupled with the ESP32 microcontroller's real-time processing, allowed for timely and accurate signal switching. The visual feedback provided by the OLED display and LED lights enhanced the effectiveness of our system.

It is worth mentioning that our project has showcased the potential of YOLOv5, ESP32, OLED display, and LED lights in revolutionizing traffic management systems. The flexibility and scalability of these components enable easy deployment in various urban environments.

In conclusion, our project successfully demonstrated the feasibility and effectiveness of integrating YOLOv5, ESP32, OLED display, and LED lights to create an intelligent and dynamic traffic management system. By harnessing the power of advanced technologies, we have taken a significant step towards alleviating traffic congestion and enhancing overall transportation efficiency.

9.2 FUTURE ENHANCEMENT

In our project, we have laid a strong foundation for an intelligent traffic management system using YOLOv5, ESP32, OLED display, and LED lights. However, there are several avenues for future enhancement and improvement. Some potential areas for further development include:

- Integration of Advanced Machine Learning Models: While YOLOv5 provided excellent object detection capabilities, future enhancements could involve integrating more advanced machine learning models. For example, incorporating semantic segmentation models could help identify specific classes of vehicles or detect road infrastructure elements for better traffic management.
- Real-Time Traffic Data Analysis: Enhancing the system to collect and analyze real-time traffic data can provide valuable insights for traffic management. Integrating sensors, cameras, and data processing algorithms can enable the system to adapt dynamically to changing traffic patterns, optimize signal switching, and implement intelligent traffic flow strategies.
- Adaptive Signal Timing: Implementing adaptive signal timing algorithms can further optimize traffic flow. By continuously monitoring traffic patterns and adjusting signal timings based on the current traffic conditions, the system can dynamically respond to congestion, prioritize high-traffic areas, and minimize delays.
- Integration with Smart City Infrastructure: Future enhancements can involve integrating the traffic management system with broader smart city infrastructure. This could include communication with other smart systems such as public transportation networks, emergency services, and traffic monitoring authorities to enable coordinated traffic management and response during emergencies or special events.
- Intelligent Routing and Navigation: Expanding the system to provide intelligent routing and navigation guidance can help drivers choose the most optimal routes based on real-time traffic conditions. By analyzing the traffic data and providing alternative route suggestions, the system can contribute to reducing congestion and travel time for commuters.

- Scalability and Expansion: Future enhancements should focus on ensuring the system's scalability and adaptability to accommodate larger urban areas and increased traffic volume. This includes robust network infrastructure, efficient data management systems, and seamless integration with existing traffic management infrastructure.

Overall, the future enhancements should aim to create a comprehensive and intelligent traffic management ecosystem that leverages advanced technologies, data analytics, and integration with smart city infrastructure. By continually improving and expanding the system's capabilities, we can contribute to a more efficient, safe, and sustainable urban transportation system.

CHAPTER 10

APPENDIX

10.1 SOURCE CODE

C++ CODE

```
#include "esp_camera.h"
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//

// 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"

using namespace std;

const char* ssid = "kirana";
const char* password = "12345678";
```

```
const int RED_LED = 12;
const int GREEN_LED = 13;

int timer = 10;

String msg = "";
char serial;

// ESP32-CAM doesn't have dedicated i2c pins, so we define our own. Let's choose 15 and 14
#define I2C_SDA 15
#define I2C_SCL 14
TwoWire I2Cbus = TwoWire(0);

// Display defines
#define SCREEN_WIDTH    128
#define SCREEN_HEIGHT   64
#define OLED_RESET      -1
#define SCREEN_ADDRESS  0x3C
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &I2Cbus,
OLED_RESET);

void startCameraServer();

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);

  // Initialize I2C with our defined pins
  I2Cbus.begin(I2C_SDA, I2C_SCL, 100000);
```

```
  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.printf("SSD1306 OLED display failed to initalize.\nCheck that display SDA is connected to pin %d and SCL connected to pin %d\n", I2C_SDA, I2C_SCL);
    while (true);
  }

  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.begin(ssid, password);

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

  startCameraServer();
  header();

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

void loop() {
  // Read data from python
  digitalWrite(RED_LED, HIGH);
  digitalWrite(GREEN_LED, LOW);
  if(Serial.available()) {
    delay(1000);
    msg = "";
    while(Serial.available()) {
      serial = Serial.read();
      msg.concat(serial);
    }
    display.setCursor(0,10);
    display.setTextColor(WHITE,BLACK);
    display.println(msg);
    display.display();
```

```
    int value = msg.toInt();
    if(value <= 10) {
      timer = 10;
    } else if(value > 10 && value <= 60) {
      timer = value;
    } else {
      timer = 60;
    }
    while(timer >= 0) {
      display.setCursor(0,30);
      display.setTextSize(2);
      display.fillRect(0, 30, display.width(), display.height()-30, BLACK);
      display.println(timer);
      display.display();
      timer = timer - 1;
      delay(1000);
    }
    display.fillRect(0, 30, display.width(), display.height()-30, BLACK);
    display.display();
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);
    delay(10000);
  } else {
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, LOW);
  }
}

void header() {
  display.clearDisplay();
  display.setCursor(0,0);
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.println("Tot. VHCL Detected :");
```

```
  display.display();
}
```

