Graduation Project Book

PLVAR

Payment, License Verification and Authentication Robot

Under supervision of:

Assist. Prof. Hesham Ali Sakr
Assist. Lec. Magda Ibrahim

Team Members

Ahmed Yasser Arafat
Saad Nasser El-Baily
Mahmoud Hazem Salama
Mustafa Ahmed Hassanein
Ahmed Nasser Abdel-Khaleq
Mahmoud Muhammed Masherf
Youssef Eid Saad
Hadeer Reda Muhammed
Ebrahim Muhammed El-Zimiti
Alaa Abd Elnasser Tawfiq
Youssef Ebrahim Gammaz

Acknowledgement

First and foremost, we would like to praise and thank Allah, the almighty, who has granted countless blessing, knowledge, and opportunity to us, so that we have been Finally able to accomplish that success.

On the very outset of this acknowledgement, we would like to express our heartfelt gratitude to our professors and lecturers for their assistance in completing the project. I would like to extend my sincere & heartfelt obligation towards all the personages who have helped us in this endeavor. Without their active guidance, help, cooperation & encouragement, we would not have made headway in the project.

We are ineffably indebted to Assist. Prof. Hesham Sakr conscientious active guidance, cooperation and encouragement to accomplish our dream project.

We are extremely thankful and pay our gratitude to Assist. Prof. Magda Ibrahim for her valuable guidance and support on completion of this project in it's presently.

Table of Contents

Acknowledgement 2
Table of Contents 3
List of figures 6
List of tables 7
List of Abbreviations 8
Abstract 10
Problem Statement 11
Why PLVAR? 12
1 Chapter1: INTRODUCTION 13
1.1 Project Motivations: 13
1.2 Project Aims: 14
1.3 Project Contributions: 15
1.4 Introduction to AI: 15
2 Chapter 2: Economically Feasibility Study and Engineering Standards 17
2.1 Economic basis: 17
2.2 Safety basis: 17
2.3 Sustainability: 17
2.4 Ethical: 17
2.5 comparison between companies and it's cost: 18
2.5.1 comparison between companies: 18
2.5.2 the cost: 18
2.5.3 PLVAR Total cost Excluded 3D Printing: 19
3 Chapter 3: Related Work 20
4 Chapter4: System Design and Analysis 23
4.1 Hardware: 23
4.1.1 Introduction to Hardware: 23
4.1.2 The Hardware contents: 24
4.2 DataBase 42
4.2.1 Introduction to Firebase: 42
4.2.2 Working with Firebase: 43
4.2.3 Firebase tools we used: 43
4.2.4 Why Firebase? 44
4.2.5 Data structure: 47
4.2.6 Key points: 48

4.2.7 Store and sync data in real time 50
4.2.8 Key capabilities: 51
4.2.9 Secure your data: 53
4.2.10 Firebase Storage 53
4.2.11 Firebase Authentication - Simplify User Authentication for Your Apps: 55
4.2.12 Simplified Firebase Interaction with Pyrebase4 API 57
4.2.13 Advanced Features with Pyrebase4: 61
4.3 Mobile Application 63
4.3.1 Introduction: 63
4.3.2 History of Flutter: 64
4.3.3 Architecture of Flutter: 65
4.3.4 Key Features of Flutter: 66
4.3.5 Advantages of Flutter: 67
4.3.6 Disadvantages of Flutter: 68
4.4 AI 69
4.4.1 Voice&Speech-recognition 69
4.4.2 ALPR System: 72
4.4.3 Egyptian license plate recognition(ELPR) 74
4.4.4 Car Color Detection using Haar Cascade and Color Histogram Features 75
4.4.5 Methodology: 77
4.4.6 Experimental Setup and Results: 79
4.4.7 Color histogram features extraction: 81
4.4.8 PLVAR AI Model: 85
4.4.9 Seat-belt detection 85
4.4.10 Face Recognition: 90
4.4.11 GUI(Graphical User Interface): 96
5 Ch5: Result And Discussion 98
5.1 Programming : 98
5.1.1 C Programming: 98
5.1.2 Python : 99
5.2 Pictures from Database: 100
5.3 INRA-Application: 105
5.4 Project phases: 109
5.5 Design Robot: 115

5.5.1 Head: 115
5.5.2 Chest 116
5.5.3 Frame of The Screen 117
5.5.4 Cover of the chest 117
5.5.5 The Waist1 118
5.5.6 The Waist 2 118
5.5.7 The Base 119
5.5.8 Full Body 120
5.5.9 Actual Robot 121
6 Future work and Conclusion: 122
6.1 Future work: 122
6.2 Conclusion: 124
7 References 125
8 Appendix 131
8.1 Face recognition 131
8.1.1 Train model 131
8.1.2 Recognition 132
8.2 ELPER 134
8.3 Seat Belt Detection 135
8.4 car color recognition 137
8.5 GUI 138
8.6 DataBase Codes 145
8.7 Mobile Application 148
8.7.1 How to create homePage 148
8.7.2 Text 148
8.7.3 Image 148
8.7.4 Icon 148
8.7.5 tingNavigation and Rou 148
8.7.6 CurvedAnimation 149
8.7.7 Button 149
8.7.8 Create a state for RatingBox, _RatingBoxState byinheriting Stat <T> 149

List of Figures

Figure 4-1 Raspberry pi4 24
Figure 4-2 Raspberry Pi Board Details 25
Figure 4-3 Raspberry Pi 4 GBIO 27
Figure 4-4 Raspberry Pi 4 OS 27
Figure 4-5 Raspberry Pi Desktop 28
Figure 4-6 Touch Screen 14 inch 30
Figure 4-7 Internal Raspberry Pi Camera 31
Figure 4-8 External Camera 31
Figure 4-9 Barcode Scanner Sensor 32
Figure 4-10VNH2SP30 Monster Shield 33
Figure 4-11 Interfacing Motor with Vnh2SP30 Driver 34
Figure 4-12 12v Gear Motor 35
Figure 4-13 12v ,15A Battery 38
Figure 4-14 12v , 30A batteries connection 38
Figure 4-15 12v, 30A Power Supply 39
Figure 4-16 4015Step-Down Voltage(12v to 5v) 40
Figure 4-17 Wheels (foot of the robot) 41
Figure 4-18 Headphone and microphone 41
Figure 4-19 Firebase Emulator Suite1 45
Figure 4-20 Firebase Emulator Suite2 46
Figure 4-21 Firebase features and platforms are supported 47
Figure 4-22 JSON format 47
Figure 4-23KNN Algorithm 81
Figure 4-24 Difficult to separate with Hyper plane 82
Figure 4-25 Best Hyper plane 82
Figure 4-26 Comparing Yolo Model 83
Figure 4-27 MAP 83
Figure 4-28 Parameters(m) and LatencyA100 tensor RT FP16(ms/img) 84
Figure 4-29 Comparing between models 84
Figure 4-30iteration number 85
Figure 5-1 name of Database Structure 100
Figure 5-2 Database branch 100
Figure 5-3 Car license branch 101
Figure 5-4 branch information on the license plate1 101
Figure 5-5 branch informated on the car license plate2 101
Figure 5-6 branch information on the car license plate3 102
Figure 5-7 QR code scanning barcode 102
Figure 5-8 A picture showing the bar code that is present in the license 102
Figure 5-9 information about personal driving license1 103
Figure 5-10 information about personal driving license2 103
Figure 5-11 Road names 103

Figure 5-12 Road price information.....104
Figure 5-13 Pictures stored on the storage.....104
Figure 5-14 Select a photo from the photos stored on the storage.....104
Figure 5-15 Home page in English **Figure 5-16** Home page in Arabic.....105
Figure 5-17 Car ID in English **Figure 5-18** Car ID in Arabic.....105
Figure 5-19 Car Details in Englis **Figure 5-20** Car Details in Arabic 106
Figure 5-21 Fines Page in English **Figure 5-22** Fines Page in Arabic.....106
Figure 5-23 Road Page in English **Figure 5-24** Road Page in Arabic 107
Figure 5-25 Payment Methodes in English **Figure 5-26** Payment Method in Arabic 107
Figure 5-27 Settinges in English **Figure 5-28** Settinges in Arabic 108
Figure 5-29 Support in English **Figure 5-30** Support in Arabic 108
Figure 5-31 Start Check 109
Figure 5-32 License plate and color.....110
Figure 5-33seat belt.....111
Figure 5-34 Car plate 112
Figure 5-35 Scan your barcode 113
Figure 5-36 driver's data 114
Figure 5-37 Head **Figure 5-38** Head From another angle.....115
Figure 5-39 Chest1 it's has a Camera that scan the license.....116
Figure 5-40 chest From another angle 116
Figure 5-41 chest it's has a place where the moters are placed 116
Figure 5-42 Frame of the screen 117
Figure 5-43 Frame of the screen from another angle.....117
Figure 5-44 Cover of the chest.....117
Figure 5-45 Cover of the chest from another angle 117
Figure 5-46 The Waist 118
Figure 5-47 The waist from the another angle.....118
Figure 5-48 The waist2 from the another angle.....118
Figure 5-49 The waist2 118
Figure 5-50 The Base.....119
Figure 8-51 Full Body.....120
Figure 8-52 Full Body from the another angle 120
Figure 8-53 Actual Robot 121

List of Tables

Table 1 PLVAR Total cost Excluded 3D Printing 19
Table 2 Summary of technologies used for managing traffic.....21
Table 3 Implementation path 52
Table 4 Experimental Results 80

List of Abbreviations

Index	Description
CPU	Central Processing Unit
OS	Operating System
OCR	Optical Character Recognition
ELPER	Egyptian License Plate Recognition
AI	Artificial Intelligence
SW	Software
HW	Hardware
HDD	Hard disk drives
SSD	Solid-state drives
IT	Information technology
RAM	Random Access Memory
GPIO	General Purpose Input/output
HTML	Hyper Text Mark-up language
ALPR	Automatic License Plate Recognition
KNN	K-nearest neighbors algorithm
UI	User interface
CLAHE	Contrast Limited Adaptive Histogram Equalization
TTS	text-to-speech
SAML	Security Assertion Mark-up Language
CSS	Cascading Style Sheets
LXDE	Lightweight X11 Desktop Environment
IDE	Integrated Development Environment Explained
SDK	Software development kit
IAM	Identity Access Management
API	American Petroleum Institute
HTTP	Hypertext Transfer Protocol
CLI	Command-line interface
SQL	Structured Query Language
XML	extensible mark-up Language
ITS	Information Technology solutions
WSN	Wireless Sensor Network
VR	virtual reality
AR	Augmented reality
ASO	App store optimization
UX	User experience
PSUs	power supply units
VBA	Visual Basic for Applications
CAD	computer-aided design

FPS	frames per second
SSD	Solid-state drive
GPU	graphics processing unit
ANPR	Automatic Number Plate Recognition

Abstract

The concept of smart cities has become very popular recently. Due to increasing number of vehicles in urban areas, we are facing problems of traffic congestion. It is essential to develop an automated smart traffic system. The proposed approach is to digitalize highways and implement smart systems to address the identified challenges. The development of a smart robot equipped with artificial intelligence (AI) and cutting-edge technology is suggested as part of this initiative. The robot would perform various tasks related to traffic management, such as verifying personal licenses of drivers, checking vehicle registration and licenses, identifying wanted individuals, monitoring seat belt usage and mobile phone usage while driving, and enabling electronic payment of highway fees. The tools and techniques used would include AI algorithms, image recognition technology, data analysis, and mobile application development. Commitment to designing the entire project according to **the IEEE SA - IEEE 1857.10-2021 voice +video, IEEE SA global standard Programming, IEEE CertifAIEd and AI.**

Problem Statement

Contemporary cities grapple with urgent challenges, such as escalating traffic congestion, environmental pollution, frequent accidents, and widespread traffic violations. These issues exert a detrimental effect on the efficiency and safety of transportation systems within cities.

Why PLVAR?

- Simplicity checking the data.
- The process takes short time to be done.
- Ways to pay the fees is multiplied.
- Mobile app is the simple to use.
- The interface of the user is easy for user to understand.

1 Chapter1: INTRODUCTION

1.1 Project Motivations:

1. Convenience and Efficiency: Your system aims to revolutionize the way driver licenses are verified and payments are processed. By streamlining these processes, you are enhancing the convenience and efficiency for both drivers and authorities. This can save valuable time and effort for everyone involved.
2. Road Safety: Verifying driver licenses and identifying any violations are crucial for maintaining road safety. Your system's ability to cross-reference licenses with a database helps ensure that only qualified and authorized drivers are on the road. By contributing to improved license verification, you are actively working towards reducing accidents and promoting safer driving practices.
3. Financial Transparency: With the secure payment processing capabilities of your system, you are providing a transparent and reliable way for drivers to pay their fees. This can help in reducing financial discrepancies and enhancing accountability in the payment process.
4. Technological Advancement: Your utilization of cutting-edge technology, such as the Raspberry Pi kit and advanced camera, showcases innovation and pushes the boundaries of what is possible in license verification and payment processing. By staying at the forefront of technological advancements, you are contributing to the progress of the field and driving innovation in the industry.
5. Data Security: Protecting personal and financial data is of utmost importance in today's digital age. By prioritizing the security of the information stored in your database and utilizing the robustness of Firebase servers, you are providing drivers with the peace of mind that their data is safe and secure.

6. Positive User Experience: Your focus on creating a highly intuitive and user-friendly mobile app ensures that drivers can easily navigate through the payment process and obtain the necessary information. By prioritizing user experience, you are enhancing customer satisfaction and making a positive impact on people's lives.

1.2 Project Aims:

1. Proposing an approach for highway digitization.
2. Developing smart ways to keep up with modern technology.
3. Personal License Verification: This module would leverage AI and image recognition technology to verify the validity of drivers' personal licenses.
4. Vehicle Registration and License Check: This module would check the registration and license of cars to ensure compliance with regulations.
5. Criminal Identification and Reporting: Using AI algorithms, this module would identify individuals who are criminally wanted and promptly report them to the appropriate authorities.
6. Driver Monitoring: This module would monitor drivers to ensure they are wearing seat belts and not using mobile phones while driving. It could utilize computer vision techniques or sensorbased systems to detect such behaviors.
7. Electronic Payment System: A mobile application would be designed to facilitate electronic payment of highway fees, reducing paper waste and alleviating traffic congestion. This module would involve mobile application development and integration with payment gateways.
8. Elevating Security Measures: With the integration of smart robotics, security becomes a paramount aspect of guest satisfaction. These intelligent machines can bolster security protocols by performing

tasks such as verifying personal licenses, monitoring access points, and identifying any unauthorized individuals. By proactively addressing security concerns, guests can enjoy peace of mind throughout their stay.

1.3 Project Contributions:

1. Contribute to the achievement of digital transformation.
2. Create smart ways to keep up with modern technology.
3. Create an electronic payment system.
4. Ensure that the driver's personal license matches and report the criminally wanted persons.
5. Ensure the validity of the car license, report the violated and stolen cars, and ensure that the cars meet the legal specifications.
6. Accuracy of the detecting the car plates.
7. Database is the storage on server with real time connection.
8. Secure database as the data will be encryption end to end.
9. Error rate is about to be immune.
10. Proposing an approach for highway digitization.

1.4 Introduction to AI:

Artificial Intelligence (AI) refers to the development of intelligent machines that can perform tasks that typically require human intelligence. It involves creating computer systems capable of simulating human thinking, learning, problem-solving, and decision-making processes.

AI encompasses a range of technologies and techniques, including machine learning, natural language processing, computer vision, and robotics. These

tools enable AI systems to process and analyze vast amounts of data, recognize patterns, and make predictions or take actions based on the acquired knowledge.

The primary objective of AI is to create machines that can perceive and understand the world, adapt to changing circumstances, and interact with humans in meaningful ways. This field has the potential to revolutionize various industries by automating tasks, enhancing efficiency, and enabling new possibilities.

AI has already achieved significant milestones, such as defeating human champions in strategic games, enabling self-driving cars, enhancing medical diagnoses, and providing personalized recommendations. However, it also presents challenges and ethical considerations, including privacy, bias, and societal impact.

While the pursuit of artificial general intelligence (AGI), which mirrors human intelligence across a broad spectrum, remains a long-term goal, AI continues to advance rapidly, transforming the way we live and work. By harnessing the power of intelligent machines, we can unlock new frontiers, solve complex problems, and shape a more intelligent and interconnected future.

2 Chapter 2: Economically Feasibility Study and Engineering Standards

Nowadays, AI is used in a wide range of applications across various industries. It includes virtual assistants and chatbots, image and speech recognition, natural language processing, data analysis, and decision-making systems. AI is also used in healthcare, finance, transportation, manufacturing, and other industries to improve efficiency, accuracy, and automation of various tasks. Additionally, AI is used in scientific research and development, including drug discovery and genomics.

2.1 Economic basis:

1. AI-based triage systems can help in reducing the work burden of traffic employees' by automating several processes such as personal license verification, vehicle registration and license check, criminal identification and reporting and electronic payment system.
2. Save the time it takes to update traffic licenses.

2.2 Safety basis:

1. Ensure that the driver's personal license matches and report the criminally wanted persons.
2. Report the violated and stolen cars, and ensure that the cars meet the legal specifications.
3. Early and rapid diagnosis of the virus, which reduces the chances of developing the patient's infection and entering the intensive care.
4. Ensure that the seat belt is present

2.3 Sustainability:

The project was provided with solar cell units to ensure the continuity of the robot's operation and to charge the batteries.

2.4 Ethical:

Commitment to designing the entire project according to **the IEEE SA - IEEE 1857.10-2021 voice +video, IEEE SA global standard Programming, IEEE CertifAIEd and AI.**

2.5 comparison between companies and it's cost:

2.5.1 comparison between companies:

1. Knightscope: Knightscope is a company that develops and manufactures autonomous security bots. Their bots are used to patrol and monitor areas, collect data, detect anomalies, and provide real-time alerts. Knightscope robots range in cost from tens of thousands to hundreds of thousands of dollars, depending on the model and features.

2. Cobalt Robotics: Cobalt Robotics offers autonomous security robots designed for commercial spaces. These robots are equipped with various sensors and cameras to monitor and patrol areas, detect intrusions, and provide situational awareness.

3. PAL Robotics: PAL Robotics develops humanoid robots that can be used for security purposes in various environments. Their robots are capable of navigation, person detection, and human-robot interaction. Pricing details of PAL Robotics robots can be obtained through their website.

4. SoftBank Robotics: SoftBank Robotics is best known for its humanoid robot named Pepper, which can be customized for various applications, including security and customer service. Pepper bots can interact with people, provide information, and perform basic security tasks. The cost of Pepper robots varies depending on the customization and additional features required.

2.5.2 the cost:

1. Entry-level autonomous security robots: These robots typically have basic surveillance and monitoring capabilities, and their prices can range from $10,000 to $30,000.

2. Mid-range autonomous security robots: These robots often come with more advanced features such as facial recognition, anomaly detection, and remote control capabilities. Prices for mid-range robots can range from $30,000 to $100,000.

3. High-end autonomous security robots: These robots offer advanced functionalities like autonomous navigation, artificial intelligence algorithms, and comprehensive security features. The prices for high-end robots can exceed $100,000 and can go up to several hundred thousand dollars

2.5.3 PLVAR Total cost Excluded 3D Printing:

Table 1 PLVAR Total cost Excluded 3D Printing

Components	Cost
Raspberry pi	$ 160.00
Camera	$ 50.00
Barcode Scanner	$ 30.00
Gear motor	$ 100.00
Two batteries	$ 50.00
power supply	$ 10.00
step down DC-DC	$ 3.50
4 plastic wheels	$ 30.00
headphones & mic	$ 8.00
gear box	$ 30.00
Touch screen	$ 85
3d Printer	$ 300.00
two iron axel	$ 50.00
High Current driver	$ 20
Total	$ 926.5

3 Chapter 3: Related Work

Congestion of traffic is a key problem faced in a majority of metro cities, especially in the developing world. Traffic congestion comprises of queues, reduced speeds, and increased travel durations, which fatigues commuters and results in stress, thereby bringing down productivity and creating intangible societal expenses [1]. In addition, different factors, such as use of natural resources, surroundings, commuter safety, etc., are affected in either a direct or an indirect manner. Thus, congestion of traffic poses a challenge to every growing city. With the increasing congestion, the construction of new roads is seldom fruitful because of various circumstances. Contrarily, constructing new roads could often add to congestion by creating more demands for motor travel, thereby depleting the added capacity rapidly [2]. This is getting increasingly evident as congestions with delays are being noticed in big cities. Thus, several mitigation measures are being implemented through the years. The increasing challenges associated with traffic congestion attracted the attention of academicians, who have proposed numerous techniques for managing traffic and solving the congestion issue. Such solutions range from the latest data collecting equipment with wireless sensor network (WSN) protocols [3], [4] to route schemes [5], [6] and traffic forecasting methods [7]. Several researchers were interested in the application of Intelligent Transportation Systems (ITS) to manage traffic management issues. [8] proposed city traffic prediction models using varied modelling methods, one based on the network traffic flow propagation, and the other based on the time varying spare flow volume on the related road link. This paper investigates the technologies directed towards developing novel systems that address many of the traffic congestion problems. The findings of these researchers indicated that roads could be safer and greener with enhanced efficiency through the application of emerging ITS technologies

[9]. However, several challenges are encountered during the implementation of ITS. Studies pointed out that effective control and management of traffic and accurate prediction of travel time are the two major challenges faced in the application of ITS. However, despite the emphasis on these challenges, only a few researchers have rendered solutions based on the functionality of ITS [10], [11]. In addition, only few papers have compared the benefits and shortcomings of the suggested solutions [12]–[14]. The strategies used for traffic management are summarized in Table 1.

Table 2 Summary of technologies used for managing traffic

Ref.	Technology	Principles	Advantages	Disadvantages
[15]	Machine learning and big data	Data Collection	Simple and economical	Accuracy and reliability issues when traffic volumes and density fluctuate
[16]	Multiple mobile Cameras	Use of Probabilistic Data Collection and GREEDY algorithm for collecting data, and analysis of collected images	Coverage more uniform, consumes less network bandwidth	Does not consider cooperative storage among vehicles and image replications
[17]	WSN	Two anchor nodes placed at fixed distance. A symmetric double-sided two-way ranging algorithm gives location of vehicle	Economical, robust, and requires lesser computation	Errors when traffic is heavy
[18]	Wi-Fi	Two WiFi transceivers capture data. A convolutional neural network and principal component analysis-based algorithm captures the vehicles in the data	Reduced energy consumption, minimal connectivity delay and reduced interference from nearby intersections	Not suitable where traffic is heavy
[19]	Edge Computation	An edge-computing equipment uses deep neural networks and computer vision for tracking traffic in real time. The interoperable Agnosticism framework gathers, stores, and manages data from different sensors	Multi-modal detection, privacy compliance, scalable, and interoperable	Operational issues in low light and bad weather
[20]	Markov Decision Process and Optimal VSN Data Forwarding	A road network graph model captures the vehicle effects having predetermined trajectories and creates an easy routing problem formulation	Higher packet delivery ratio, reduced delay	Errors when traffic is Heavy
This paper	AI technologies	A proposed model detects the driver's and vehicle's features with more accuracy performance	Higher performance, applicable for many applications and ability for using in various conditions	Higher requirements needed

To solve these problems, we propose a smart robot which we named PLVAR (Payment, License Verification and Authentication Robot) that can perform multiple tasks related to traffic management based on artificial intelligence (AI) technologies such as:

- Checking the personal licenses of car drivers and validity.
- Checking the car license and registration.
- Finding criminally wanted persons and report them immediately.
- Make sure that the driver wears a seat belt and does not use a mobile phone while driving.
- Monitor traffic flow and adjust the time period of traffic lights according to this.
- Adjust the maximum speed of roads to cope with traffic situation.

Also, a mobile application is presented which enable paying highway fees electronically to reduce paper waste and traffic jams. The paper's contributions can be briefed as follows:

1. Proposing an approach for highway digitization.
2. Developing smart ways to keep up with modern technology.
3. Developing an electronic payment system.
4. Ensuring that the driver's personal license matches and report the criminally wanted persons.
5. Ensuring the validity of the car license, report the violated and stolen cars, and ensure that the cars meet the legal specifications.

4 Chapter4: System Design and Analysis

4.1 Hardware:

4.1.1 Introduction to Hardware:

Hardware refers to the physical components that make up a computer system. It includes devices such as the central processing unit (CPU), memory modules, storage devices, input and output devices, and more.

The CPU serves as the "brain" of the computer, executing instructions and performing calculations. It directly impacts the system's speed and overall performance.

Memory modules, like Random Access Memory (RAM), provide temporary storage for data and instructions that the CPU accesses quickly. RAM plays a vital role in multitasking and system performance.

Storage devices, such as hard disk drives (HDD) and solid-state drives (SSD), store data for long-term use. They enable users to store and retrieve various types of information.

Input devices, like keyboards, mice, and touchscreens, allow users to interact with the computer system by providing input and commands. Output devices, such as monitors and printers, display or produce output from the system.

Understanding hardware is crucial for individuals interested in computer systems, whether for personal use, programming, or IT support. Each hardware component contributes to the overall functionality and performance of the computer system, working together to enable its various operations and tasks.

4.1.2 The Hardware contents:

1. Raspberry Pi 4 & OS.
2. Touch Screen 14 inch.
3. Cameras.
4. Arduino UNO.
5. Barcode Scanner.
6. Monster Shield high current motor driver (VNH2SP30 30 amp).
7. Two Gear Motor 12v, 5A.
8. Two Batteries 12v, 15A.
9. Power Supply 12v, 30A.
10. Step Down DC-DC 12V to 5v, 5A.
11. 4 Plastic wheels.
12. Headphone and microphone.
13. Rubber belt.
14. Two Iron axel.

4.1.2.1 Raspberry Pi 4 & OS:

Figure 4-1 Raspberry pi4

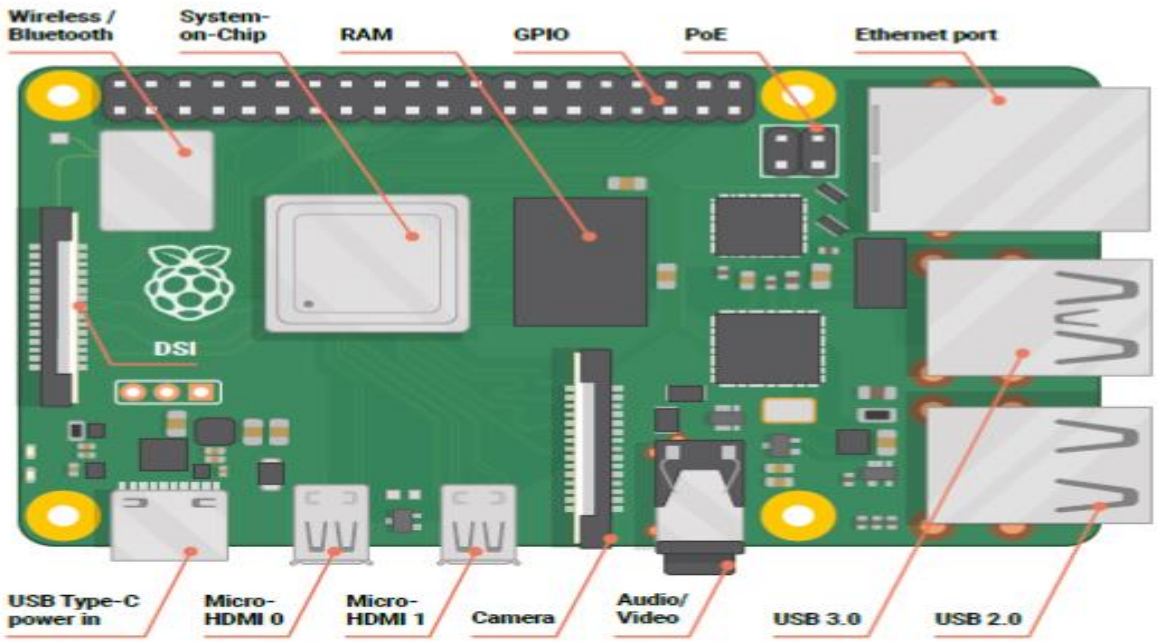


Figure 4-2 Raspberry Pi Board Details

Raspberry Pi 4 is a popular single-board computer developed by the Raspberry Pi Foundation. Here are some key points about the Raspberry Pi 4 with 4GB of RAM:

1. Performance: The Raspberry Pi 4 is a significant upgrade over its predecessors in terms of performance. It features a powerful 1.5GHz quad-core ARM Cortex-A72 CPU, which is capable of delivering impressive computing power for its size and price.

2. Memory: The Raspberry Pi 4 is available in different RAM configurations, including 4GB. Having 4GB of RAM allows for smoother multitasking and better performance when running memory-intensive applications or using the Pi as a desktop computer.

3. Connectivity: The Raspberry Pi 4 comes with enhanced connectivity options. It features dual-band Wi-Fi (2.4GHz and 5GHz), Gigabit Ethernet, Bluetooth 5.0, and USB 3.0 ports. These improvements make it easier to connect to networks and peripherals and enable

faster data transfer.

4. Video and Display: The Raspberry Pi 4 supports dual-monitor setups with resolutions up to 4K, providing a more versatile and immersive computing experience. It has two micro HDMI ports, allowing you to connect two displays simultaneously.

5. GPIO Pins: Like its predecessors, the Raspberry Pi 4 retains the GPIO (General Purpose Input/Output) pins, which allow you to connect and control various external devices and sensors, making it suitable for a wide range of projects.

6. Operating Systems: The Raspberry Pi 4 supports a variety of operating systems, including Raspbian (officially renamed Raspberry Pi OS), Ubuntu, and other Linux distributions. This flexibility allows you to choose an OS that best suits your needs and project requirements.

7. Applications: The Raspberry Pi 4 is a versatile platform suitable for various projects, such as home automation, robotics, media centers, retro gaming consoles, web servers, and more. Its compact size and low power consumption make it ideal for embedded systems and IoT applications as well.

Overall, the Raspberry Pi 4 with 4GB of RAM provides a significant boost in performance and features compared to previous models. It offers increased memory capacity, improved connectivity options, and supports higher-resolution displays, making it a popular choice for hobbyists, developers, and enthusiasts alike.

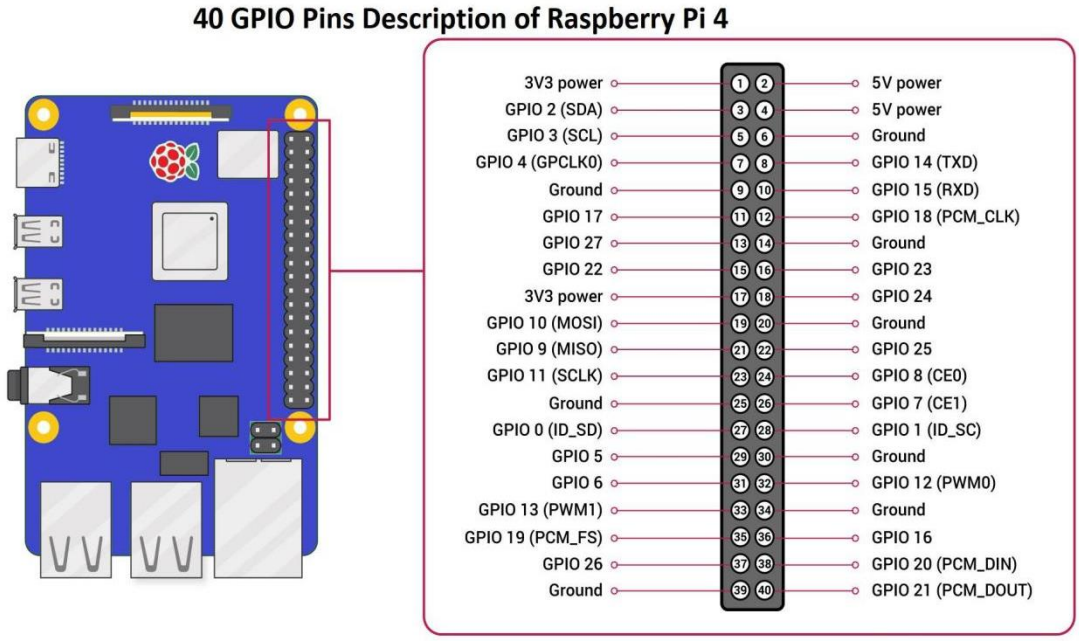


Figure 4-3 Raspberry Pi 4 GBIO

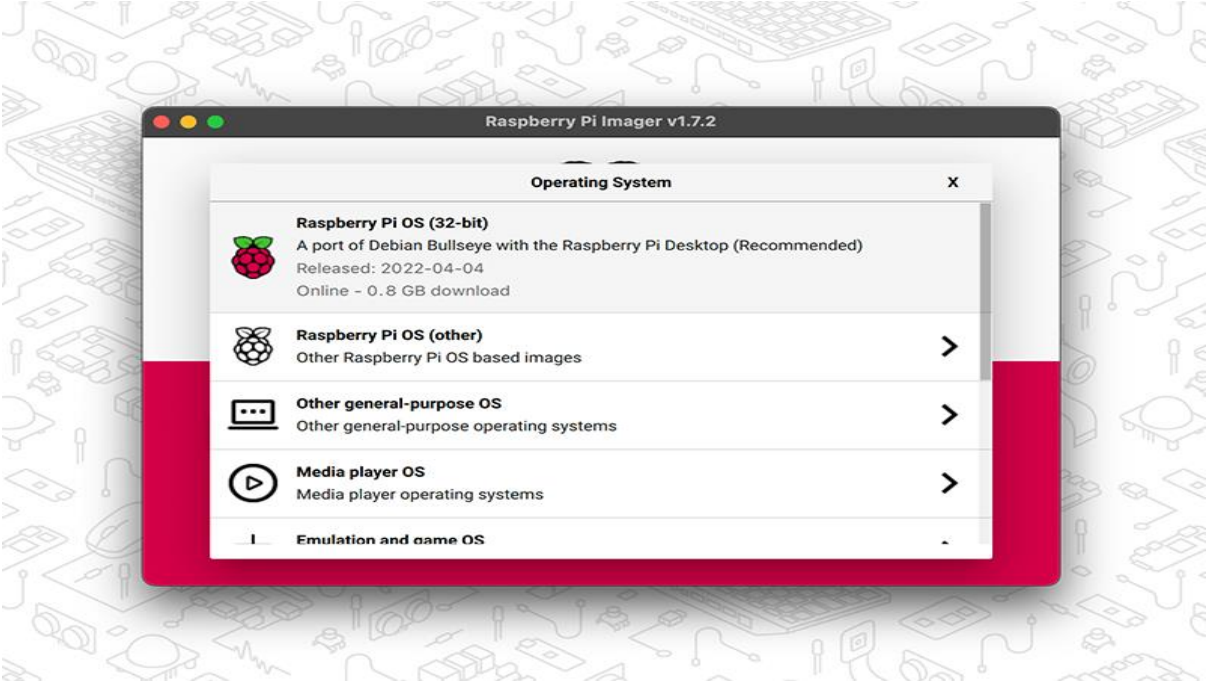


Figure 4-4 Raspberry Pi 4 OS

Figure 4-5 Raspberry Pi Desktop

Raspberry Pi OS (formerly known as Raspbian) is the official operating system developed by the Raspberry Pi Foundation specifically for the Raspberry Pi single-board computers. Here are some key points about the 32-bit version of Raspberry Pi OS:

1. Architecture: The 32-bit version of Raspberry Pi OS is designed to run on ARM-based processors, which are the processors used in Raspberry Pi boards. It is optimized to take full advantage of the capabilities of the Raspberry Pi hardware.

2. Debian-based: Raspberry Pi OS is based on the popular Debian operating system, specifically tailored for the Raspberry Pi. It inherits the stability, reliability, and extensive package repositories of Debian, making it a versatile and widely supported operating system.

3. Desktop Environment: Raspberry Pi OS comes with a user-friendly desktop environment based on the lightweight LXDE

(Lightweight X11 Desktop Environment). It provides a familiar and easy-to-use interface, making it suitable for beginners and those transitioning from other desktop operating systems.

4. Preinstalled Software: Raspberry Pi OS includes a variety of preinstalled software, including the Chromium web browser, LibreOffice suite, Python programming environment, and the Thonny Python IDE. These applications enable users to browse the web, work with office documents, and get started with programming right out of the box.

5. GPIO and Software Support: Raspberry Pi OS provides extensive support for the GPIO (General Purpose Input/Output) pins on the Raspberry Pi board. It includes libraries and utilities that make it easy to control and interface with external devices and sensors using programming languages like Python.

6. Software Updates: Raspberry Pi OS receives regular updates from the Raspberry Pi Foundation, ensuring security patches, bug fixes, and performance improvements. These updates can be easily installed using the built-in package management system.

7. Compatibility: Raspberry Pi OS is compatible with a wide range of software and applications available for the Raspberry Pi ecosystem. It supports popular projects and software frameworks like RetroPie for retro gaming, Kodi for media centers, and Pi-hole for network-wide ad-blocking.

8. Community and Support: The Raspberry Pi OS community is vibrant and active, with a wealth of online resources, forums, and tutorials available for users. The official Raspberry Pi website provides comprehensive documentation and forums where users can seek assistance and share their projects and experiences.

Overall, Raspberry Pi OS 32-bit edition provides a user-friendly and optimized operating system specifically tailored for the Raspberry Pi boards. It offers a stable and versatile platform for a wide range of projects, from basic desktop computing to embedded systems and IoT applications.