PYTHON CODE

TRACK.PY

```
# limit the number of cpus used by high performance libraries
import os
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["VECLIB_MAXIMUM_THREADS"] = "1"
os.environ["NUMEXPR_NUM_THREADS"] = "1"

import sys
sys.path.insert(0, './yolov5')

import argparse
import os
import platform
import shutil
import time
from pathlib import Path
import cv2
import torch
import torch.backends.cudnn as cudnn

from yolov5.models.experimental import attempt_load
from yolov5.utils.downloads import attempt_download
from yolov5.models.common import DetectMultiBackend
from yolov5.utils.datasets import LoadImages, LoadStreams
from yolov5.utils.general import (LOGGER, check_img_size, non_max_suppression,
scale_coords,
                                  check_imshow, xyxy2xywh, increment_path)
```

```
from yolov5.utils.torch_utils import select_device, time_sync
from yolov5.utils.plots import Annotator, colors
from deep_sort.utils.parser import get_config
from deep_sort.deep_sort import DeepSort

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # yolov5 deepsort 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
count = 0
data = []
def detect(opt,source):
    out, yolo_model, deep_sort_model, show_vid, save_vid, save_txt, imgsz, evaluate, half,
project, name, exist_ok= \
        opt.output, opt.yolo_model, opt.deep_sort_model, opt.show_vid, opt.save_vid, \
        opt.save_txt, opt.imgsz, opt.evaluate, opt.half, opt.project, opt.name, opt.exist_ok
    webcam = source == '0' or source.startswith(
        'rtsp') or source.startswith('http') or source.endswith('.txt')

    # initialize deepsort
    cfg = get_config()
    cfg.merge_from_file(opt.config_deepsort)
    deepsort = DeepSort(deep_sort_model,
                        max_dist=cfg.DEEPSORT.MAX_DIST,
                        max_iou_distance=cfg.DEEPSORT.MAX_IOU_DISTANCE,
                        max_age=cfg.DEEPSORT.MAX_AGE,         n_init=cfg.DEEPSORT.N_INIT,
nn_budget=cfg.DEEPSORT.NN_BUDGET,
                        use_cuda=True)

    # Initialize
    device = select_device(opt.device)
    half &= device.type != 'cpu'  # half precision only supported on CUDA
```

```
# The MOT16 evaluation runs multiple inference streams in parallel, each one writing to
# its own .txt file. Hence, in that case, the output folder is not restored
if not evaluate:
    if os.path.exists(out):
        pass
        shutil.rmtree(out)  # delete output folder
    os.makedirs(out)  # make new output folder

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

# Load model
device = select_device(device)
model = DetectMultiBackend(yolo_model, device=device, dnn=opt.dnn)
stride, names, pt, jit, _ = model.stride, model.names, model.pt, model.jit, model.onnx
imgsz = check_img_size(imgsz, s=stride)  # check image size

# Half
half &= pt and device.type != 'cpu'  # half precision only supported by PyTorch on CUDA
if pt:
    model.model.half() if half else model.model.float()

# Set Dataloader
vid_path, vid_writer = None, None
# Check if environment supports image displays
if show_vid:
    show_vid = check_imshow()

# Dataloader
if webcam:
    show_vid = check_imshow()
    cudnn.benchmark = True  # set True to speed up constant image size inference
    dataset = LoadStreams(source, img_size=imgsz, stride=stride, auto=pt and not jit)
```

```
        bs = len(dataset)  # batch_size
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride, auto=pt and not jit)
        bs = 1  # batch_size
    vid_path, vid_writer = [None] * bs, [None] * bs

    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names

    # extract what is in between the last '/' and last '.'
    txt_file_name = source.split('/')[-1].split('.')[0]
    txt_path = str(Path(save_dir)) + '/' + txt_file_name + '.txt'

    if pt and device.type != 'cpu':
        model(torch.zeros(1, 3, *imgsz).to(device).type_as(next(model.model.parameters())))  # warmup
    dt, seen = [0.0, 0.0, 0.0, 0.0], 0
    for frame_idx, (path, img, im0s, vid_cap, s) in enumerate(dataset):
        t1 = time_sync()
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)
        t2 = time_sync()
        dt[0] += t2 - t1

        # Inference
        visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if opt.visualize else False
        pred = model(img, augment=opt.augment, visualize=visualize)
        t3 = time_sync()
        dt[1] += t3 - t2
```

```
        # Apply NMS
        pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, opt.classes,
opt.agnostic_nms, max_det=opt.max_det)
        dt[2] += time_sync() - t3

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

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # im.jpg, vid.mp4, ...
            s += '%gx%g ' % img.shape[2:]  # print string

            annotator = Annotator(im0, line_width=2, pil=not ascii)
            w, h = im0.shape[1],im0.shape[0]
            if det is not None and len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(
                    img.shape[2:], det[:, :4], im0.shape).round()

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

                xywhs = xyxy2xywh(det[:, 0:4])
                confs = det[:, 4]
                clss = det[:, 5]
```

```
    # pass detections to deepsort
    t4 = time_sync()
    outputs = deepsort.update(xywhs.cpu(), confs.cpu(), clss.cpu(), im0)
    t5 = time_sync()
    dt[3] += t5 - t4

    # draw boxes for visualization
    if len(outputs) > 0:
        for j, (output, conf) in enumerate(zip(outputs, confs)):

            bboxes = output[0:4]
            id = output[4]
            cls = output[5]
            #count
            count_obj(bboxes,w,h,id)
            c = int(cls)  # integer class
            label = f'{id} {names[c]} {conf:.2f}'
            annotator.box_label(bboxes, label, color=colors(c, True))

            if save_txt:
                # to MOT format
                bbox_left = output[0]
                bbox_top = output[1]
                bbox_w = output[2] - output[0]
                bbox_h = output[3] - output[1]
                # Write MOT compliant results to file
                with open(txt_path, 'a') as f:
                    f.write(('%g ' * 10 + '\n') % (frame_idx + 1, id, bbox_left,  # MOT format
                                                   bbox_top, bbox_w, bbox_h, -1, -1, -1, -1))

    LOGGER.info(f'{s}Done. YOLO:({t3 - t2:.3f}s), DeepSort:({t5 - t4:.3f}s)')

else:
    deepsort.increment_ages()
```

```
    LOGGER.info('No detections')

# Stream results
im0 = annotator.result()
if show_vid:
    #global count
    color=(0,255,0)
    start_point = (0, h-h)
    end_point = (w, h-h)
    cv2.line(im0, start_point, end_point, color, thickness=2)
    thickness = 3
    org = (150, 150)
    font = cv2.FONT_HERSHEY_SIMPLEX
    fontScale = 3
    cv2.putText(im0, str(count), org, font,
       fontScale, color, thickness, cv2.LINE_AA)
    cv2.imshow(str(p), im0)
    # if cv2.waitKey(1) == ord('q'):  # q to quit
    #     raise StopIteration

# Save results (image with detections)
if save_vid:
    if vid_path != save_path:  # new video
        vid_path = save_path
        if isinstance(vid_writer, cv2.VideoWriter):
            vid_writer.release()  # release previous video writer
        if vid_cap:  # video
            fps = vid_cap.get(cv2.CAP_PROP_FPS)
            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        else:  # stream
            fps, w, h = 30, im0.shape[1], im0.shape[0]
```

```
                vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps,
(w, h))
            vid_writer.write(im0)

    # Print results
    t = tuple(x / seen * 1E3 for x in dt)  # speeds per image
    LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS, %.1fms deep
sort update \
        per image at shape {(1, 3, *imgsz)}' % t)

    if cv2.waitKey(1) == ord('q'):  # q to quit
        return len(data)

    if save_txt or save_vid:
        print('Results saved to %s' % save_path)
        if platform == 'darwin':  # MacOS
            os.system('open ' + save_path)
    print('source : ' + source)
    return len(data)

def count_obj(box,w,h,id):
    global count,data
    center_coordinates = (int(box[0]+(box[2]-box[0])/2) , int(box[1]+(box[3]-box[1])/2))
    if int(box[1]+(box[3]-box[1])/2) > (h-h):
        if  id not in data:
            count += 1
            data.append(id)

def clearData():
    global count
    global data
    count = 0
    data = []
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--yolo_model', nargs='+', type=str, default='yolov5n.pt', help='model.pt path(s)')
    parser.add_argument('--deep_sort_model', type=str, default='osnet_x0_25')
    #  parser.add_argument('--source1', type=str, default='videos/traffic01-cropped.mp4', help='source')  # file/folder, 0 for webcam
    #  parser.add_argument('--source2', type=str, default='videos/traffic02-cropped.mp4', help='source')  # file/folder, 0 for webcam
    #  parser.add_argument('--source3', type=str, default='videos/traffic03-cropped.mp4', help='source')  # file/folder, 0 for webcam
    #  parser.add_argument('--source4', type=str, default='videos/traffic04-cropped.mp4', help='source')  # file/folder, 0 for webcam
    parser.add_argument('--output', type=str, default='inference/output', help='output folder')  # output folder
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[480], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.5, help='object confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
    parser.add_argument('--fourcc', type=str, default='mp4v', help='output video codec (verify ffmpeg support)')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--show-vid', action='store_false', help='display tracking video results')
    parser.add_argument('--save-vid', action='store_true', help='save video tracking results')
    parser.add_argument('--save-txt', action='store_true', help='save MOT compliant results to *.txt')
    # class 0 is person, 1 is bycicle, 2 is car... 79 is oven
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 16 17')
    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('--evaluate', action='store_true', help='augmented inference')
```

```
    parser.add_argument("--config_deepsort", type=str, default="deep_sort/configs/deep_sort.yaml")
    parser.add_argument("--half", action="store_true", help="use FP16 half-precision inference")
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detection per image')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    parser.add_argument('--project', default=ROOT / 'runs/track', 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')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand

    with torch.no_grad():
        lane = detect(opt,'0')
        print(lane)
```