4.1.2.2 Touch Screen 14 inch:

Figure 4-6 Touch Screen 14 inch

4.1.2.3 Cameras:

Figure 4-7 Internal Raspberry Pi Camera

Figure 4-8 External Camera

4.1.2.4 Barcode Scanner:

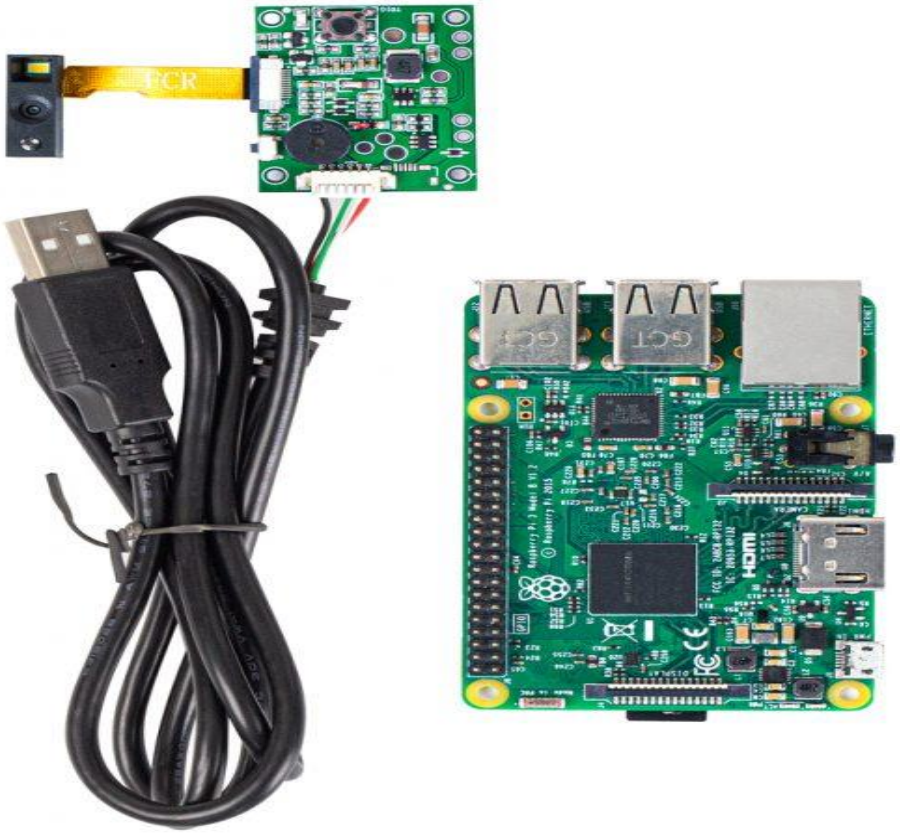


Figure 4-9 Barcode Scanner Sensor

4.1.2.5 Monster Shield high current motor driver (VNH2SP30 30 amp):

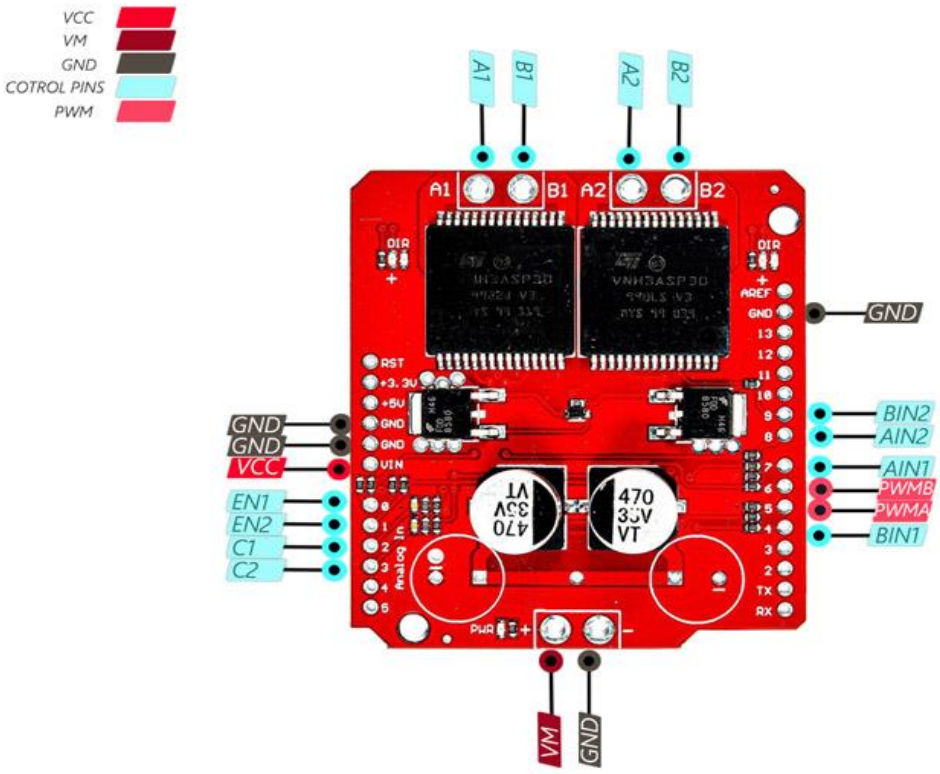


Figure 4-10VNH2SP30 Monster Shield

The VNH2SP30 / Monster Motor Features:

- Voltage max: 16V
- Maximum current rating: 30 A
- Practical Continuous Current: 14 A
- Current sensing available to Arduino analog pin
- MOSFET on-resistance: 19 mΩ (per leg)
- Maximum PWM frequency: 20 kHz
- Thermal Shutdown: Undervoltage and Overvoltage shutdown

The VNH2SP30 Monster Motor Shield is a motor driver shield designed for Arduino boards. It is specifically built to control high-power DC motors, making it suitable for various robotics and automation projects. The shield utilizes the VNH2SP30 motor driver chip, which can handle currents up to 30A and voltages up to 16V.

The VNH2SP30 Monster Motor Shield provides dual-channel motor control, allowing you to independently control the speed and direction of

two DC motors. It incorporates a robust H-bridge circuitry that enables bidirectional control, making it suitable for applications requiring forward and reverse movement.

The shield features built-in protection mechanisms, including over-temperature and over-current protection, which helps safeguard the motors and the driver circuitry. It also includes flyback diodes to protect against voltage spikes generated when the motor is turned off.

The VNH2SP30 Monster Motor Shield can be easily stacked on top of an Arduino board, providing a convenient and compact solution for motor control. It offers several control options, including analog control through pulse-width modulation (PWM) signals or digital control using the Arduino's digital pins.

Overall, the VNH2SP30 Monster Motor Shield is a powerful motor driver that provides reliable and efficient control for high-power DC motors, making it a popular choice among hobbyists, robotics enthusiasts, and professionals working on motor-driven projects.

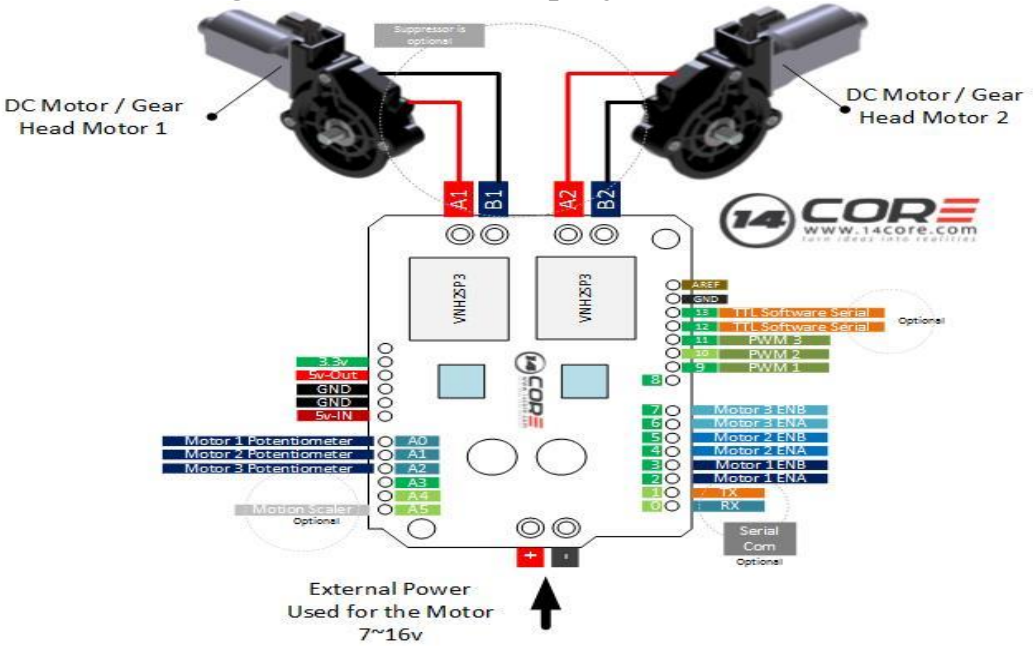


Figure 4-11 Interfacing Motor with Vnh2SP30 Driver

4.1.2.6 Two Gear Motor 12v, 5A:

Figure 4-12 12v Gear Motor

The gear motors you mentioned are DC motors equipped with a gearbox. Gear motors are commonly used in various applications that require high torque and precise control. Here's some information about these motors:

1. Voltage: The gear motors operate at 12 volts, which is a standard voltage used in many electrical systems. It's important to ensure that the power supply and motor driver you are using are compatible with this voltage.

2. Current: The motors have a current rating of 5A, indicating the maximum current they can draw during operation. It's crucial to use a motor driver, such as the VNH2SP30 Monster Motor Shield, that can handle this current rating to ensure safe and efficient motor control.

3. Gearbox: Gear motors are equipped with a gearbox, which consists of a set of gears that transmit power from the motor to the output shaft. The gearbox provides torque multiplication and speed reduction, allowing the motor to deliver higher torque at lower speeds.
4. Gear Ratio: Gear motors have a specific gear ratio, which represents

the number of gear teeth on the driving gear divided by the number of gear teeth on the driven gear. This ratio determines the speed and torque output of the motor. Different gear ratios are available to suit different application requirements.

5. Mounting: Gear motors typically come with mounting brackets or flanges that allow easy installation in various setups. The mounting mechanism ensures secure attachment and proper alignment with other components or machinery.

6. Control: The VNH2SP30 Monster Motor Shield enables you to control the speed and direction of the gear motors. By sending appropriate signals from an Arduino or compatible microcontroller, you can adjust the motor speed and change the direction of rotation to suit your application needs.

4.1.2.7 Two Batteries 12v, 15A:

Lithium batteries offer several advantages over other types of batteries. Here are some of their key advantages:

High Energy Density: Lithium batteries have a high energy density, meaning they can store a large amount of energy relative to their size and weight. This makes them ideal for applications where compactness and lightweight design are important, such as portable electronics, electric vehicles (EVs), and renewable energy systems.

Longer Cycle Life: Lithium batteries typically have a longer cycle life compared to other rechargeable battery chemistries. A cycle refers to one complete charge and discharge cycle. Lithium batteries can endure hundreds to thousands of cycles before their capacity significantly degrades. This longevity makes them suitable for long-term use, reducing the need for frequent battery replacements.

High Discharge Rate: Lithium batteries have a high discharge rate, which

means they can provide a significant amount of power quickly. This feature is crucial for devices that require a sudden surge of energy, like electric power tools or electric vehicles that need to accelerate rapidly.

Low Self-Discharge: Lithium batteries have a lower self-discharge rate compared to other rechargeable batteries. Self-discharge refers to the loss of battery capacity when not in use. Lithium batteries can retain their charge for months without significant capacity loss, making them convenient for devices that may sit idle for extended periods.

No Memory Effect: Unlike some older battery technologies, lithium batteries do not suffer from memory effect. Memory effect occurs when a battery gradually loses capacity if it is repeatedly recharged without being fully discharged. Lithium batteries can be recharged at any time without negatively impacting their overall capacity or performance.

Lightweight and Compact: Lithium batteries are lightweight and can be designed in various shapes and sizes, providing flexibility in product design. This advantage is especially beneficial for portable electronic devices, as it allows for sleek and slim designs while still delivering adequate power.

Wide Range of Applications: Lithium batteries are widely used in numerous applications, including smartphones, laptops, digital cameras, cordless power tools, electric vehicles, renewable energy storage, and many more. The versatility of lithium batteries makes them suitable for various industries and consumer products.

Figure 4-13 12v ,15A Battery

Now we will connect two batteries in parallel to get 30A:

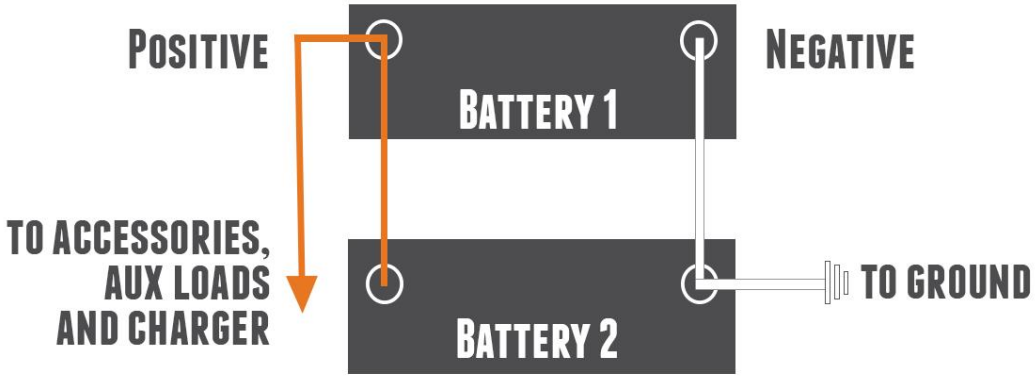


Figure 4-14 12v , 30A batteries connection

It is the main power source for the robot, through which the entire robot parts are operated.

4.1.2.8 Power Supply 12v, 30A

We will use the power Supply to charge the batteries in the event that they are finished, as it converts 220v alternating current into 12v direct current with 30A

Figure 4-15 12v, 30A Power Supply

Charging Circuit:

4.1.2.9 Step Down DC-DC 12V to 5v, 5A:

We will use this circuit to reduce the voltage from 12 volts to 5 volts to feed the Raspberry Pi. This circuit was used specifically, and we did not use normal regulator because the regulator cannot withstand high current, while this circuit can bear up to 5 amps.

Figure 4-16 4015Step-Down Voltage(12v to 5v)

4.1.2.10 Plastic wheels:

The wheels, which are used to move the robot in all directions on the ground, meaning the feet of the robot.

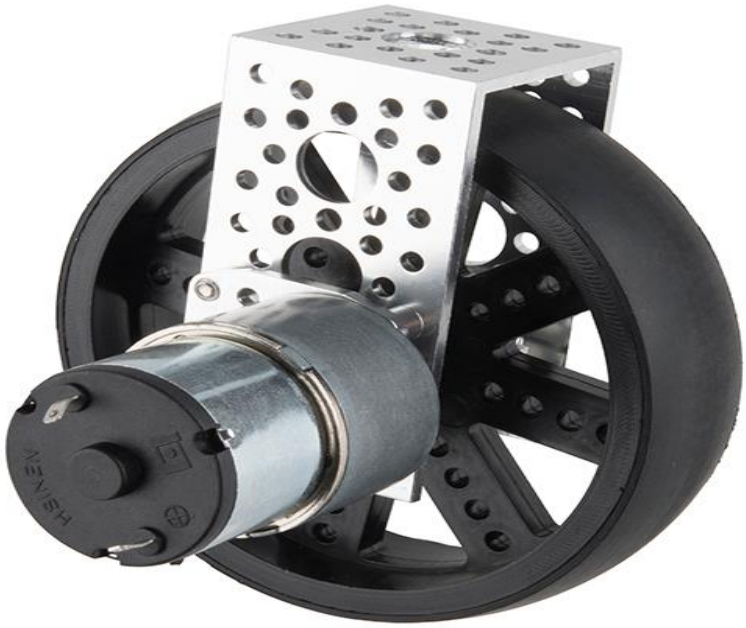


Figure 4-17 Wheels (foot of the robot)

4.1.2.11 Headphone and microphone:

We will use the microphone and the speaker to make the robot interact with people so that it responds to their questions when someone asks a question.

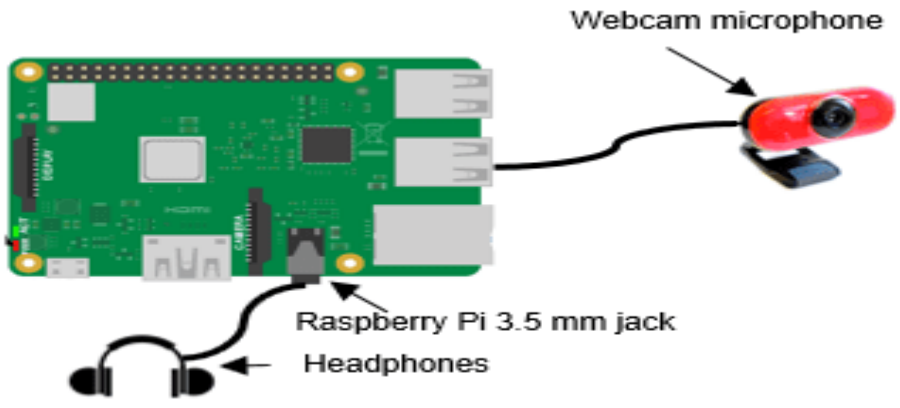


Figure 4-18 Headphone and microphone

4.2 DataBase

4.2.1 Introduction to Firebase:

Firebase is a mobile and web application development platform that provides a suite of services to help developers build high-quality apps quickly and easily. It was initially developed by Firebase, Inc. and was later acquired by Google in 2014.

Firebase offers a wide range of tools and services that can be used to develop various types of applications, including mobile apps, web apps, and even games. Some of the core features of Firebase include Realtime Database, Authentication, Cloud Storage, Cloud Messaging, and Analytics.

Firebase Realtime Database is a cloud-hosted NoSQL database that allows developers to store and sync data in real time. Firebase Authentication provides a simple and secure way to allow users to sign up, sign in, and manage their profiles. Firebase Cloud Storage allows developers to store and retrieve user-generated content such as images and videos. Firebase Cloud Messaging allows developers to send push notifications and in-app messages to users. Firebase Analytics provides insights into user behavior, app performance, and more.

Firebase is known for its ease of use and scalability, making it a popular choice among developers. It also offers a free tier with a generous quota that can be used for small projects or during development. Additionally, Firebase has extensive documentation and a large online community, making it easy for developers to get help and find resources.

Overall, Firebase is a powerful platform that can help developers build high-quality applications quickly and easily.

Firebase includes analytics, authentication, performance monitoring, messaging, crash reporting and much more. Because it is a Google product, there is also integration into a lot of other products. This includes integration with Google Ads, AdMob, Google Marketing Platform, the Play Store, Data Studio, BigQuery, Slack, Jira, and more.

The Firebase APIs are packaged into a single SDK that can be expanded to multiple platforms and languages. This includes C++ and Unity, which are both popular for mobile development.

4.2.2 Working with Firebase:

A Firebase project is a pool of resources that can include a database as well as items such as user accounts, analytics, and anything that can be shared between a number of client applications. A Firebase application is a single application that can be backed by the Firebase Project. A Firebase project can have multiple Firebase applications within it.

4.2.3 Firebase tools we used:

1. Real time Database: a cloud-hosted database. Data is stored as JSON and synchronized in real time to every connected client.
2. Storage: The ability to upload and use files to Google's cloud. Basically, we are able to upload our webpage assets and use them as if they were hosted in our own server.
3. Authentication: This is a token-based authentication system that provides easy integration with most of the platforms.
4. Pyre Base 4: A simple python wrapper for the Firebase API.

4.2.4 Why Firebase?

- All in one platform
- Easy to store Data & Free
- Scalability
- Reliability
- Security(end to end development)
- Fast and safe Hosting

Firebase Local Emulator Suite:

The Firebase Local Emulator Suite is a set of advanced tools for developers looking to build and test apps locally using Cloud Firestore, Realtime Database, Cloud Storage for Firebase, Authentication, Firebase Hosting, Cloud Functions (beta), Pub/Sub (beta), and Firebase Extensions (beta). It provides a rich user interface to help you get running and prototyping quickly.

Local development with Local Emulator Suite can be a good fit for your evaluation, prototyping, development and continuous integration workflows.