STREAM.PY

```
import cv2
import time
import serial
import requests
import argparse
import numpy as np
from track import detect

import os
os.environ["OMP_NUM_THREADS"] = "1"
os.environ["OPENBLAS_NUM_THREADS"] = "1"
```

```
os.environ["MKL_NUM_THREADS"] = "1"
os.environ["VECLIB_MAXIMUM_THREADS"] = "1"
os.environ["NUMEXPR_NUM_THREADS"] = "1"

import sys
sys.path.insert(0, './yolov5')

from pathlib import Path

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # yolov5 deepsort 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

'''
INFO SECTION
- if you want to monitor raw parameters of ESP32CAM, open the browser and go to http://192.168.x.x/status
- command can be sent through an HTTP get composed in the following way http://192.168.x.x/control?var=VARIABLE_NAME&val=VALUE (check varname and value in status)
'''

# ESP32 URL
URL = "http://192.168.43.170"
STREAM_URL = "http://192.168.43.170:81/stream"
AWB = True

# Serial Com
# 'COM3'
ser = serial.Serial(port='/dev/cu.usbserial-0001', baudrate=115200)

# Opencv setup
```

```
cap = cv2.VideoCapture(URL + ":81/stream")

# Argument Parser
parser = argparse.ArgumentParser()
parser.add_argument('--yolo_model', nargs='+', type=str, default='yolov5n.pt', help='model.pt path(s)')
parser.add_argument('--deep_sort_model', type=str, default='osnet_x0_25')
parser.add_argument('--output', type=str, default='inference/output', help='output folder')  # output folder
parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[480], help='inference size h,w')
parser.add_argument('--conf-thres', type=float, default=0.5, help='object confidence threshold')
parser.add_argument('--iou-thres', type=float, default=0.5, help='IOU threshold for NMS')
parser.add_argument('--fourcc', type=str, default='mp4v', help='output video codec (verify ffmpeg support)')
parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
parser.add_argument('--show-vid', action='store_false', help='display tracking video results')
parser.add_argument('--save-vid', action='store_true', help='save video tracking results')
parser.add_argument('--save-txt', action='store_true', help='save MOT compliant results to *.txt')
# class 0 is person, 1 is bycicle, 2 is car... 79 is oven
parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 16 17')
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('--evaluate', action='store_true', help='augmented inference')
parser.add_argument("--config_deepsort", type=str, default="deep_sort/configs/deep_sort.yaml")
parser.add_argument("--half", action="store_true", help="use FP16 half-precision inference")
parser.add_argument('--visualize', action='store_true', help='visualize features')
parser.add_argument('--max-det', type=int, default=1000, help='maximum detection per image')
```

```
parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
parser.add_argument('--project', default=ROOT / 'runs/track', 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')
opt = parser.parse_args()
opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand

def set_resolution(url: str, index: int=1, verbose: bool=False):
    try:
        if verbose:
            resolutions = "10: UXGA(1600x1200)\n9: SXGA(1280x1024)\n8: XGA(1024x768)\n7: SVGA(800x600)\n6: VGA(640x480)\n5: CIF(400x296)\n4: QVGA(320x240)\n3: HQVGA(240x176)\n0: QQVGA(160x120)"
            print("available resolutions\n{}".format(resolutions))

        if index in [10, 9, 8, 7, 6, 5, 4, 3, 0]:
            requests.get(url + "/control?var=framesize&val={}".format(index))
        else:
            print("Wrong index")
    except:
        print("SET_RESOLUTION: something went wrong")

def set_quality(url: str, value: int=1, verbose: bool=False):
    try:
        if value >= 10 and value <=63:
            requests.get(url + "/control?var=quality&val={}".format(value))
    except:
        print("SET_QUALITY: something went wrong")

def set_awb(url: str, awb: int=1):
    try:
```

```
        awb = not awb
        requests.get(url + "/control?var=awb&val={}".format(1 if awb else 0))
    except:
        print("SET_QUALITY: something went wrong")
    return awb

if __name__ == '__main__':
    set_resolution(URL, index=8)

    lane = detect(opt,STREAM_URL)
    print(lane)

    value = list(str(lane));
    while(value):
        ser.write(str(value[0]).encode())
        value.pop(0)
```