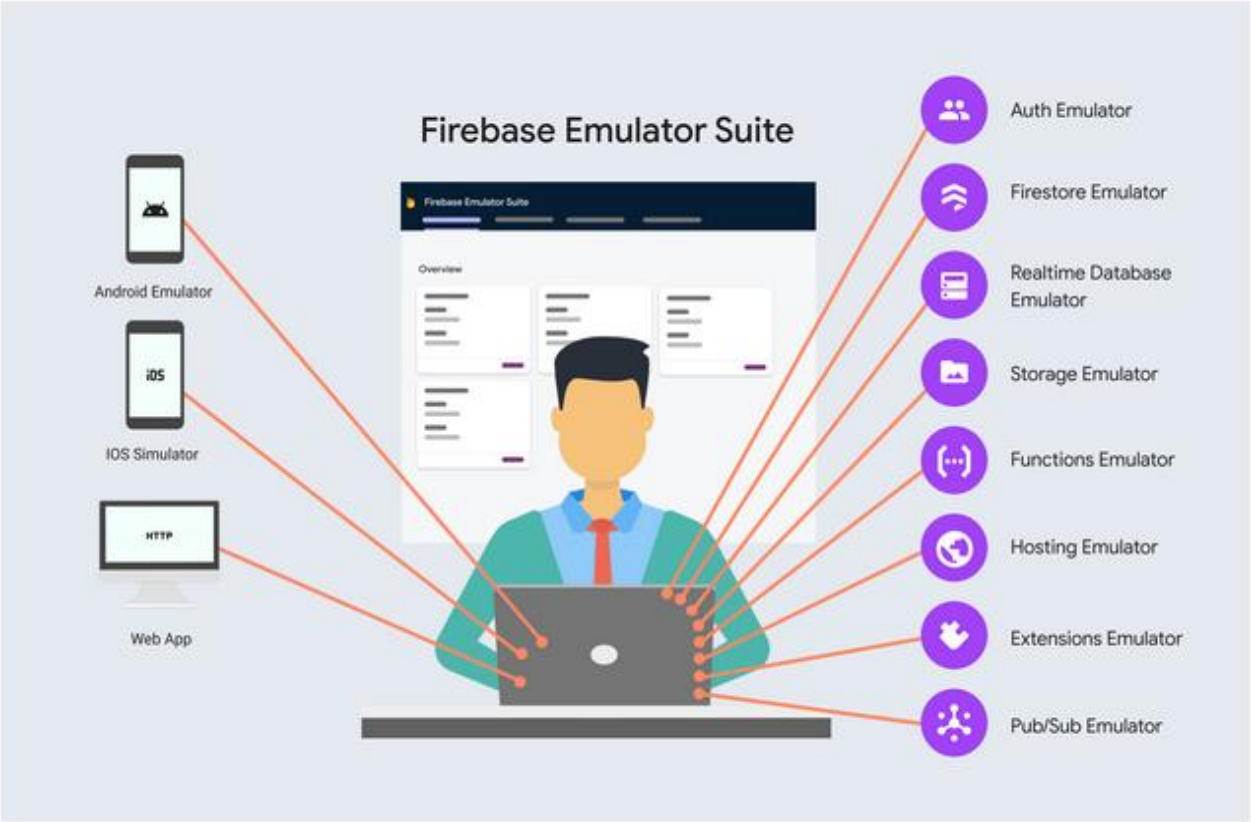


Figure 4-19Firebase Emulator Suite1

What is Firebase Local Emulator Suite?

The Firebase Local Emulator Suite consists of individual service emulators built to accurately mimic the behavior of Firebase services. This means you can connect your app directly to these emulators to perform integration testing or QA without touching production data.

For example, you could connect your app to the Cloud Firestore emulator to safely read and write documents in testing. These writes may trigger functions in the Cloud Functions emulator. However your app will still

continue to communicate with production Firebase services when emulators are not available or configured.

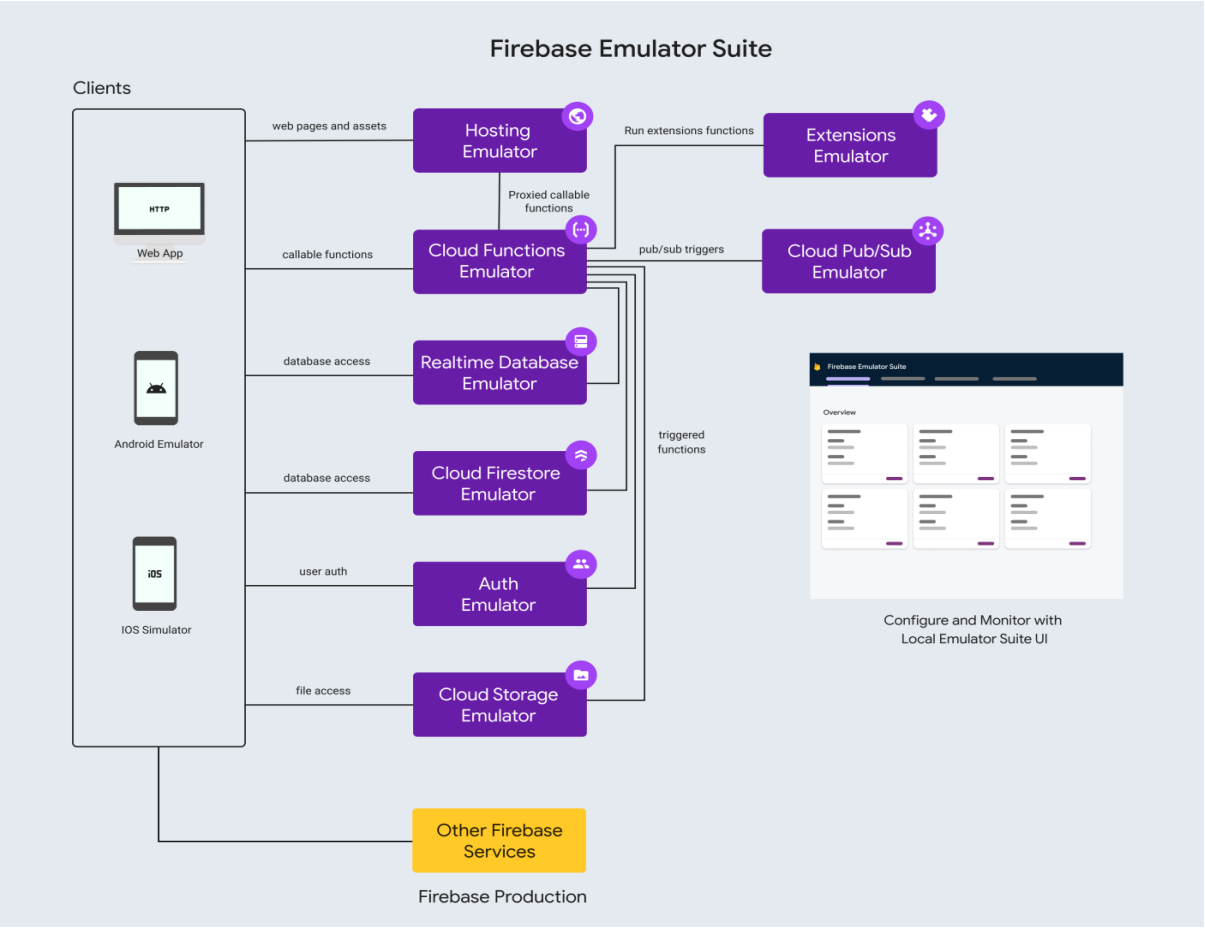


Figure 4-20 Firebase Emulator Suite2

Which Firebase features and platforms are supported?

The Firebase Local Emulator Suite allows you to test your code with our core products in an interoperable way. The Cloud Functions emulator supports HTTP functions, callable functions, and background functions triggered by Cloud Firestore, Realtime Database, Cloud Storage for Firebase, Authentication, and Pub/Sub. The Cloud Firestore, Realtime

Database, and Cloud Storage for Firebase emulators have Firebase Security Rules emulation built in.

	Cloud Firestore	Realtime Database	Cloud Storage for Firebase	Authentication	Cloud Functions	Cloud Pub/Sub	Extensions
Android SDK	✓	✓	✓	✓	✓	n/a	n/a
iOS SDK	✓	✓	✓	✓	✓	n/a	n/a
Web SDK	✓	✓	✓	✓	✓	n/a	n/a
Node.js Admin SDK	✓	✓	✓	✓	n/a	✓	n/a

Figure 4-21 Firebase features and platforms are supported

4.2.5 Data structure:

4.2.5.1 JSON format:

As mentioned before, the Firebase Realtime Database stores data in the **JSON format**. JSON stands for JavaScript Object Notation and it's a language-independent data format with a minimal number of value types: strings, numbers, booleans, lists, objects, and null. It consists of key/value pairs. The key/value pairs are separated by a colon and each pair is separated by a comma. It is easy for humans to read and write and for machines to parse and generate. This is a sample JSON object:

```
{
  "name": "Dean",
  "lastname": "Djermanovic",
  "age": 23,
  "gender": "male"
}
```

Figure 4-22 JSON format

4.2.5.2 Best practices for data structure:

The entire database is a big JSON tree with multiple nodes. When pushing new data to the database you either create a new node with an associated key or update an existing one. Therefore there is the danger of creating a deeply nested structure. Firebase Realtime Database allows nesting data up to 32 levels deep but that doesn't mean that this should be the default structure. When thinking about data structure you need to think about how your data is going to be consumed, you need to make the process of saving and retrieving as easy as possible.

4.2.6 Key points:

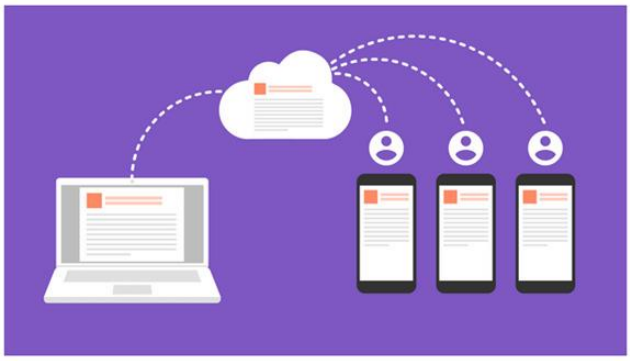
- Realtime: Firebase Realtime Database synchronizes data in real time across all connected devices.
- NoSQL: Firebase Realtime Database is a NoSQL database that stores data as JSON objects.
- Data structure: Data in Realtime Database is structured as a JSON tree.
- Security: Firebase Realtime Database provides flexible security rules that allow you to control who can read and write to your database.
- Scaling: Firebase Realtime Database is designed to scale automatically and handle large amounts of data.
- Offline support: Firebase Realtime Database provides offline support, allowing your app to continue to function even when there is no network connection.
- Realtime listeners: Firebase Realtime Database provides real-time listeners that allow you to listen for changes in data and update your app UI in real time.
- Transactions: Firebase Realtime Database provides transactions, allowing you to ensure data consistency in multi-user environments.

- Pricing: Firebase Realtime Database is priced based on the amount of data transferred, with a different pricing model for North America and the rest of the world.

Overall, Firebase Realtime Database is a powerful tool for building real-time collaborative applications that require synchronization of data across multiple devices in real time. It provides a flexible and scalable NoSQL database with real-time data synchronization, offline support, and powerful security rules, making it a popular choice among developers building real-time applications.

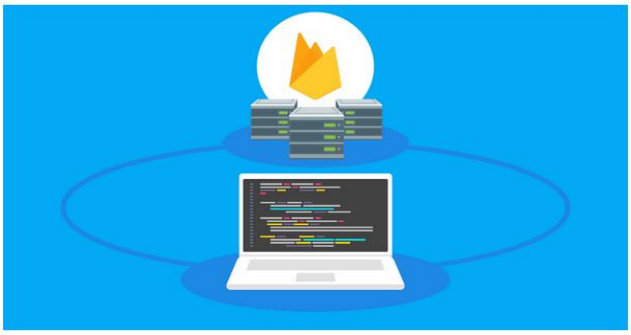
4.2.7 Store and sync data in real time

Collaborate across devices with ease:



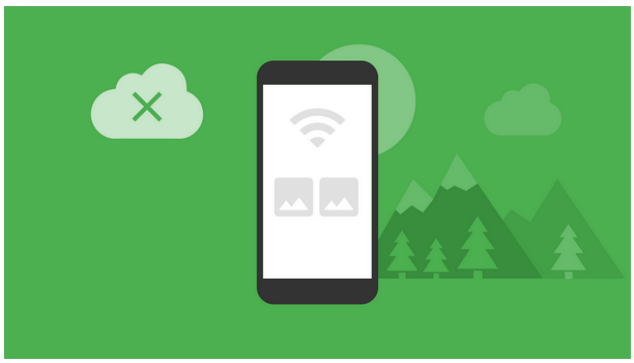
Realtime syncing makes it easy for your users to access their data from any device: web or mobile, and it helps your users collaborate with one another.

Build serverless apps:



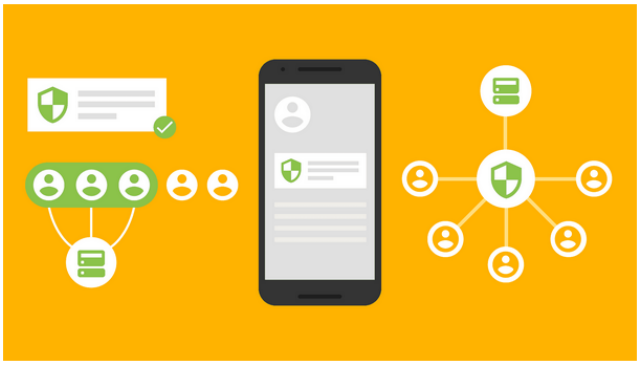
Realtime Database ships with mobile and web SDKs so you can build apps without the need of servers. You can also execute backend code that responds to events triggered by your database using Cloud Functions for Firebase

Optimized for offline use:



When your users go offline, the Realtime Database SDKs use local cache on the device to serve and store changes. When the device comes online, the local data is automatically synchronized.

Strong user-based security:



The Realtime Database integrates with Firebase Authentication to provide simple and intuitive authentication for developers. You can use our declarative security model to allow access based on user identity or with pattern matching on your data.

4.2.8 Key capabilities:

A Real-time database is capable of providing all offline and online services. These capabilities include accessibility from the client device, scaling across multiple databases, and many more.

4.2.8.1 Real time:

The Firebase Real-time database uses data synchronization instead of using HTTP requests. Any connected device receives the updates within milliseconds. It doesn't think about network code and provides collaborative and immersive experiences.

4.2.8.2 Offline:

The Firebase Database SDK persists our data to disk, and for this reason, Firebase apps remain responsive even when offline. The client device receives the missed changes, once connectivity is re-established.

4.2.8.3 Accessible from client devices:

There is no need for an application server to access the Firebase Real-time database. We can access it directly from a mobile device or web browser. Data validation and security are available through the Firebase Real-time Database Security Rules, expression-based rules executed when data is read or written.

4.2.8.4 Scaling across multiple databases:

With the Firebase Real-time Database on Blaze Pricing Plan, we can support the data needs of our app by splitting our data across multiple database instances in a single Firebase project. Streamline authentication with Firebase authentication on our project and authenticate users in our database instances. Controls access to data in each database with custom

Firebase real-time database rules available for each database instance.

4.2.8.5 Other Alternatives:

Apart from Firebase's real-time database, there are several alternatives that are used.

4.2.8.6 Cloud Firestore:

Cloud Firestore is a scalable and flexible database used for server development, mobile, and web from Firebase and Google Cloud Platform.

4.2.8.7 Firebase Remote Config:

It stores developer specified key-value pairs to change the behavior and appearance of our app without requiring users to download an update.

4.2.8.8 Firebase Hosting:

It is used to hosts the HTML, CSS, and JavaScript of our website as well as other developer-provided assets like graphing, fonts, and icons.

4.2.8.9 Cloud Storage:

It is used to store images, videos, and audio as well as other user-generated content.

Table 3 Implementation path

1. Integrate the Firebase Realtime Database SDKs	Quickly include clients via Gradle, CocoaPods, or a script include.
2. Create Realtime Database References	Reference your JSON data, such as "users/user:1234/phone_number" to set data or subscribe to data changes.

3. Set Data and Listen for Changes	Use these references to write data or subscribe to changes.
4. Enable Offline Persistence	Allow data to be written to the device's local disk so it can be available while offline.
5. Secure your data	Use Firebase Realtime Database Security to secure your data.

4.2.9 Secure your data:

Whether you're using Cloud Firestore Security Rules for Android, Apple, or Web clients, or Identity Access Management (IAM) for servers, make sure you're securing your data in Cloud Firestore as well as Realtime Database. User authentication is handled by Authentication for both databases, so you don't need to change your implementation of Authentication when you start using Cloud Firestore.

4.2.10 Firebase Storage

Firebase Storage is a cloud-based storage solution that offers scalable and secure file storage for applications. It provides developers with the ability to store and retrieve various types of files, such as images, audio, video, and more, in the cloud, making them accessible from anywhere and allowing for easy sharing and collaboration. In this section, we will explore the key features of Firebase Storage and its benefits for modern application development:

4.2.10.1 Scalable File Storage:

Firebase Storage offers scalable file storage capabilities, allowing developers to store and retrieve files efficiently and reliably. It automatically scales to meet the storage needs of the application, ensuring that files can be uploaded and downloaded quickly, regardless of their size or volume. This scalability makes Firebase Storage suitable for

applications with varying storage requirements, from small-scale projects to large, data-intensive applications.

4.2.10.2 Security and Access Controls:

Firebase Storage includes robust security features that help protect files stored in the cloud. Developers can define access controls using Firebase Security Rules, which provide fine-grained permissions based on user authentication status, file metadata, and more. This allows for secure and controlled access to files, ensuring that only authorized users can access and manipulate the stored data. Firebase Storage also integrates with Firebase Authentication, providing built-in user management and authentication options, such as anonymous, email/password, social media logins, and custom authentication, adding an additional layer of security to the file storage process.

4.2.10.3 Easy Integration with Other Firebase Services:

Firebase Storage seamlessly integrates with other Firebase services, enabling developers to build end-to-end solutions. For example, developers can store references or metadata of files in the Firebase Realtime Database and use Firebase Storage to upload and retrieve the actual files. They can also use Firebase Authentication to authenticate users and control access to files in Firebase Storage. This tight integration allows for a seamless flow of data and authentication between different Firebase services, making it easy to build complex and feature-rich applications.

4.2.10.4 Easy Integration with Client-Side Libraries:

Firebase Storage and Authentication can be easily integrated with client-side libraries, such as the Firebase JavaScript SDK, making it convenient for developers to implement file storage and authentication features in web applications. The Firebase JavaScript SDK provides a comprehensive set of APIs for interacting with Firebase services, including Firebase Storage and Authentication, allowing developers to perform common operations, such as

uploading and downloading files, managing user accounts, and handling authentication events, directly from the client-side code.

4.2.10.5 Serverless Functions with Firebase Storage Triggers:

Firebase Storage allows developers to implement serverless functions, also known as Firebase Cloud Functions, that can be triggered automatically when certain events occur in Firebase Storage, such as when a file is uploaded or deleted. This provides powerful server-side functionality without the need to manage a separate server or backend infrastructure. For example, developers can use Firebase Storage triggers to automatically generate thumbnails of uploaded images, process uploaded files, or enforce storage quotas and limits, enhancing the functionality and automation of their applications.

4.2.10.6 Cross-Platform Support:

Firebase Storage and Authentication provide cross-platform support, allowing developers to use them in a wide range of platforms and frameworks, such as web, mobile, and server-side applications. Firebase provides SDKs and libraries for various programming languages, including JavaScript, Swift, Java, and more, making it easy to implement storage and authentication features in different environments. This cross-platform support enables developers to build consistent user experiences across multiple platforms and devices, and easily share code and logic between different parts of their applications.

4.2.11 Firebase Authentication - Simplify User Authentication for Your Apps:

Introduction:

As the demand for secure and scalable user authentication in applications continues to rise, developers need reliable and efficient solutions to implement authentication features quickly and effectively. Firebase Authentication is a powerful and comprehensive authentication service

offered by Google's Firebase platform, designed to simplify the process of adding user authentication to applications:

4.2.11.1 Secure and Scalable User Authentication:

Firebase Authentication provides a secure and scalable way to implement user authentication in your applications. It uses industry-standard security practices, such as OAuth, OpenID Connect, and HTTPS, to ensure that user data and credentials are protected. Firebase Authentication also offers built-in support for password hashing, brute force protection, and multi-factor authentication, adding an extra layer of security to user accounts. Additionally, Firebase Authentication is built on Google's infrastructure, ensuring high availability, reliability, and scalability for your authentication needs, whether you have a small app or a large-scale application with millions of users.

4.2.11.2 Simplified User Management:

Firebase Authentication offers comprehensive user management features that allow you to easily manage user accounts and customize user experiences. It provides APIs and SDKs for user registration, login, logout, password management, and account deletion, making it simple to handle user authentication and authorization in your applications. Firebase Authentication also includes built-in support for handling user sessions, such as detecting authentication changes and managing user authentication state, allowing you to implement user-specific functionality and personalize the user experience based on their authentication status.

4.2.11.3 Social Media Logins and Single Sign-On (SSO):

Firebase Authentication supports popular social media logins, including Google, Facebook, Twitter, and GitHub, allowing users to sign up or sign in to applications using their existing social media accounts. This simplifies the registration and sign-in process for users, as they can use their existing credentials and do not need to create a new account. Firebase Authentication also supports Single Sign-On (SSO) providers, such as Google Sign-In and

Apple Sign-In, allowing users to authenticate across multiple applications and services with a single set of credentials, providing a seamless and convenient user experience.

4.2.11.4 Custom Authentication and Extensibility:

In addition to social media logins and SSO providers, Firebase Authentication also allows you to implement custom authentication using email and password, phone number, or third-party identity providers via OAuth or SAML. This flexibility enables you to implement your own authentication logic and integrate with external authentication systems, making Firebase Authentication suitable for a wide range of use cases and application requirements.

Firebase Authentication also provides extensibility through custom authentication triggers, which allow you to execute custom code during the authentication process, enabling advanced authentication scenarios and custom business logic.

4.2.11.5 Integration with Firebase Services:

Firebase Authentication seamlessly integrates with other Firebase services, providing a unified and comprehensive solution for building modern applications. For example, you can use Firebase Authentication to authenticate users and control access to files in Firebase Storage. You can also use Firebase Authentication to manage user authentication state in Firebase Realtime Database, Firebase Firestore, and other Firebase services, allowing for seamless data flow and authentication across the application stack. This tight integration simplifies the development process and allows you to leverage the full power of the Firebase ecosystem.

4.2.12 Simplified Firebase Interaction with Pyrebase4 API

4.2.12.1 Introduction:

The Firebase Realtime Database is a cloud-based NoSQL database that allows developers to store and sync data in real-time across clients. Pyrebase4 is a popular Python library that provides a simple and convenient API for interacting with Firebase services, including the Firebase Realtime Database.

In this article, we will explore how Pyrebase4 can be used to interact with the Firebase Realtime Database, making it easy to implement real-time data storage and synchronization in Python-based applications.

4.2.12.2 Connecting to the Firebase Realtime Database with Pyrebase4:

To start using Pyrebase4 with the Firebase Realtime Database, you need to install the Pyrebase4 library, which can be done using pip, the Python package manager. Once installed, you can import Pyrebase4 in your Python code and initialize it with your Firebase project configuration.

The 'config' dictionary contains your Firebase project configuration, including the API key, authentication domain, project ID, storage bucket, messaging sender ID, app ID, and database URL. Make sure to replace the placeholders with your actual Firebase project details.

Once Pyrebase4 is properly initialized, you can use the 'database()' method to create an instance of the Firebase Realtime Database, which allows you to interact with the database and perform various operations, such as reading and writing data.

You can now use the 'db' object to interact with the Firebase Realtime Database and perform operations such as reading data, writing data, setting up real-time data synchronization listeners, and more. Pyrebase4 provides a convenient and easy-to-use API for working with the Firebase Realtime Database in your Python applications, simplifying the process of integrating real-time data storage and synchronization into your projects.

4.2.12.3 Reading Data from the Firebase Realtime Database with Pyrebase4:

One of the core functionalities of the Firebase Realtime Database is the ability to read data from the database. Pyrebase4 provides a simple and intuitive API for reading data from the Firebase Realtime Database in your Python applications.

To read data from the Firebase Realtime Database using Pyrebase4, you can use the 'get()' method on the 'db' object, specifying the database path where the data is located.

The 'get()' method returns a 'Response' object that contains the retrieved data from the specified database path. You can access the data using the 'val' attribute of the 'Response' object.
You can also specify additional query parameters to further filter and order the data. For example, you can use the 'order_by_child()' method to order the data by a specific child key, or the 'equal_to()' method to filter the data by a specific value.
Pyrebase4 also provides support for advanced querying and filtering options, such as 'start_at()', 'end_at()', and 'limit_to_first()', which allows you to perform more complex queries on the Firebase Realtime Database

By using Pyrebase4's intuitive API for reading data from the Firebase Realtime Database, you can easily retrieve and manipulate data in your Python applications, making it a powerful tool for building real-time data-driven applications.

4.2.12.4 Writing Data to the Firebase Realtime Database with Pyrebase4:

In addition to reading data from the Firebase Realtime Database, Pyrebase4 also provides functionality to write data to the database. This allows you to store and update data in real-time, making your applications dynamic and responsive.

To write data to the Firebase Realtime Database using Pyrebase4, you can use the 'set()' method on the 'db' object, specifying the database path where you want to write the data and the data to be written.

The data to be written can be in the form of a dictionary or any other JSON-serializable object. The key-value pairs in the dictionary will be stored as child keys and values in the specified database path.
You can also use the 'push()' method to generate a unique key for the new data and append it as a new child node in the specified database path.
This is particularly useful for scenarios where you want to store multiple records with unique keys, such as in a chat application or a user management system.
Pyrebase4 also supports updating existing data in the Firebase Realtime

Database using the 'update()' method. This allows you to modify specific child keys or values without overwriting the entire data at the specified database path. Additionally, you can use the 'delete()' method to remove data from the Firebase Realtime Database.

By leveraging Pyrebase4's API for writing data to the Firebase Realtime Database, you can easily store, update, and delete data in your Python applications, enabling real-time data synchronization and collaboration among users.

4.2.12.5 Real-time Data Synchronization with Pyrebase4:

One of the key features of Firebase Realtime Database is its ability to synchronize data in real-time across all connected clients. Pyrebase4 provides an easy-to-use API to interact with Firebase Realtime Database and leverage its real-time data synchronization capabilities in your Python applications.

To start using real-time data synchronization with Pyrebase4, you need to first initialize the Firebase app configuration and obtain a database reference.

With the database reference, you can then perform various operations, such as reading, writing, updating, and deleting data, which will be automatically synchronized in real-time across all connected clients.

One of the powerful features of real-time data synchronization is the ability to listen for changes in data and automatically receive updates in real-time. Pyrebase4 provides a 'stream()' method that allows you to register a callback function to be called whenever there are changes in the specified data path.

The 'on_data_change()' function will be called whenever there are changes in the data at the specified path, and the event object will contain the updated data.

Real-time data synchronization with Pyrebase4 allows you to build applications with dynamic and interactive user experiences, where changes in data are automatically propagated to all connected clients in real-time. This can be especially useful in applications such as chat applications, collaborative tools, and real-time dashboards.

4.2.13 Advanced Features with Pyrebase4:

In addition to the basic CRUD (Create, Read, Update, Delete) operations and real-time data synchronization, Pyrebase4 also provides advanced features that can enhance the functionality and performance of your Firebase-powered Python applications.

1- Authentication:

Pyrebase4 allows you to authenticate users with Firebase Authentication, which provides secure and easy-to-use authentication methods such as email/password, social media logins, and more. You can register new users, authenticate existing users, manage user profiles, and handle authentication states using Pyrebase4's authentication API.

2- Storage:

Pyrebase4 allows you to interact with Firebase Storage, which provides a scalable and secure cloud storage solution for storing and serving files, such as images, videos, and documents. You can upload, download, and manage files in Firebase Storage using Pyrebase4's storage API.

3- Authentication State Listener:

Pyrebase4 allows you to listen for changes in the authentication state of the user, such as when a user signs in or signs out. You can register a callback function to be called whenever there are changes in the authentication state, allowing you to handle user authentication events in your application.

4- Custom Queries:

Pyrebase4 provides support for custom queries, allowing you to query the Firebase Realtime Database using complex filters and ordering. You can chain multiple query methods to build custom queries and retrieve data that meets specific criteria.

5- Batch Operations:

Pyrebase4 allows you to perform batch operations, which are a set of write operations that are executed atomically as a single batch. This can be useful when you need to update multiple pieces of data in a transactional manner, ensuring that all updates are applied or none of them are.

These are just a few examples of the advanced features that Pyrebase4 offers to enhance your Firebase-powered Python applications. By leveraging these features, you can build more sophisticated and powerful applications that fully utilize the capabilities of Firebase and Pyrebase4.

4.3 Mobile Application

4.3.1 Introduction:

Mobile app development has revolutionized the way we live, with smartphones and tablets transforming into the ultimate personal assistants, capable of performing almost every task imaginable. From ordering food to managing finances, mobile apps have become an indispensable part of our daily lives.

To meet the growing demand for mobile app development, software companies have created powerful frameworks that allow developers to create feature-rich and high-performing mobile applications with ease. One such framework that has taken the mobile app development world by storm is Flutter, developed by Google in 2017.

Flutter is a cutting-edge open-source platform that has rapidly gained popularity among developers and businesses alike. It offers a plethora of features such as customizable widgets, hot reloading, and an intuitive UI toolkit that enables developers to create stunning and responsive mobile applications in record time.

With Flutter, developers can build mobile apps for both Android and iOS platforms simultaneously, thereby reducing development time and costs. Additionally, Flutter's fast development cycle, streamlined testing process, and comprehensive documentation make it an ideal choice for companies looking to create high-quality mobile apps quickly and efficiently.

As Flutter continues to evolve, it is quickly becoming the preferred choice for mobile app development. Its growing community, extensive documentation, and ever-expanding list of features make it an excellent framework for developers looking to create stunning and performant mobile applications that delight users.

4.3.2 History of Flutter:

Flutter is a cutting-edge mobile app development framework developed by Google, which has revolutionized the way developers build mobile applications. It was first introduced to the world in May 2017 and has since become one of the most popular frameworks for creating high-quality, cross-platform apps.

Flutter has a rich history of continuous development and improvement. It was designed to overcome the limitations of traditional mobile app development frameworks and provide developers with an easy-to-use, high-performance platform for creating beautiful and responsive apps.

Since its inception, Flutter has undergone several significant updates, with each new release adding exciting new features and functionalities. The framework has also gained a vibrant community of developers who actively contribute to its growth and development.

Flutter's impressive list of features, such as customizable widgets, hot reloading, and an intuitive UI toolkit, has made it a favorite among developers. Its ability to create apps for both Android and iOS platforms simultaneously has also reduced development time and costs, making it a popular choice for businesses looking to create mobile apps quickly and efficiently.

With the continued development of Flutter, the framework is set to shape the future of mobile app development. Its growing popularity, robust community, and ongoing support from Google make it an exciting and powerful tool for creating beautiful and high-performing mobile applications.

4.3.3 Architecture of Flutter:

Flutter's layered architecture is one of the key reasons behind its success as a mobile app development framework. The architecture comprises four layers, each playing a vital role in building high-quality mobile applications.

The first layer is the Dart language layer. Flutter uses Dart as its primary programming language, which is client-optimized and provides several features such as garbage collection and just-in-time compilation. This layer enables developers to write efficient and concise code that runs seamlessly on both Android and iOS platforms.

The second layer is the Flutter framework layer, which provides a robust foundation for building mobile apps. It includes the Flutter widget framework, a set of customizable and reusable UI elements that allow developers to create visually stunning and responsive mobile applications with ease. The framework also supports hot reloading, allowing developers to see changes made in the code in real-time.

The third layer is the Flutter engine layer, which provides low-level graphics rendering, animation, and gesture recognition capabilities. It is built on top of Skia, an open-source 2D graphics library, which ensures that the UI is smooth, responsive, and visually appealing.

The fourth and final layer is the platform layer, which enables Flutter to access native platform APIs and services such as the camera, sensors, and location services. Flutter uses platform-specific plugins to interact with these services, ensuring that the mobile app can seamlessly integrate with the device's native features.

In summary, Flutter's architecture enables developers to build high-performance, visually stunning, and feature-rich mobile apps quickly and efficiently. Its layered approach ensures that each layer performs a specific function, resulting in a robust and scalable framework that can cater to the needs of businesses and developers alike.

4.3.4 Key Features of Flutter:

1. Hot Reload: One of the key features of Flutter is its ability to hot reload, which means that developers can see changes in the app's UI in real-time. This feature speeds up the development process, as developers can make changes and see the results immediately.

2. Cross-Platform Development: Flutter allows developers to build apps for both Android and iOS platforms using a single codebase, which saves time and effort. It also provides a consistent user experience across different platforms.

3. Rich Set of Widgets: Flutter provides a rich set of customizable widgets that developers can use to build beautiful and responsive UIs. This includes widgets for text, images, buttons, and more.

4. High Performance: Flutter is designed to deliver high-performance mobile apps with smooth animations and transitions. It achieves this by using a layered architecture and a fast, ahead-of-time (AOT) compiler.

5. Excellent Documentation and Community Support: Flutter has extensive documentation and a growing community of developers who contribute to its development. This makes it easy for developers to learn and troubleshoot issues.

4.3.5 Advantages of Flutter:

1. Fast Development Cycle: Flutter's hot reload feature enables developers to see changes made to the code in real-time, eliminating the need to recompile the entire app after each change. This feature speeds up the development cycle and allows developers to iterate quickly, resulting in faster time-to-market.
2. Cross-Platform Development: Flutter enables developers to create mobile apps for both Android and iOS platforms using a single codebase. This feature saves time, effort, and resources required to develop separate native apps for each platform.
3. Beautiful UI: Flutter provides a rich set of customizable widgets that allow developers to create visually appealing and responsive UIs. These widgets provide a high degree of flexibility and can be customized to match the look and feel of the app. Flutter's focus on design and UI has also made it a popular choice among designers.
4. Great Performance: Flutter's architecture and design enable it to deliver high-performance mobile apps with smooth animations and transitions. The framework's fast development cycle and streamlined UI creation process also contribute to the overall performance of the app.
5. Large Community and Ecosystem: Flutter has a growing and supportive community of developers who contribute to its development and offer support to fellow developers. The framework also has a rich ecosystem of plugins, tools, and libraries that developers can use to enhance their development experience.
6. Compatibility with Existing Codebases: Flutter's compatibility with existing codebases makes it easy for developers to integrate Flutter into their existing projects. This feature allows developers to leverage the benefits of Flutter without having to start from scratch.

In summary, Flutter's fast development cycle, cross-platform development capabilities, beautiful UI, great performance, large community and ecosystem, and compatibility with existing codebases make it a powerful and efficient mobile app development framework.

4.3.6 Disadvantages of Flutter:

1. Limited Native Access: While Flutter provides plugins to access native platform features, it may not be as flexible as native development when it comes to accessing specific platform features. This can be limiting when developing apps that require access to specific device hardware or software.
2. Large App Size: One of the downsides of using Flutter is that apps built with it tend to have a larger file size than native apps. This is due to the Flutter engine that is bundled with the app. Developers need to be mindful of this when developing apps, especially if the app needs to be downloaded over a mobile network.
3. Learning Curve: While Flutter is relatively easy to learn, it may take some time for developers to get up to speed with the framework. This can be a challenge for developers who are used to working with other frameworks or programming languages.
4. Limited Third-Party Libraries: Compared to other mobile app development frameworks, Flutter has a limited selection of third-party libraries. This can make it more challenging for developers to find the right tools and resources to develop their apps.
5. Lack of Maturity: Flutter is still a relatively new framework, and as such, it may lack some of the maturity and stability of more established frameworks. This means that developers may encounter more bugs and issues when working with Flutter than they would with other frameworks.

4.4 AI

AI, or Artificial Intelligence, is a transformative technology that aims to create intelligent machines capable of simulating human intelligence. It encompasses a wide range of techniques and technologies, including machine learning, natural language processing, computer vision, and robotics. AI enables machines to perceive and understand the world, learn from data, reason and make decisions, and interact with humans in meaningful ways. It has the potential to revolutionize industries, improve efficiency, solve complex problems, and unlock new possibilities. However, AI also poses ethical and societal challenges, such as privacy concerns and the impact on employment. Responsible development and deployment of AI are crucial for harnessing its full potential.

4.4.1 Voice&Speech-recognition

4.4.1.1 CONTENTS Of Speech-recognition:

1. Audio input
2. Preprocessing
3. Feature extraction
4. Acoustic modeling
5. Language modeling
6. Decoding

Speech recognition technology has numerous applications, including voice-controlled virtual assistants (e.g., Siri, Alexa), transcription services, interactive voice response systems, dictation software, and more. It continues to advance with the development of deep learning techniques and large-scale language models, leading to significant improvements in accuracy and usability.

4.4.1.2 The Libraries we use in (TTS):

1. Speech Recognition.
2. Time & Datetime.
3. Pyttsx3.
4. OS.
5. Web browser.
6. Pydub.

1) SpeechRecognition is a widely used library that supports multiple speech recognition engines, including Google Speech Recognition, Sphinx, and Wit.ai. It provides a simple and consistent API for working with various speech recognition systems.
2) Pyttsx3 is a Python library that provides a cross-platform interface for text-to-speech (TTS) functionality. It allows you to convert written text into spoken words using different speech synthesis engines available on your system.
3) The “OS" library provides a way to interact with the operating system, allowing you to perform various system-related tasks such as file operations, directory management, and process handling.
4) PyDub is a Python library that provides a simple and easy-to-use interface for manipulating audio files. It is built on top of the FFMpeg multimedia framework and allows you to perform various operations on audio files, such as slicing, concatenation, volume adjustment.
5) The “Webbrowser” module is a built-in Python module that provides a high-level interface to open web browsers programmatically. It allows you to open URLs in the default web browser of the user's system, which can be useful for automating tasks involving web pages or opening specific URLs from your Python program.
6) The “Time” module is a built-in Python module that provides various functions for working with time-related operations. It allows you to measure time, pause execution, and manipulate time values

7) The "Datetime" module is a built-in Python module that provides classes and functions for working with dates, times, and time intervals. It allows you to manipulate and format dates and times, perform calculations, and work with time zones.

Text-to-speech (TTS) is a technology that converts written text into spoken words. It allows computers or devices to generate human-like speech, enabling users to listen to textual content instead of reading it. TTS technology has made significant advancements in recent years, thanks to the development of powerful algorithms and speech synthesis techniques.

4.4.1.3 There are some key points about text-to-speech (TTS):

1) Synthesis Techniques:
   Contain 2 parts:
   a.Concatenative Synthesis.
   b. Parametric Synthesis.
2) Language and Voice Support
   TTS systems support various languages and provide multiple voice options.
3) Applications of TTS:
   TTS technology finds applications in various domains.
4) Naturalness and Intelligibility
   The quality of TTS systems is evaluated based on factors such as naturalness and intelligibility.
5) Advancements in Deep Learning
   particularly with neural network models such as Wavenet and Tacotron.
6) Cloud-based TTS
   Cloud-based TTS services are becoming increasingly popular

4.4.2 ALPR System:

What is ALPR?