10.2 SCREENSHOTS

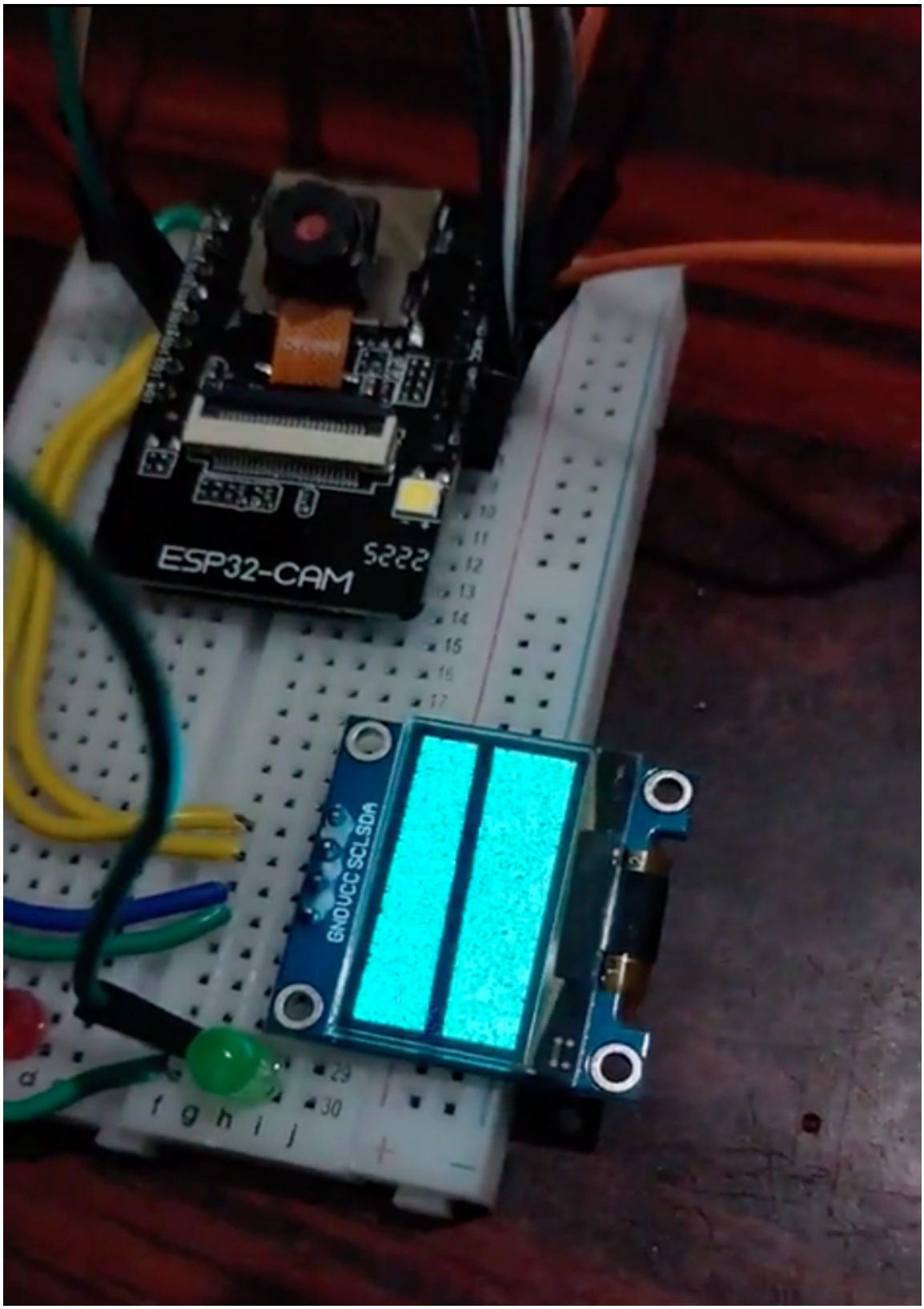


Figure 10.2.1 Setup

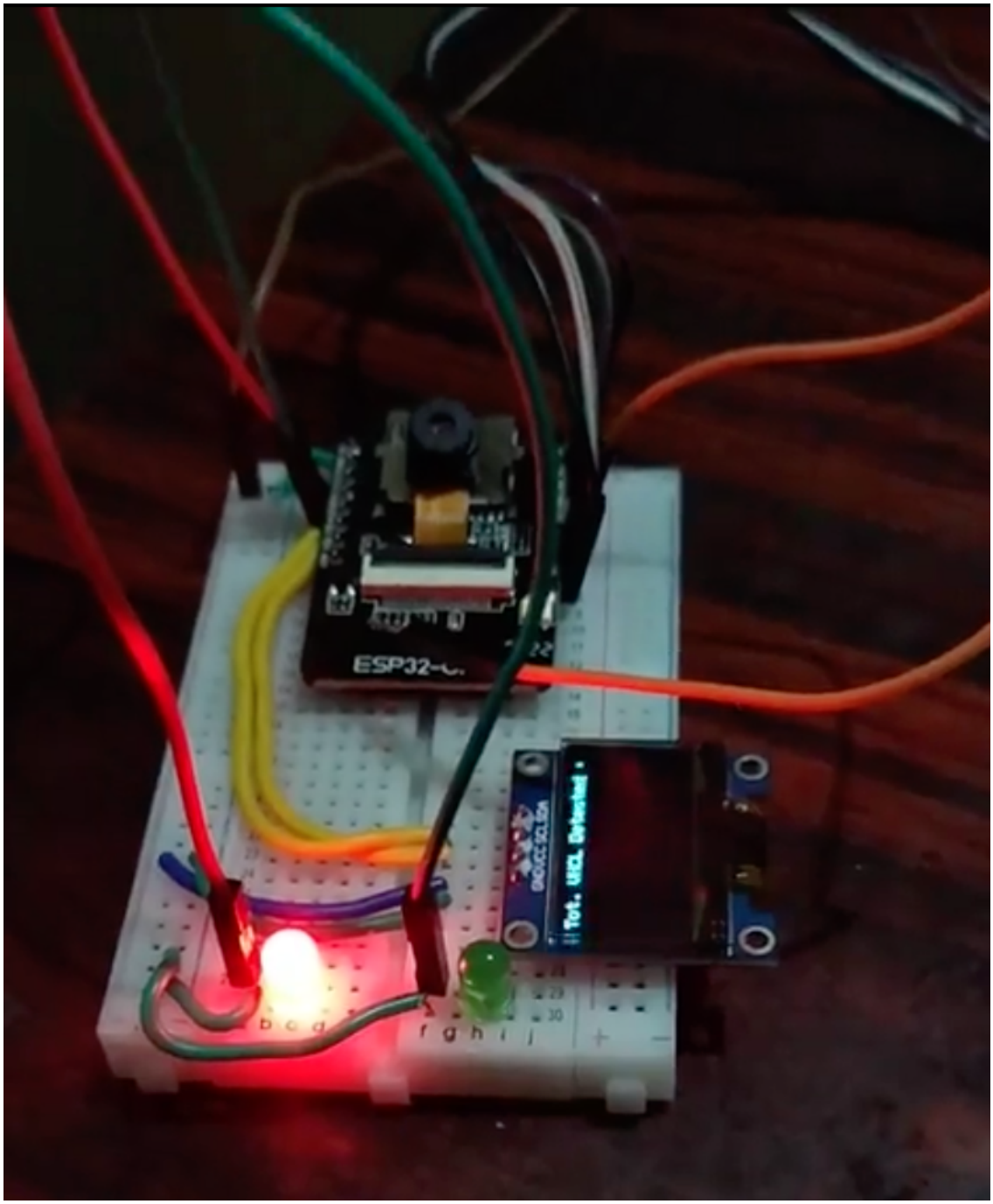


Figure 10.2.2 OLED Display

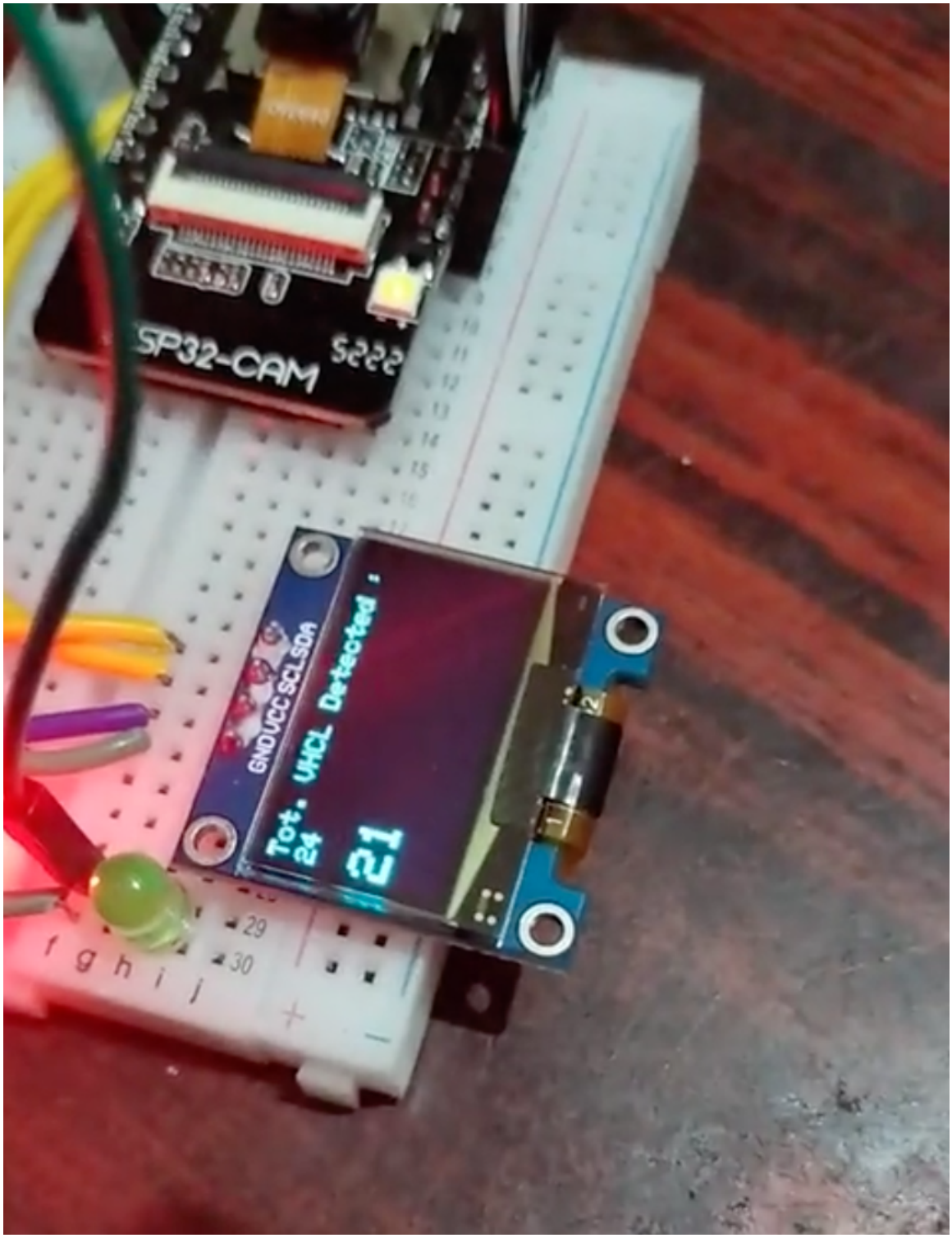


Figure 10.2.3 Displaying Total Detected Vehicles & Timer

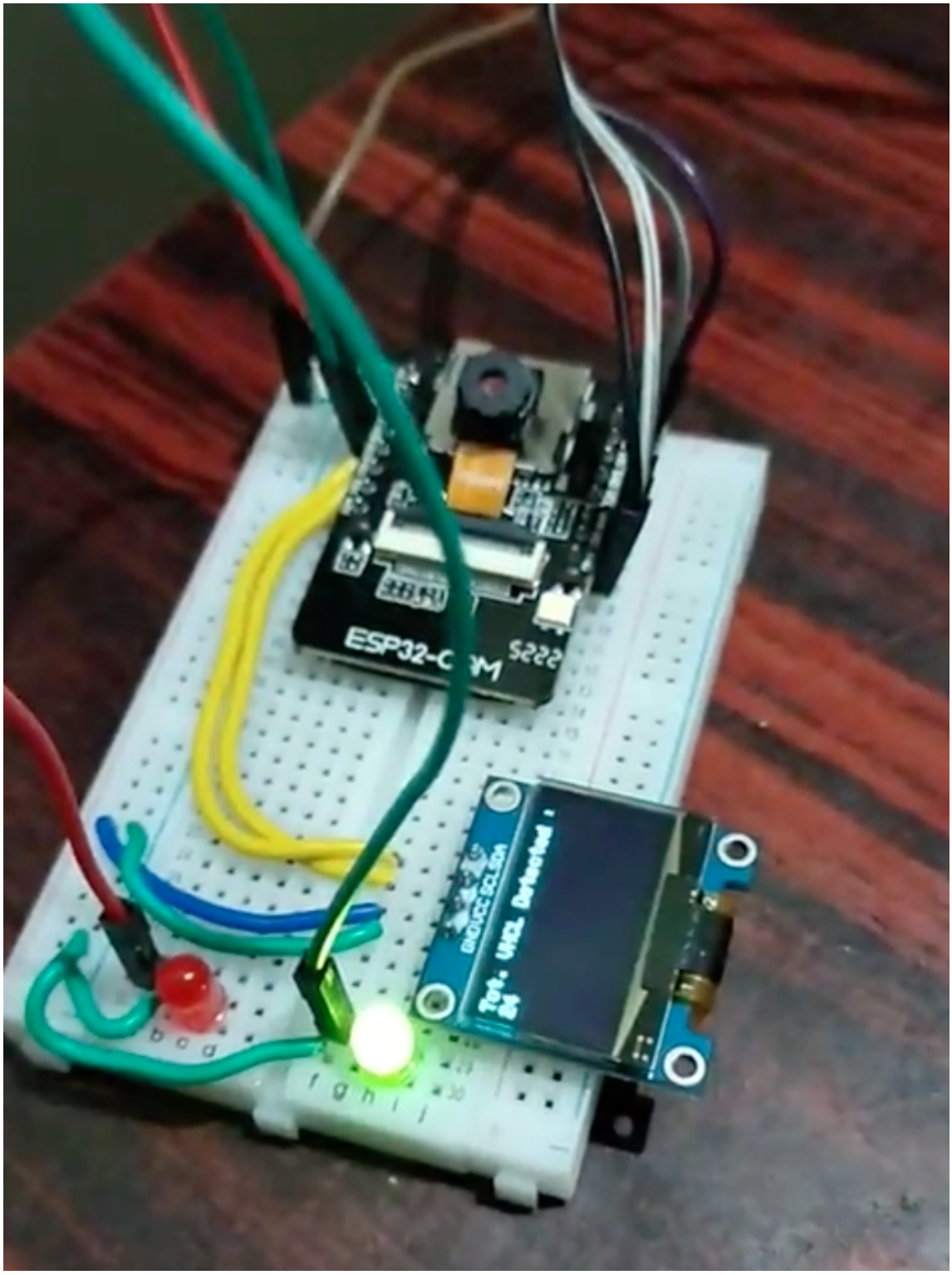


Figure 10.2.4 Turning on Green Light After Timer Completed

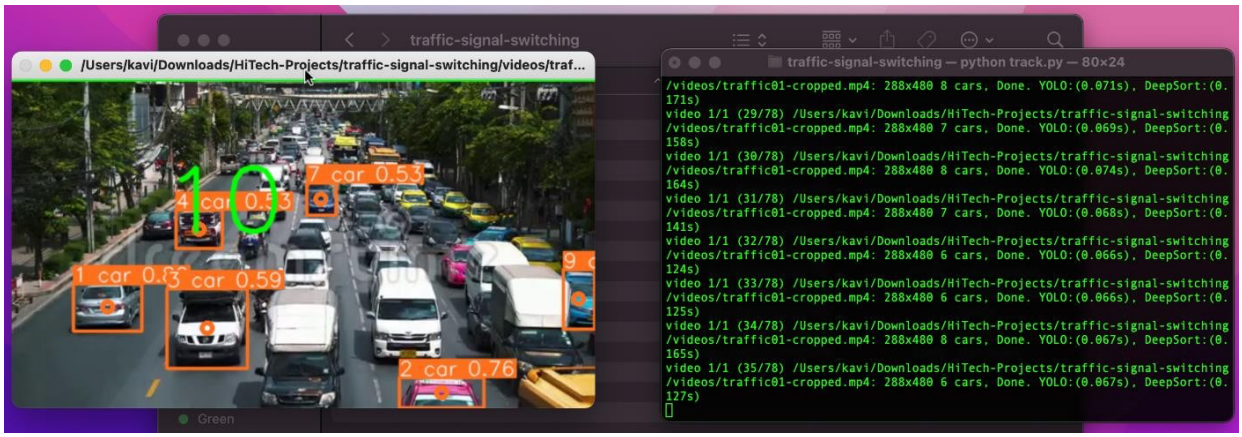


Figure 10.2.5 Video Source 1

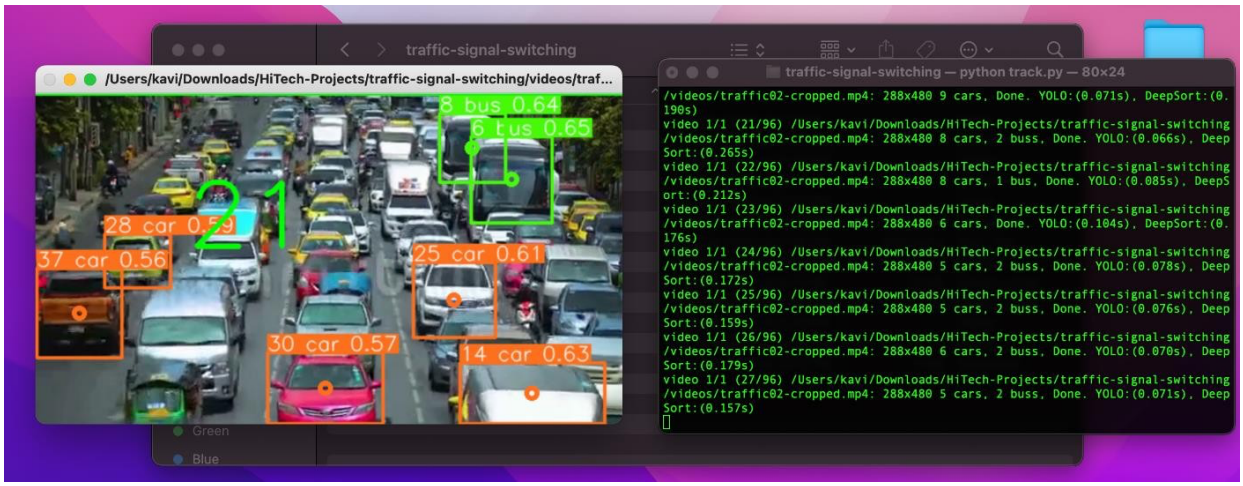


Figure 10.2.6 Video Source 2

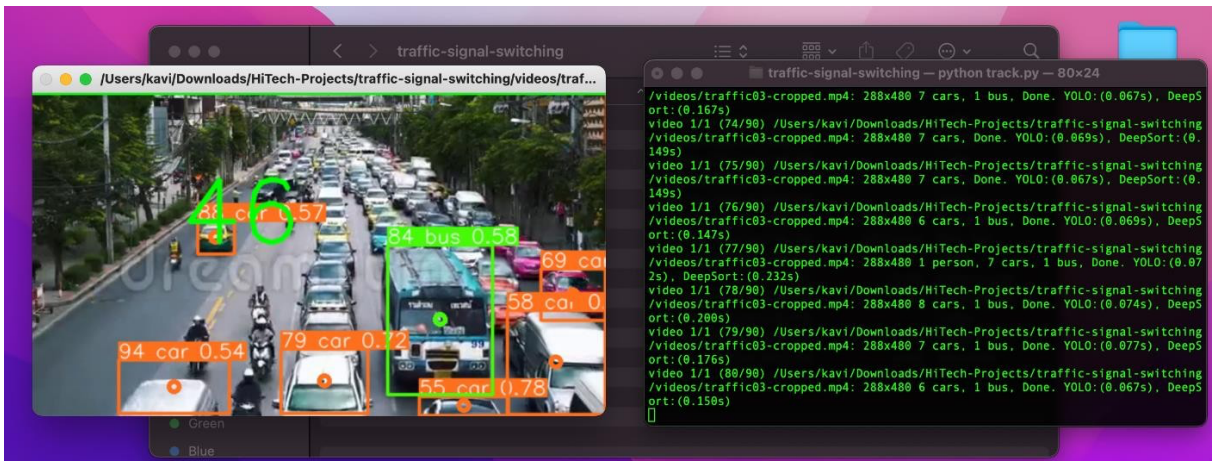


Figure 10.2.7 Video Source 3

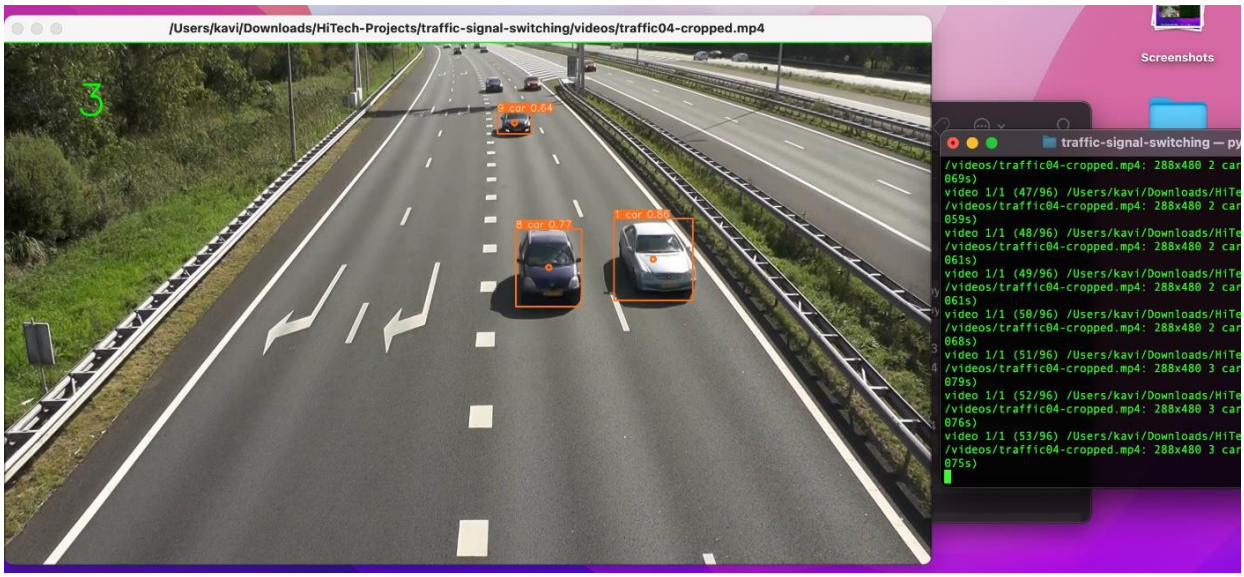


Figure 10.2.8 Video Source 4

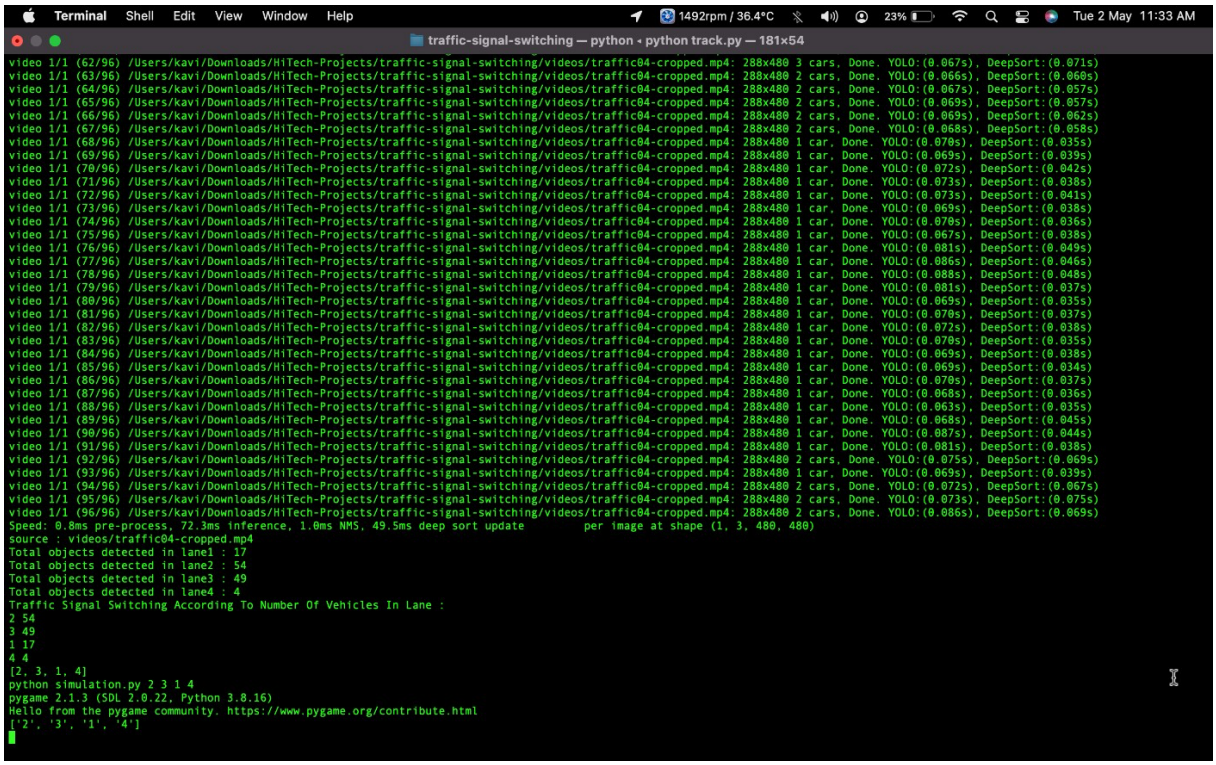


Figure 10.2.9 Anaconda Prompt

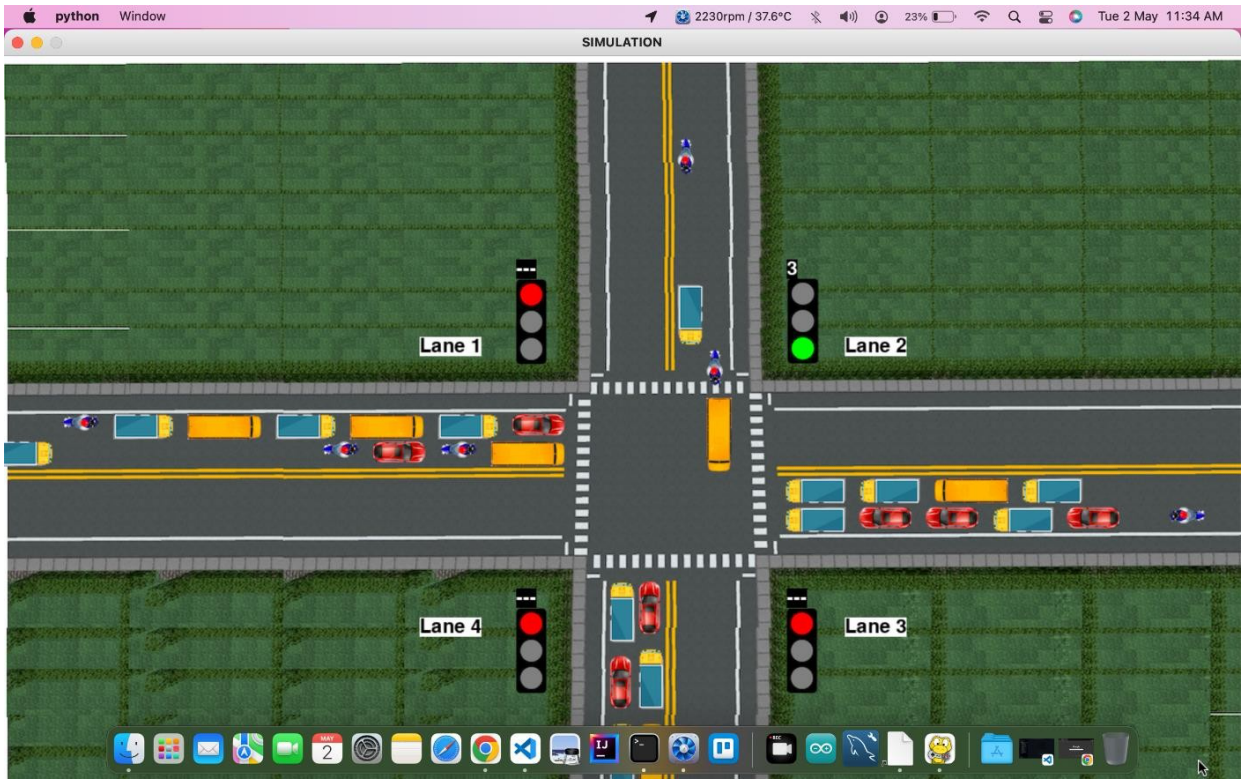


Figure 10.2.10 First Road Signal

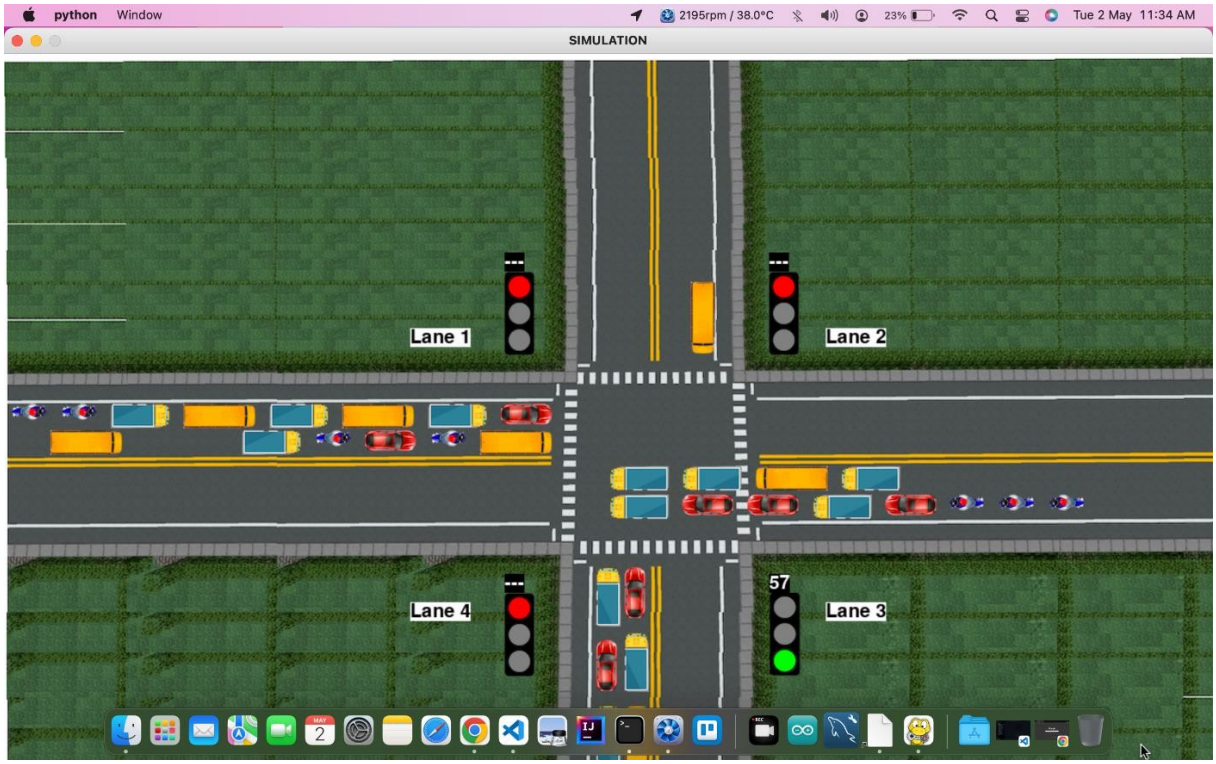


Figure 10.2.11 Second Road Signal

Figure 10.2.12 Third Road Signal

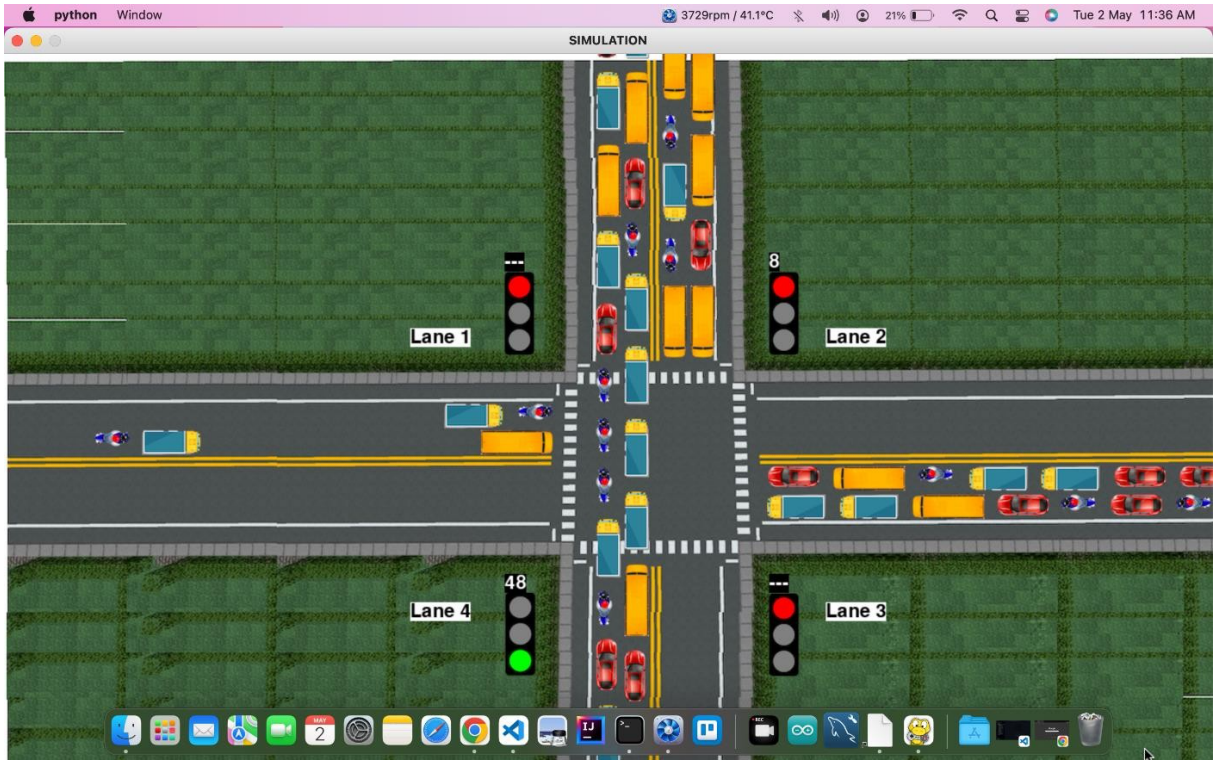


Figure 10.2.13 Fourth Road Signal

CHAPTER 11

REFERENCES

[1] Jyoti Tiwari, Real Time Traffic Management Using Machine Learning, Department of Information Technology, 2020.