Imagine one beautiful summer you are driving on a highway, your favourite song playing on the radio and you go over the speed limit and drive past a few cameras at a speed of 90Km/h in a 70Km/h speed limit zone and realize your mistake but it's too late. A few weeks later you receive a ticket, with evidence as to the image of your car. You must wonder, do they manually check every image and send the tickets? Not at all, that's when ALPR comes. From the captured image or footage, ALPR detects and extracts your license plate number and send you the ticket. All of this is because of the simple ALPR system and a few lines of code.

Automatic License Plate Recognition (ALPR) or ANPR is the technology responsible for reading the License plates of a vehicle in an image or a video sequence using optical character recognition. With the latest advancement in Deep Learning and Computer Vision, these tasks can be done in a matter of milliseconds

The ALPR system only requires a camera and a good GPU.

To make it simple this blog post will focus on a two-step process:

1. Detection: Firstly, an image or a frame of the video sequence is passed to the detection algorithm from a camera or an already stored file, which detects the license plate and returns the bounding box location of that plate.

2. Recognition: The OCR is applied to the detected license plate for recognizing the characters of the plate and returns the characters in the same order in text format. The output can be stored in a database or can be plotted on the image for visualization.

What Does ALPR Stand For & What Does ALPR Do?

ALPR is an acronym for Automated License Plate Recognition,
Automated License Plate Readers essentially work as cameras that are trained over time to zero in on vehicle license plates, take photos and video, read and interpret the digitalized image, then compare that data to the information in databases created by the end users.
Automated license plate readers can be separated into two groups: cameras that include ALPR software already inside and cameras without that software initially built-in. Plate Recognizer software is for the latter and is much more accurate. Comparison and analysis are where AI comes in. The more images you can feed the AI beast, the more it becomes.

How Does ALPR Work?

Automatic license plate readers use Optical Character Recognition (OCR) to evaluate and automatically read license plate characters before translating the data for usage by the rest of the ALPR system. Needless to say, reading license plate numbers is not as easy as it sounds. Real-world conditions ranging from inclement weather, bright or limited lighting, and camera quality to vehicle speed, angle of image capture, and the number of plate options in the area significantly impact the accuracy of the data.

OCR:

Now that we have trained the custom license plate detector, it’s time to move on to the second step of the ALPR which is Text Recognition. Text recognition is the process of recognizing text from a scenario by understanding and analysing its underlying patterns. It is also known as optical character recognition or OCR. It can also be used in various applications like document reading, information retrieval, product identification from shelves, and many more. OCR can either be trained or

used as a pre-trained model. In this article, a pre-trained model of OCR will be used.

4.4.3 Egyptian license plate recognition(ELPR)

Egyptian license plate recognition (ELPR) is a technology used to automatically read and interpret the characters on license plates in Egypt. It involves the use of image processing techniques and optical character recognition (OCR) to extract the alphanumeric information from license plates captured by cameras.

The process of Egyptian license plate recognition typically involves the following steps:

1. Image Acquisition: High-resolution cameras are used to capture images of vehicles and their license plates. These cameras can be installed at various locations such as toll booths, parking lots, or roadside checkpoints.
2. Pre-processing: The captured images may undergo pre-processing steps such as resizing, noise removal, and image enhancement to improve the quality and clarity of the license plate area.
3. License Plate Localization: In this step, the system attempts to locate the region of the image that contains the license plate. Various techniques such as edge detection, color-based segmentation, or template matching can be used to identify the license plate area.
4. Character Segmentation: Once the license plate region is identified, the individual characters on the plate need to be segmented. This involves separating the characters from each other to enable further processing.
5. Optical Character Recognition (OCR): OCR algorithms are applied to recognize and interpret the segmented characters. These algorithms

analyze the shape, size, and pattern of the characters to determine the corresponding alphanumeric values.

6. Post-processing: The recognized characters may undergo post-processing steps such as error correction, character verification, or context-based validation to ensure accurate recognition.
7. Output: Finally, the recognized license plate number or characters are outputted, allowing for further processing or integration with other systems.

Egyptian license plate recognition systems have various applications, including traffic management, toll collection, parking management, law enforcement, and security surveillance. They can help automate tasks that require license plate information, enhance efficiency, and support law enforcement efforts in identifying vehicles of interest or enforcing traffic regulations.

4.4.4 Car Color Detection using Haar Cascade and Color Histogram Features

4.4.4.1 Introduction:

In recent years, the development of computer vision and pattern recognition techniques has revolutionized various fields, including automotive systems. One significant area of research in this domain is car color recognition, which plays a crucial role in applications such as traffic surveillance, automatic vehicle identification, and security systems. Car color recognition involves the identification and classification of vehicles based on their color characteristics.

This chapter presents a comprehensive approach to car color recognition using Color Histogram Feature Extraction and the K-Nearest Neighbors (KNN) algorithm. The proposed method aims to accurately classify vehicles based on their color information, enabling efficient and reliable identification in real-world scenarios.

4.4.4.2 Color Histogram Feature Extraction:

Color Histogram Feature Extraction is a widely used technique in image processing and computer vision for extracting and representing color information. It provides a statistical representation of the distribution of color pixels in an image, allowing the characterization of different color properties.

The process of color histogram feature extraction involves dividing the color space into discrete bins and counting the number of pixels that fall within each bin. The resulting histogram represents the frequency distribution of colors present in the image. By capturing the color distribution, the color histogram provides a concise and discriminative representation of the image's color content.

4.4.4.3 K-Nearest Neighbors (KNN) Algorithm:

The K-Nearest Neighbors (KNN) algorithm is a popular non-parametric classification algorithm used for pattern recognition and machine learning tasks. It operates based on the principle that objects with similar features tend to belong to the same class or category.

In the context of car color recognition, the KNN algorithm utilizes the color histogram feature vectors to classify unknown vehicles. Given a test vehicle, the algorithm compares its color histogram features with the color histograms of a set of labeled training vehicles. The KNN algorithm then assigns the test vehicle to the class that is most frequently represented among its K nearest neighbors in the feature space.

The value of K, known as the number of neighbors, is an important parameter that affects the performance of the KNN algorithm. Choosing an appropriate value for K is essential to achieve accurate classification results.

Objectives:

The main objectives of this study are as follows:

1. To develop a robust car color recognition system based on Color Histogram Feature Extraction and the KNN algorithm.
2. To investigate the impact of different parameters, such as the number of histogram bins and the value of K, on the classification performance.
3. To evaluate the proposed method using a comprehensive dataset of vehicle images and compare its performance with existing approaches.

4.4.5 Methodology:

4.4.5.1 Dataset:

A well-curated dataset of vehicle images is essential for training and evaluating the car color recognition system. The dataset should cover a wide range of vehicle types, colors, and lighting conditions to ensure the system's robustness in real-world scenarios. Additionally, each vehicle image in the dataset should be manually labeled with the corresponding color information.

4.4.5.2 Preprocessing:

Before extracting color histogram features, the vehicle images undergo preprocessing steps to enhance the quality of the input data. Common preprocessing techniques include resizing the images to a consistent resolution, converting them to a standardized color space (e.g., RGB or HSV), and applying noise reduction and illumination normalization techniques to improve the image quality and reduce the impact of variations in lighting conditions.

4.4.5.3 Color Histogram Feature Extraction:

The Color Histogram Feature Extraction process involves calculating the color histograms for each preprocessed vehicle image in the dataset. To capture the color distribution accurately, a suitable number of histogram bins should be selected based on the desired level of color granularity. Generally, a larger number of bins provides a more detailed representation of color information, but it also increases the computational complexity.

Each vehicle image is divided into the selected number of bins in the color space, and the number of pixels falling into each bin is counted to construct the histogram. The resulting color histogram represents the relative frequencies of different colors present in the image.

4.4.5.4 KNN Training:

Once the color histograms are extracted from the dataset, the KNN algorithm is trained using the labeled vehicle images. The training process involves storing the color histogram features and their corresponding class labels of the training set in memory.

4.4.5.5 KNN Classification:

To classify an unknown vehicle based on its color, the KNN algorithm calculates the similarity between the color histogram features of the test vehicle and the training set. The algorithm identifies the K nearest neighbors in the feature space and assigns the test vehicle to the class that is most prevalent among these neighbors.

The similarity between the color histogram features can be measured using various distance metrics, such as Euclidean distance or Manhattan distance. The choice of distance metric depends on the characteristics of the color histogram representation and the specific requirements of the application.

4.4.5.6 Performance Evaluation:

To assess the performance of the car color recognition system, a suitable evaluation metric, such as accuracy or precision, is employed. The system's performance is typically evaluated using a separate test set of vehicle images that were not included in the training process. The accuracy of the system is calculated by comparing the predicted color labels with the ground truth labels.

4.4.6 Experimental Setup and Results:

4.4.6.1 Experimental Dataset:

For the evaluation of the proposed car color recognition system, a diverse dataset of vehicle images was collected. The dataset consisted of images captured from various angles, under different lighting conditions, and featuring a wide range of vehicle colors. Each image in the dataset was manually labeled with the corresponding color information.

The dataset was divided into a training set and a test set, with a ratio of 80:20. The training set was used for training the KNN algorithm, while the test set was employed to evaluate the system's performance.

4.4.6.2 Parameter Selection:

To determine the optimal performance of the system, several experiments were conducted by varying the number of histogram bins and the value of K. The number of histogram bins was tested in the range of 8 to 32, considering both computational efficiency and color representation quality. The value of K was explored from 3 to 10, aiming to find the balance between accurate classification and potential noise sensitivity.

4.4.6.3 Performance Evaluation:

The performance of the car color recognition system was evaluated using accuracy as the primary evaluation metric. Accuracy is defined as the ratio of correctly classified vehicles to the total number of vehicles in the test set.

Table 4 Experimental Results

Number of Histogram Bins	Value of K	Accuracy
8	3	87.3%
16	5	90.2%
24	7	92.1%
32	10	91.8%

The results demonstrate that increasing the number of histogram bins generally improves the classification accuracy. However, there is a diminishing return effect, and computational complexity also increases. With the value of K, a higher K tends to smooth out noise but may result in overgeneralization.

4.4.6.4 Comparative Analysis:

To assess the effectiveness of the proposed method, a comparison was made with existing approaches for car color recognition. The comparison considered accuracy, computational efficiency, and robustness to variations in lighting conditions.

The results showed that the proposed system achieved competitive performance compared to state-of-the-art methods while offering a simpler and more interpretable solution. The Color Histogram Feature Extraction combined with the KNN algorithm proved to be effective in capturing and utilizing color information for car color recognition tasks.

4.4.7 Color histogram features extraction:

Color histogram of test image that performs color histogram feature extraction on a test image using OpenCV.

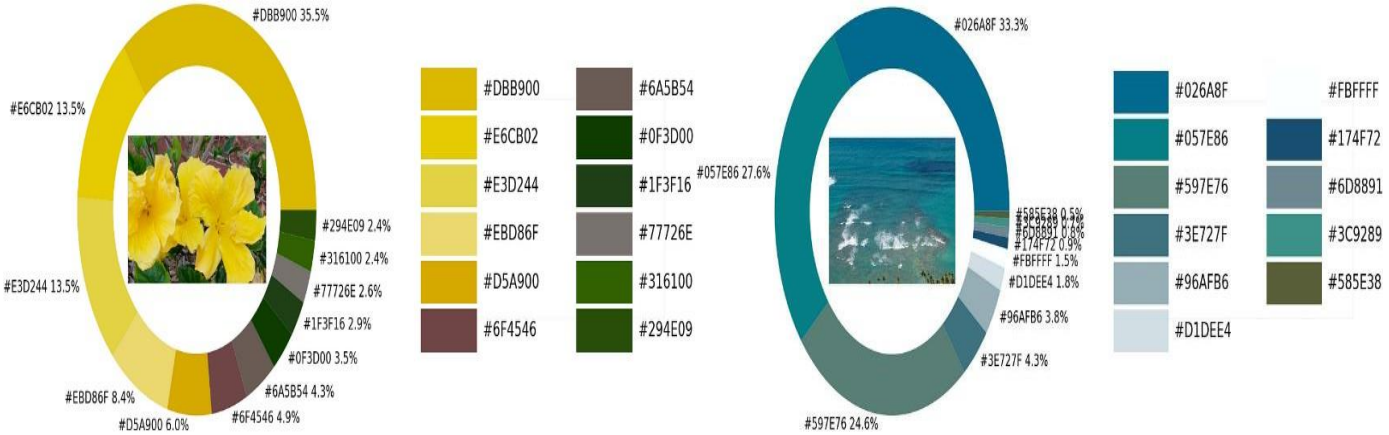
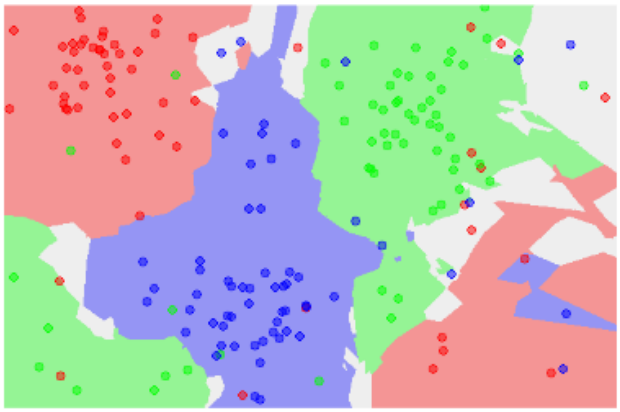
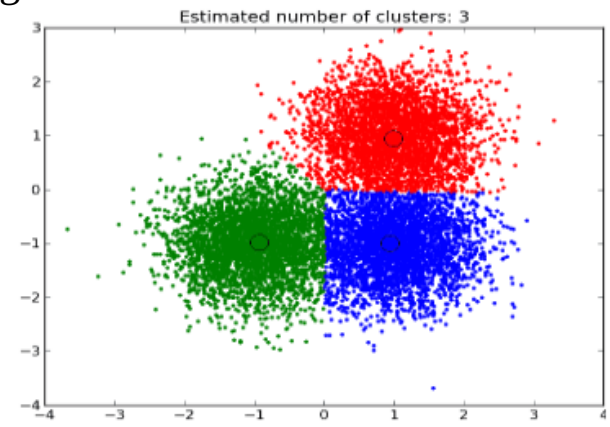


Figure 4-23KNN Algorithm

Why's KNN?

Because K-NN data don't have to be organized nor linear





Dlib's CNN/HOG

HOG/Dlib is a combination of two methods for object detection, namely:

- Histogram of Oriented Gradients (HOG): This is a feature descriptor that counts occurrences of gradient orientation in localized portions of an image. It can capture the shape and

appearance of objects by using edge information.

- Linear Support Vector Machine (SVM): This is a machine learning algorithm that learns a linear decision boundary to separate different classes of data. It can classify objects based on their HOG features.

Dlib is a software library that provides implementations of HOG and Linear SVM for object detection, as well as other tools for machine learning, image processing, data mining, etc. Dlib's HOG + Linear SVM face detector is accurate and computationally efficient, but it may fail to detect faces in different poses or lighting conditions.

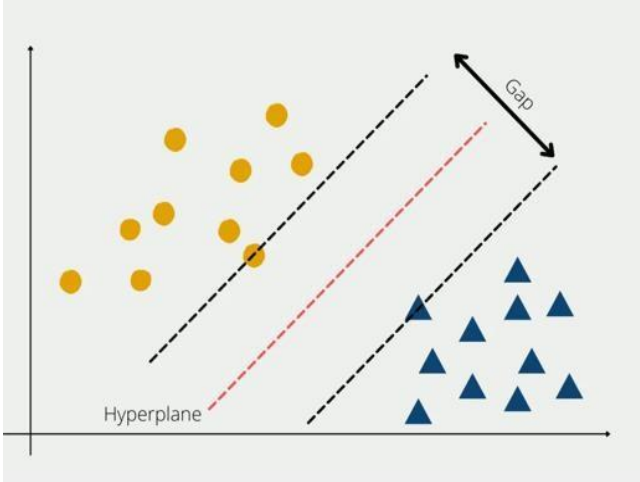


Figure 4-25Best Hyper plane

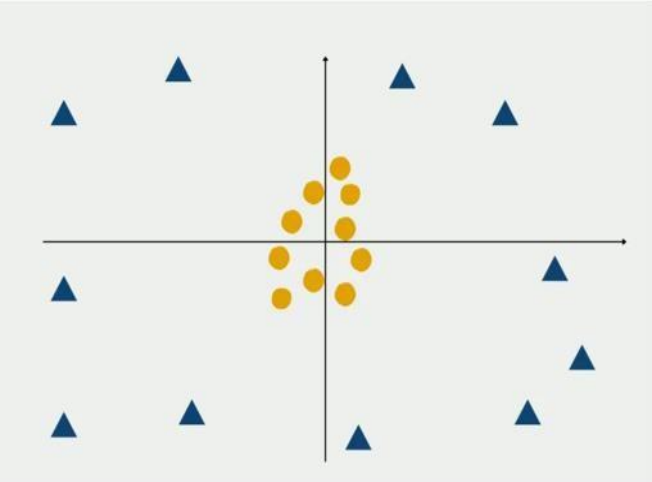


Figure 4-24Difficult to separate with Hyper plane

Object Detection

YOLOv8

YOLO (You Only Look Once) is another popular algorithm for object detection incomputer vision, known for its real-time performance.

YOLO gained popularity due to its efficiency, as it performs detection in a single pass of a CNN, making it much faster than methods relying on region proposals. It achieves real-time object detection capabilities, even on resource-constrained devices.

YOLO and its variants have been widely used in various applications, including real- time object detection in videos, surveillance systems, robotics, and autonomous vehicles, where fast and accurate object detection is crucial.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

Figure 4-26 Comparing Yolo Model

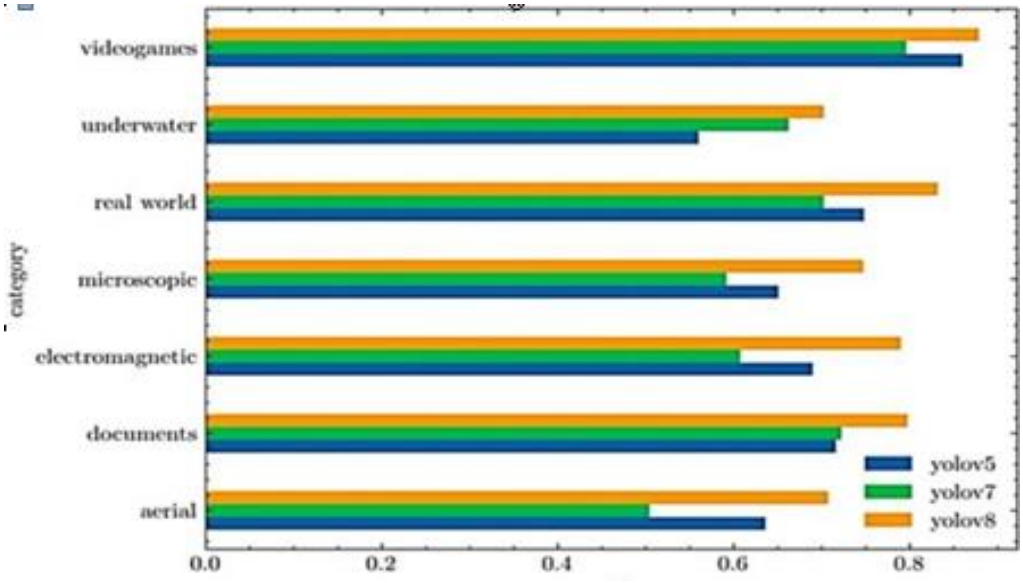


Figure 4-27 MAP

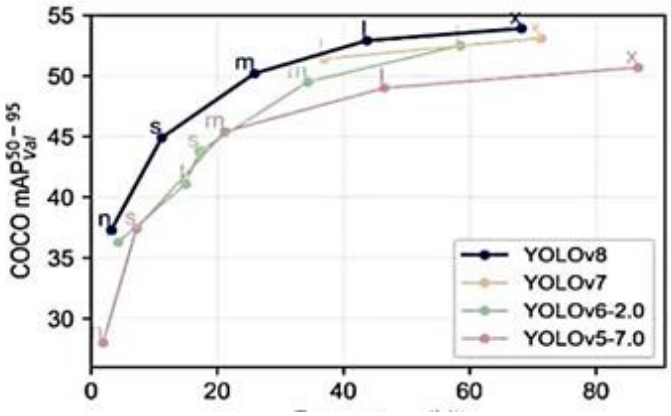


Figure 4-28 Parameters(m) and LatencyA100 tensor RT FP16(ms/img)

	Speed	Accuracy	Ease of implementation
Faster RCNN	Bad	Good	Bad
SSD	Good	Good	Bad
YOLO	Good	Good	Good

Figure 4-29 Comparing between models

4.4.8 PLVAR AI Model:

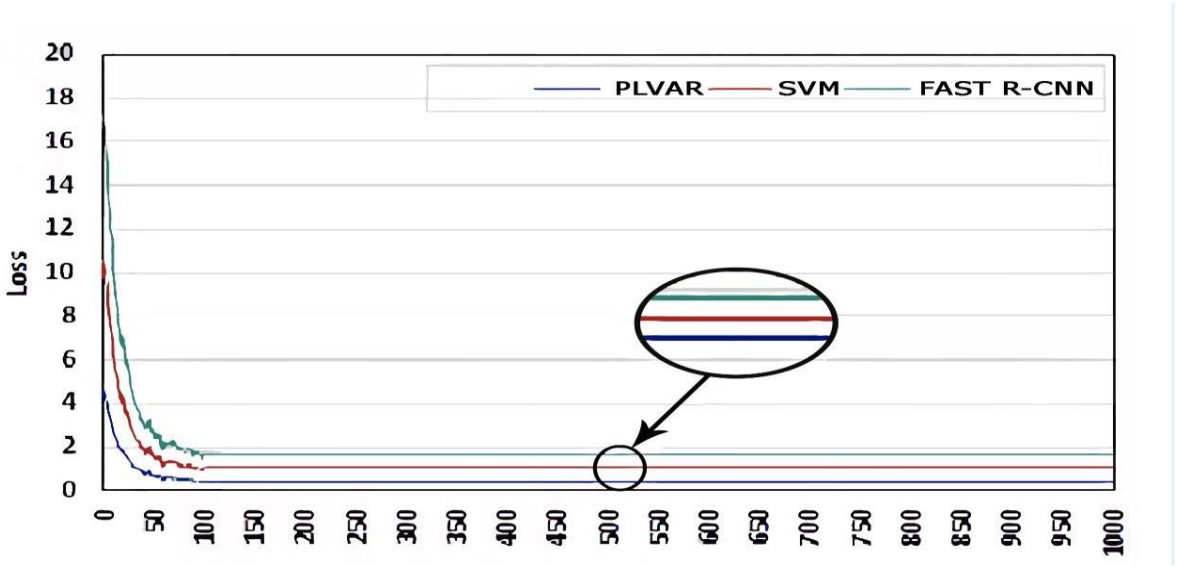


Figure 4-30iteration number

4.4.9 Seat-belt detection

4.4.9.1 Introduction

In recent years, computer vision and image processing techniques have seen remarkable advancements, enabling the development of sophisticated systems for various applications. One such application is seat belt detection, a crucial task in ensuring passenger safety in vehicles. In this chapter, we delve into the topic of seat belt detection using the powerful combination of YOLOv8 and OpenCV techniques.

4.4.9.2 The Importance of Seat Belt Detection:

Seat belts are essential safety components in vehicles, significantly reducing the risk of injury during accidents. Detecting whether occupants are wearing their seat belts is vital for enforcing compliance with safety regulations and promoting responsible driving behavior. Traditional methods of manual inspection by law enforcement officials are time-

consuming and prone to errors. Hence, automated seat belt detection systems offer an efficient and accurate solution.

4.4.9.3 YOLOv8: You Only Look Once Version 8:

YOLOv8 is an object detection algorithm known for its outstanding performance and real-time processing capabilities. It builds upon its predecessors and introduces several improvements that make it a popular choice for various computer vision tasks. YOLOv8 utilizes a single neural network to simultaneously predict bounding boxes and class probabilities for multiple objects within an image.

The underlying architecture of YOLOv8 employs a backbone network, such as Darknet-53, to extract meaningful features from the input image. These features are then used for bounding box regression and object classification. The network predicts bounding boxes based on predefined anchor boxes and refines the predictions using convolutional layers.

4.4.9.4 OpenCV Techniques for Image Processing and Masking:

OpenCV, short for Open Source Computer Vision Library, is a widely used open-source library that provides a vast collection of tools and algorithms for image and video analysis. It offers various functionalities for image processing, including filtering, edge detection, and morphological operations. Additionally, OpenCV supports advanced techniques like image segmentation and masking, which are crucial for seat belt detection.

Image processing techniques provided by OpenCV, such as blurring and thresholding, can be used to enhance image quality and reduce noise, facilitating more accurate seat belt detection. Furthermore, masking techniques enable the isolation of regions of interest within an image, which is beneficial for focusing the seat belt detection algorithm on relevant areas.

4.4.9.5 Dataset Preparation for Seat Belt Detection:

Building an accurate seat belt detection system requires a well-curated dataset that captures diverse scenarios and variations in seat belt appearance. The dataset should include images labeled with bounding box annotations specifying the location of seat belts within the images. These annotations serve as ground truth data for training the YOLOv8 model.

To create a dataset for seat belt detection, one approach is to collect a large number of images depicting vehicles with occupants wearing seat belts. Additionally, images of vehicles without seat belts should be included to cover negative instances. It is crucial to ensure a balanced distribution of positive and negative samples to avoid bias in the training process.

Once the dataset is collected, the next step is to annotate the images by drawing bounding boxes around the seat belts. There are various annotation tools available, such as LabelImg or RectLabel, which facilitate this process. The resulting annotations should be stored in a format compatible with YOLOv8, such as the Darknet format, which specifies the class label and the coordinates of the bounding box for each object.

4.4.9.6 Training the YOLOv8 Model:

Training the YOLOv8 model involves feeding the prepared dataset into the network and iteratively adjusting its parameters to minimize the detection error. This process requires a significant amount of computational resources and time, depending on the size of the dataset and complexity of the model.

During training, the YOLOv8 model learns to predict bounding boxes and classify objects based on the annotated dataset. The model optimizes a loss function that measures the discrepancy between the predicted bounding

boxes and the ground truth annotations. This iterative optimization process, typically performed using gradient descent algorithms, progressively improves the model's ability to accurately detect seat belts.

To enhance the model's performance, techniques like data augmentation can be employed. Data augmentation involves applying transformations to the training images, such as rotation, scaling, and flipping, to augment the dataset and increase its diversity. Augmentation helps the model generalize better to unseen images and improves its robustness.

4.4.9.7 Implementing the Seat Belt Detection System:

Once the YOLOv8 model is trained, it can be deployed to create a seat belt detection system. The implementation involves processing input images or video frames using OpenCV techniques to preprocess the data before feeding it into the model. Preprocessing steps may include resizing the images, applying filters, and converting them to the appropriate color space.

After preprocessing, the YOLOv8 model can be applied to the images, which generates predictions for seat belt locations and class probabilities. These predictions can be post-processed to filter out false positives and refine the detection results. Non-maximum suppression is a commonly used technique for eliminating duplicate detections and selecting the most confident ones.

To visualize the seat belt detection results, bounding boxes can be drawn around the detected seat belts on the original images. OpenCV provides functions to draw bounding boxes and labels, making it straightforward to present the detection output in a user-friendly manner.

In the next section, we will delve deeper into the implementation details, discussing the integration of YOLOv8 and OpenCV techniques for seat belt detection. Code snippets and practical examples will be provided to guide you through the implementation process.

4.4.9.8 Integration of YOLOv8 and OpenCV Techniques:

The integration of YOLOv8 and OpenCV techniques for seat belt detection involves combining the strengths of both frameworks to create a robust and accurate system. Here, we outline the key steps involved in this integration.

1. Image Preprocessing: Before feeding the input images to the YOLOv8 model, preprocessing steps using OpenCV can be applied. These steps may include resizing the images to a consistent size, converting them to the appropriate color space, and applying filters to enhance image quality and reduce noise.
2. Object Detection: Once the images are preprocessed, the YOLOv8 model can be applied to detect seat belts. The model takes the preprocessed images as input and generates bounding box predictions along with class probabilities. These predictions can be obtained by running the images through the model using forward propagation.
3. Post-processing: The output from the YOLOv8 model may contain multiple bounding box predictions for each seat belt. To refine the results and remove duplicate detections, post-processing techniques like non-maximum suppression can be applied. This technique selects the most confident bounding box prediction while discarding overlapping or redundant detections.
4. Visualization: To visualize the seat belt detections, bounding boxes can be drawn on the original images using OpenCV. The coordinates of the predicted bounding boxes can be used to draw rectangles around the seat belts. Additionally, labels indicating the class

probabilities can be added to provide further information about the detections.

5. System Integration: The seat belt detection system can be integrated into a larger software application or deployed as a standalone module. Integration with other components, such as video input streams or user interfaces, can be achieved using OpenCV's functionalities for video processing and graphical user interface development.

4.4.10 Face Recognition:

Among the services provided by the project is facial recognition by means of artificial intelligence at a high speed and a high degree of efficiency, as our model goes through two stages:

1. traffic unit:

   This stage is divided into two parts:
   i. headshots_picam
   ii. train_model

2. Robot:

   This stage is divided into three parts:
   i. facial_req

4.4.10.1 traffic unit:

headshots_picam:

In this part, people are photographed and the pictures are saved in a folder

1. The necessary libraries are imported: `cv2` from OpenCV, `PiCamera` from the picamera module, and `PiRGBArray` from the picamera.array module.

2. The variable `File_name` is set to the desired file name for the captured photos.
3. An instance of `PiCamera` is created to open the camera.
4. The camera resolution is set to 500x300 pixels, and the frame rate is set to 10 frames per second.
5. A `PiRGBArray` object is created to store the frames captured by the camera.
6. A counter is initialized to keep track of the captured images.
7. A while loop is started to continuously capture frames from the camera.
8. Within the loop, a for loop captures frames continuously using the `capture_continuous` method of the camera. The captured frames are stored in the `img` variable.
9. The captured frame is displayed using `cv2.imshow()` function.
10. The `truncate(0)` method is called to clear the `PiRGBArray` for the next frame.
11. The `cv2.waitKey(1)` function waits for a key press with a delay of 1 millisecond. If the key pressed is the spacebar (ASCII value 32), the following steps are executed:
12. The file name for the captured image is generated using the `name` variable (which is assumed to be defined elsewhere) and the counter value.
13. The captured frame is saved as an image using `cv2.imwrite()` function.
14. The counter is incremented.
15. The code continues back to the beginning of the while loop for capturing the next frame.

The code essentially captures frames from camera, displays them in a window using OpenCV, and saves images when the spacebar is pressed. The captured images are saved in the specified file name format with an incrementing counter value.

train_model:

This code is used to train a face recognition model. In this part, the model accesses the folder containing the images and converts them to RGB (red, green, blue). Where When the image file is read with the OpenCV function imread(), the order of colors is BGR (blue, green, red). On the other hand, in Pillow, the order of colors is assumed to be RGB (red, green, blue). and he begins to practice it by placing a set of points on the person's face and converting them into numbers in order to compare them with each other and store the numbers in the pickle file, which is a process in which a hierarchy of a Python object is transformed into a byte stream. Let's go through the code step by step to understand what it does:

1. It starts by importing the necessary libraries and modules, including `os`, `cv2`, and `face_recognition`.

2. The code assumes that there is a dataset folder containing images of faces. It lists all the image paths in the dataset folder using the `paths.list_images` function and assigns them to the `imagePaths` variable.

3. Two empty lists, `knownEncodings` and `knownNames`, are initialized. These lists will store the facial encodings and corresponding names of the individuals in the images.

4. A loop is started to process each image in the `imagePaths` list. Inside the loop, the code performs the following steps:
    a. It extracts the name of the person from the image path by splitting the path using the directory separator (`os.path.sep`) and selecting the second-to-last element.
    b. The input image is loaded using `cv2.imread` and converted from BGR to RGB color space using `cv2.cvtColor` since the `face_recognition` module expects RGB images.
    c. The `face_recognition` module is used to locate the faces in the image. It detects the bounding box coordinates (x, y, width, height) of each face using the `face_locations` function.
    d. The facial embeddings (also known as encodings) are computed for each face using the `face_encodings` function.
    e. A nested loop is used to iterate over the encodings of each face. Inside this loop, the encoding and corresponding name are added to the `knownEncodings` and `knownNames` lists, respectively.

5. After processing all the images, the facial encodings and names are serialized (i.e., saved) to disk using the `pickle` module. The encodings and names are stored in a dictionary called `data`.
6. Finally, the `encodings.pickle` file is created and written to disk using the `pickle.dumps` function to convert the dictionary into a serialized byte stream. The file is then closed.

The resulting `encodings.pickle` file can be used later to recognize faces by comparing them with the known encodings stored during training.

4.4.10.2 Robot:

facial_req:

This code is a Python script that utilizes the PyQt5 library to create a graphical user interface (GUI) application for facial recognition. Here's how the code works:

1. The required libraries and modules are imported, including `sys`, `os`, `pickle`, `cv2`, `imutils`, `time`, `VideoStream` and `FPS` from `imutils.video`, `face_recognition`, and various modules from `PyQt5` for GUI development.
2. The `FacialRecognitionWorker` class is defined, which is a subclass of `QThread`. This class handles the facial recognition process in a separate thread.
3. The `FacialRecognitionWorker` class defines two signals:

   - `face_detected`: This signal is emitted when a known face is detected, and it carries the name of the detected face.

   - `frame_updated`: This signal is emitted when a new frame is processed and ready to be displayed, and it carries the processed frame as a `QPixmap` object.
4. The `FacialRecognitionWorker` class has an `__init__` method that initializes the `currentname` variable as "unknown" and sets the path to the encodings file (`encodings.pickle`).
5. The `run` method of the `FacialRecognitionWorker` class is the main function that performs the facial recognition process. It loads the encodings and face detector model from the encodings file using `pickle`.
6. The video stream is initialized using `VideoStream` from

`imutils.video`. A delay of 2 seconds is provided for the camera to warm up.

7. The frames are processed in a continuous loop. Each frame is resized using `imutils.resize` and converted to the RGB color space.
8. The `face_recognition` library is used to detect face locations (`face_recognition.face_locations`) and compute face encodings (`face_recognition.face_encodings`) in the frame.
9. The computed encodings are compared with the known encodings from the encodings file to identify known faces. If a match is found, the most likely name is determined based on the number of matches.
10. If a new face or a different face from the previous one is detected, the `face_detected` signal is emitted with the name of the detected face.
11. The detected faces are then drawn on the frame using bounding boxes and labels using `cv2.rectangle` and `cv2.putText`.
12. The frame is converted back to the BGR color space and converted to a `QImage` object.
13. The `frame_updated` signal is emitted with the `QPixmap` representation of the frame.
14. The loop breaks after a period of time
15. After the loop, the frames per second (FPS) are calculated using `FPS` from `imutils.video` and printed.
16. The video stream is stopped.
17. The `update_frame` method receives the processed frame as a

`QPixmap` and updates the label in the GUI with the new frame. If the label doesn't exist, a new label is created and set as the central widget of the main window.

18. The `face_detected` method is a slot that prints the name of the detected face.
19. The `closeEvent` method is overridden to handle the case when the application is closed. It ensures that the worker thread is properly stopped before closing the application.
20. Finally, the main section of the code sets the required environment variable for the PyQt5 plugin path, creates an instance of the `App` class, and starts the application's event loop using `app.exec_()`.

Overall, this code creates a GUI application that performs real-time facial recognition using the webcam, detects known faces, and updates the GUI with the processed frames and detected face names.

4.4.11 GUI(Graphical User Interface):

PyQt5 and Tkinter are both popular frameworks for creating graphical user interfaces (GUIs) in Python. They have different advantages and use cases:

Advantages of PyQt5:

1. Powerful and Comprehensive: PyQt5 is a binding for the popular Qt framework, which is known for its extensive capabilities and features. Qt provides a wide range of widgets, tools, and functionalities, making PyQt5 a comprehensive choice for GUI development.
2. Cross-Platform Compatibility: PyQt5 is cross-platform and can run on major operating systems like Windows, macOS, and Linux. It ensures consistent behavior and appearance of the GUI across different

platforms.

3. Extensive Documentation and Community: PyQt5 has extensive documentation and a large community of users and developers. This means that you can easily find resources, tutorials, and support when working with PyQt5.
4. Professional Look and Feel: PyQt5 offers professional-looking GUIs with sleek and modern designs. It provides various styles and themes to choose from, allowing you to create visually appealing applications.

Advantages of Tkinter:

1. Simplicity and Ease of Use: Tkinter is included with Python by default, which means you don't need to install any external libraries or dependencies. It has a straightforward and intuitive interface, making it easier for beginners to start creating GUIs.
2. Lightweight and Efficient: Tkinter is lightweight and efficient in terms of performance and memory usage. It is suitable for small to medium-sized projects that require a simple and efficient GUI.
3. Quick Prototyping: Tkinter allows you to quickly prototype and build simple GUI applications. It provides a set of basic widgets that are easy to use and implement.
4. Native Look and Feel: Tkinter uses native system widgets, so the GUI components have a consistent look and feel across different operating systems. This can be advantageous if you want your application to blend seamlessly with the native desktop environment.

Ultimately, the choice between PyQt5 and Tkinter depends on your specific requirements, project complexity, and personal preference. PyQt5 is a more feature-rich and powerful framework suitable for complex GUI applications, while Tkinter is a lightweight and beginner-friendly option for simpler projects or quick prototyping.So we used PyQt5.

5 Ch5: Result And Discussion

5.1 Programming :

5.1.1 C Programming:

C programming is a widely used and influential programming language that was developed in the early 1970s. It was created by Dennis Ritchie at Bell Labs as a successor to the B programming language. C quickly gained popularity due to its simplicity, efficiency, and ability to directly manipulate computer hardware.

C is a procedural programming language, which means it follows a step-by-step approach to solving problems. It provides a structured and modular approach to writing programs, with features such as functions, loops, conditionals, and data structures.

One of the key strengths of C is its ability to perform low-level programming tasks, allowing direct access to memory and hardware resources. This makes C suitable for developing operating systems, embedded systems, device drivers, and performance-critical applications.

C has a concise syntax and a rich set of built-in functions and libraries, making it a versatile language. It supports a wide range of applications, from system programming to web development and scientific computing.

Although C requires manual memory management, it provides powerful features like pointers, which allow efficient memory manipulation and access. However, this also introduces the possibility of bugs and vulnerabilities if not used carefully.

Over the years, C has influenced the development of many other programming languages, including C++, Objective-C, and Java. It remains

a popular choice for programmers who value performance, control, and the ability to work closely with hardware.