[2] Xiaochen Hu, Beijing Signal and Information, A Moving Objects Based Real-time Defogging Method for Traffic Monitoring Videos. Aug 2014.

[3] Sunayana Jadhav, June 2020.Traffic Signal Management using Machine Learning Algorithm

[4] Prof Dr. A.Ravi, April 2021.Traffic Management System using Machine Learning Algorithm.

[5] Ritu, Traffic Management using Machine Learning, Department of Computing, Jan 2022.

[6] Li Xun, Nan Kaikai, Liu Yao , ZuoTao “A Real-Time Traffic Detection Method Based on Improved Kalman Filter” , 2018 3rd International Conference on Robotics and Automation Engineering (ICRAE) Guangzhou, China, 17-19 November 2018.

[7] Prof. S. T. Dumbre[1], Panchal Siddhi Santosh[2] Thakur Tejashri Chandrashekhar Smart traffic management system using machine learning, March. 2022.

[8] Zahra Zamani, Mohammad HosseinSaraee, Mahmoud Pourmand," Application of Data Mining in Traffic Management: Case of City of Isfahan", IEEE 2010.

[9] Celine Jacob, Baher, "Abdulhai Integrated Traffic Corridor Control Using Machine Learning", IEEE2005.

[10] Soufiene Djahel, Mazeiar Salehie, Irina Tal, Pooyan Jamshidi, "Adaptive Traffic Management for Secure and Efficient Emergency Services in Smart Cities", IEEE PerCom conference 2013.

[11] Abhijit Gadge1, Hardik Bhawsar2, Piyush Gondkar3, Smart Traffic Control System Using Deep Learning July 2021.

[12] Mohamad Belal Natafgi Department of Electrical and Computer Engineering, Smart Traffic Light System Using Machine Learning, 2018

[13] Li, X., Yang, Y., Li, Z., & Liu, Y. (2017). A deep learning approach for real-time traffic signal control. IEEE Transactions on Intelligent Transportation Systems, 18(11), 3060-3073.

[14] Yang, Q., Zhang, Y., Wang, F. Y., & Zheng, N. (2018). Traffic signal control for urban road networks using deep reinforcement learning. Transportation Research Part C: Emerging Technologies, 91, 296-316.

[15] Haddad, A., Cheung, C. Y., & Chan, A. H. (2019). A novel deep learning approach for real-time traffic signal control using a high-resolution camera. Transportation Research Part C: Emerging Technologies, 104, 145-162.

[16] Wang, J., & Meng, Q. (2016). A survey of traffic signal control methods. Transportation Reviews, 36(6), 748-776. 58

[17] Zhang, Y., Liu, S., & Wang, F. Y. (2019). Coordinated signal control for a large-scale road network using deep reinforcement learning. Transportation Research Part C: Emerging Technologies, 106, 345-366.

[18] Liao, Y., Zhang, L., & Yu, X. (2021). A survey on deep reinforcement learning for traffic signal control. Neural Computing and Applications, 33, 11675-11687.

[19] Ma, W., & Zheng, N. (2019). Urban traffic signal control: A deep learning approach. IEEE Transactions on Intelligent Transportation Systems, 20(3), 982-991.

[20] Zheng, H., Chen, J., Wang, Q., & Zhang, L. (2019). A reinforcement learning-based signal control method for a single intersection. IEEE Access, 7, 184626-184637.

[21] Wu, H., Zhang, Y., Wang, F. Y., & Zhang, Y. (2019). Coordinated signal control at multiple intersections using deep reinforcement learning. Transportation Research Part C: Emerging Technologies, 98, 1-16.

[22] Sun, C., Yang, Y., & Xu, Y. (2020). A survey of intelligent traffic signal control. IEEE Transactions on Intelligent Transportation Systems, 21(2), 766-783.

[23] Liu, H., Zhang, H., Wang, J., & Zou, H. (2019). Real-time traffic signal control for large-scale urban road networks using deep reinforcement learning. Transportation Research Part C: Emerging Technologies, 98, 175-193.

[24] Qian, K., & Wang, Y. (2021). Real-time traffic signal control for a single intersection with uncertain arrival rates using deep reinforcement learning. IEEE Transactions on Intelligent Transportation Systems, 22(1), 9-22.

[25] Kim, J., Jang, W., & Kim, K. (2018). Deep reinforcement learning-based traffic signal control considering multiple intersections. IEEE Access, 6, 52506-52515.

[26] Li, Y., Lin, H., Wu, J., & Li, J. (2019). Urban traffic signal control with deep reinforcement learning. Neurocomputing, 365, 121-133.

[27] Li, M., Qian, K., & Yang, L. (2021). Dynamic traffic signal control for urban intersections via a hybrid deep reinforcement learning method. Transportation Research Part C: Emerging Technologies, 124, 103096.

[28] Zhang, Y., Wang, F. Y., & Zheng, N. (2016). Deep learning for traffic prediction and intelligent transportation systems: A survey. IEEE Transaction on intelligent Transportation System, 17(12),3210-3223.