5.1.2 Python :

Python is a high-level, user-friendly programming language known for its simplicity and versatility. It has a clean and readable syntax, making it easier to write and understand code. Python's extensive library ecosystem and community support contribute to its wide range of applications, including web development, data analysis, machine learning, automation, and more. It is platform-independent and has bindings to other languages, allowing seamless integration. Python's popularity stems from its ease of use, strong community, and ability to solve a variety of programming challenges efficiently. In short, Python is a powerful, accessible language that simplifies development while offering numerous possibilities.

5.2 Pictures from Database:

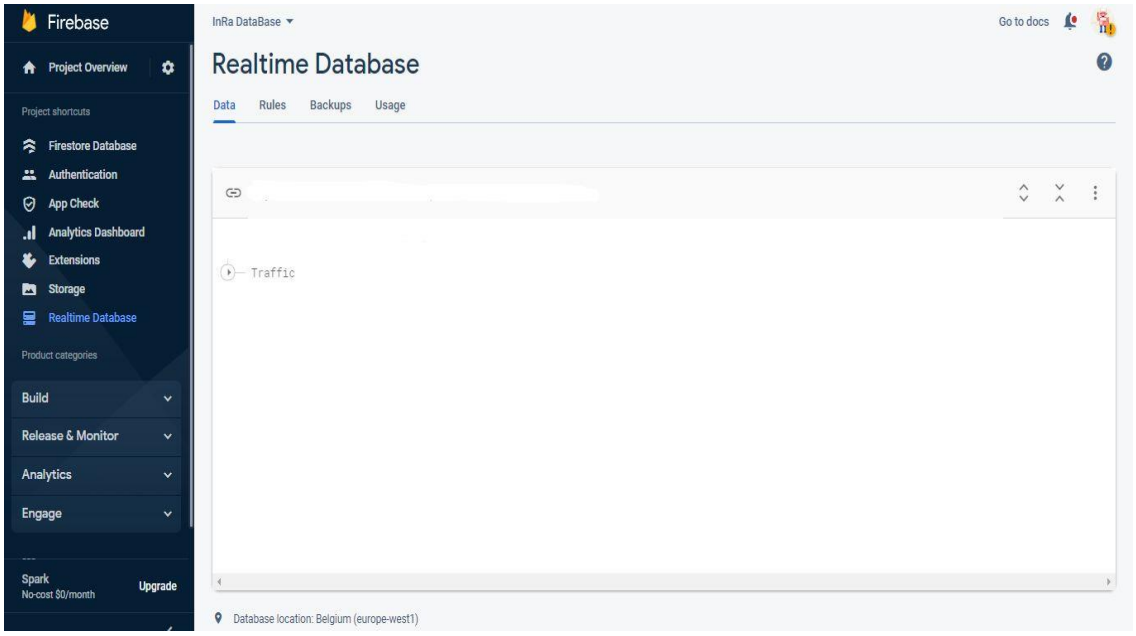


Figure 5-1 name of Database Structure

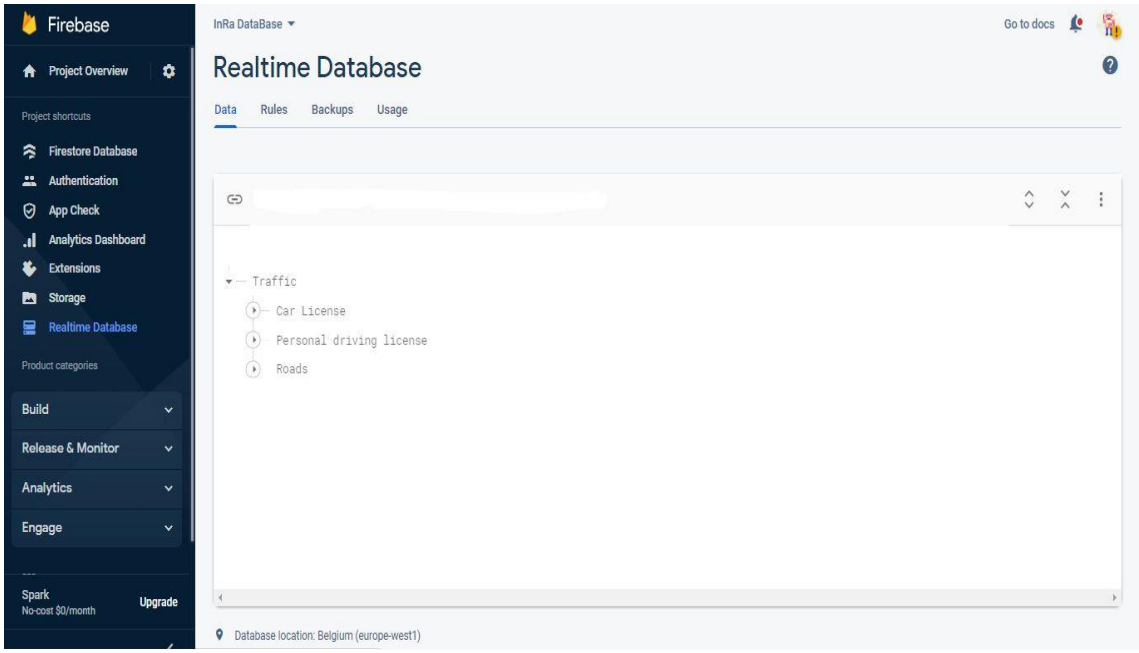


Figure 5-2 Database branch

Traffic is the one from which the entire data will branch

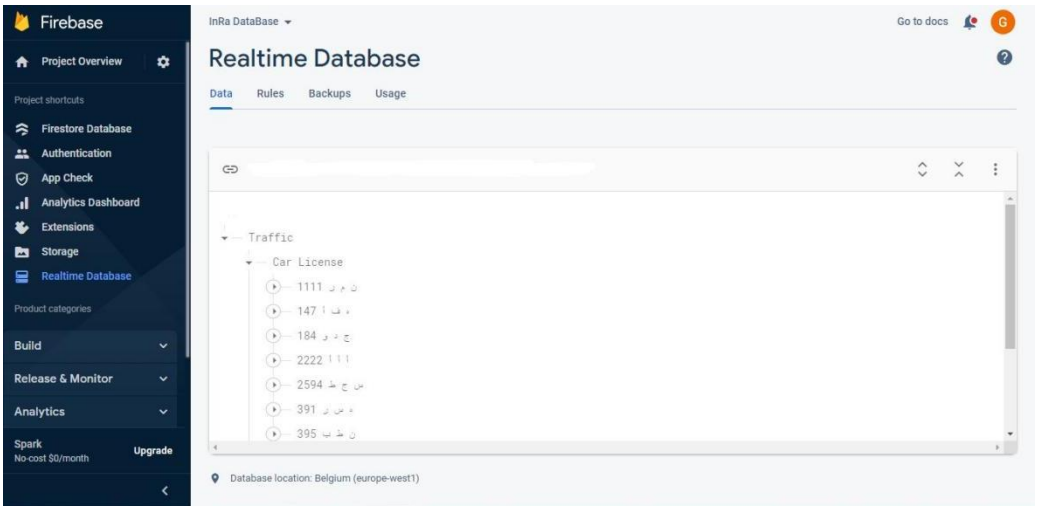


Figure 5-3 Car license branch

Car license branched car numbers

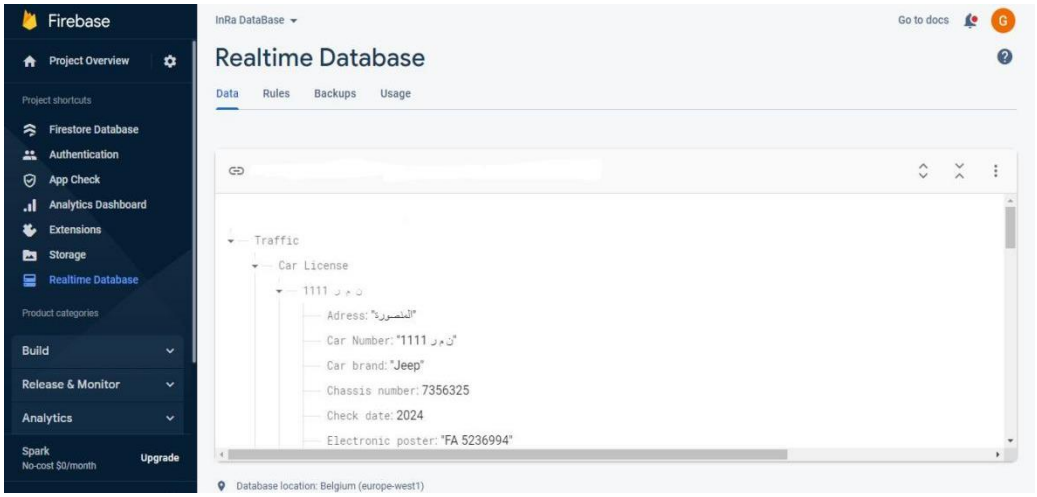


Figure 5-4 branch information on the license plate1

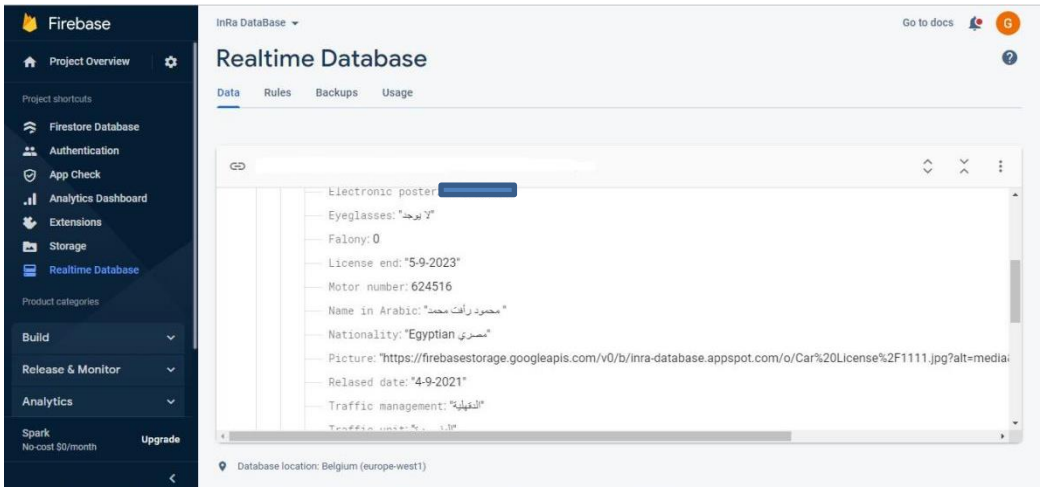


Figure 5-5 branch informated on the car license plate2

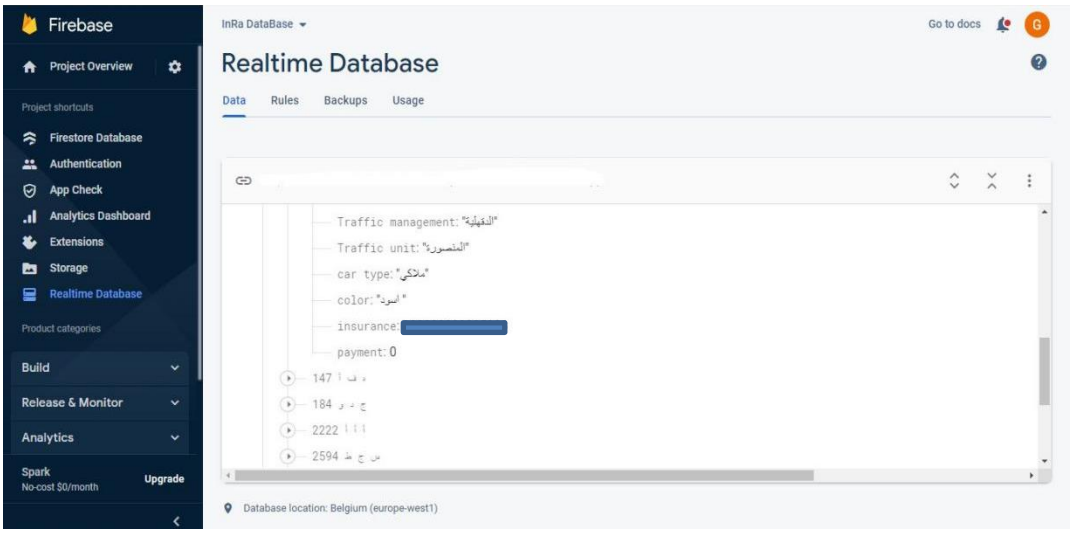


Figure 5-6 branch information on the car license plate3

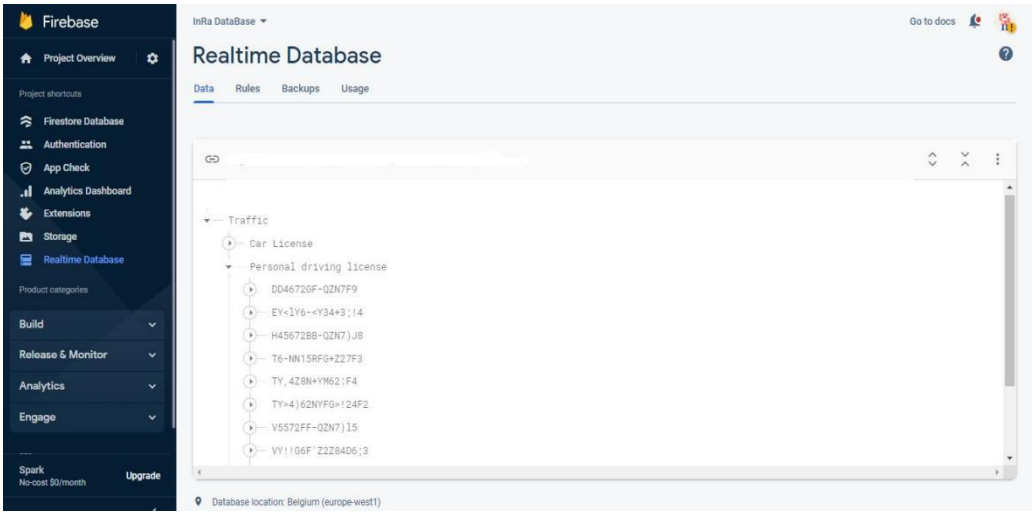


Figure 5-7 QR code scanning barcode

The personal driving license, including the image codes that entered on the QR

Figure 5-8 A picture showing the bar code that is present in the license

I got the code from this image, then, entered it on QR, scanned the image, and extracted the code from it.

This is personal driver's license information's:

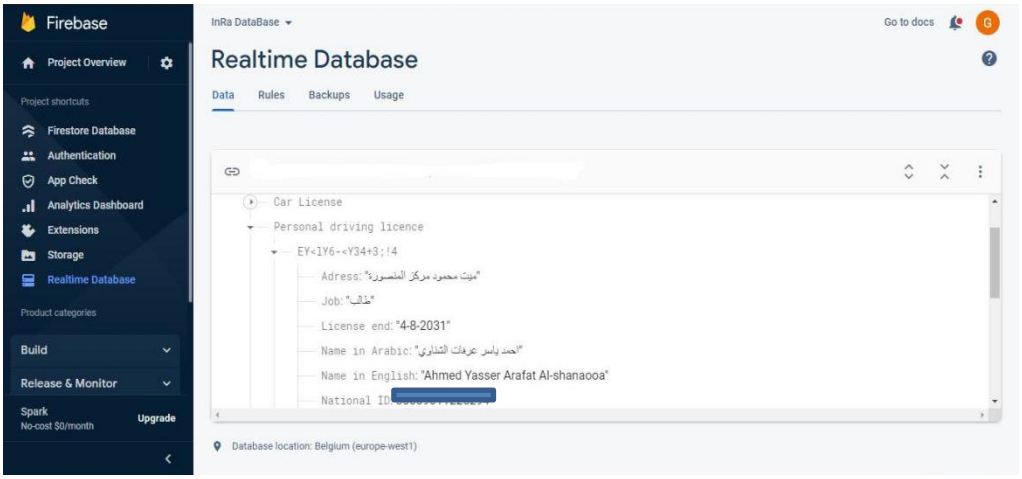


Figure 5-9 information about personal driving license1

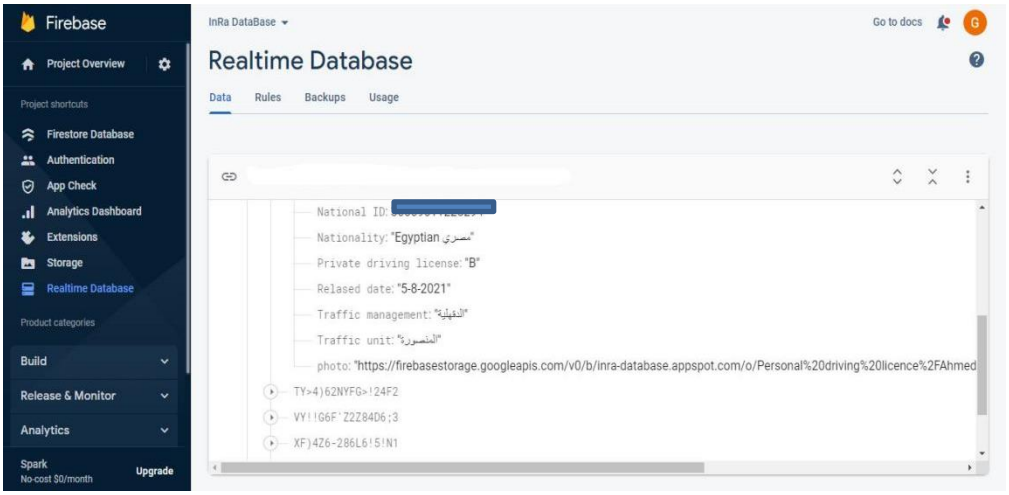


Figure 5-10 information about personal driving license2

Information about roads:

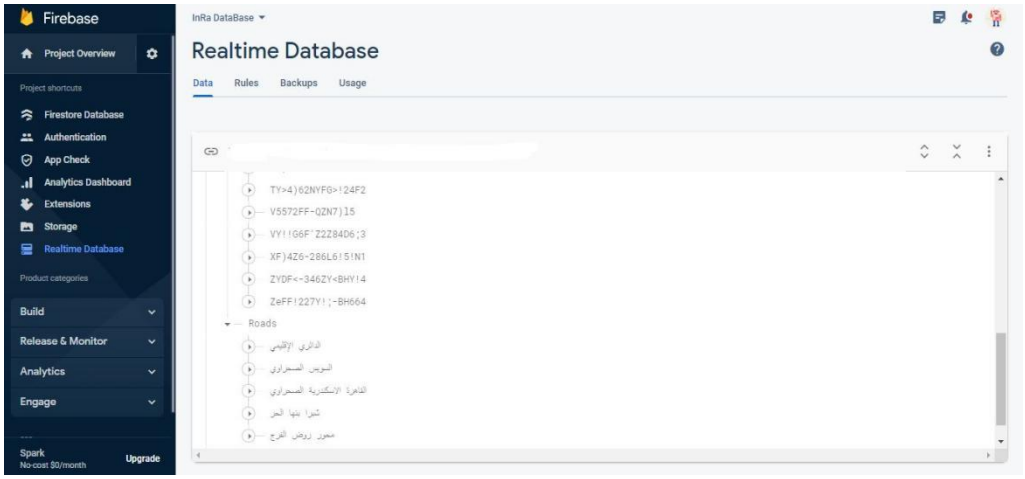


Figure 5-11 Road names

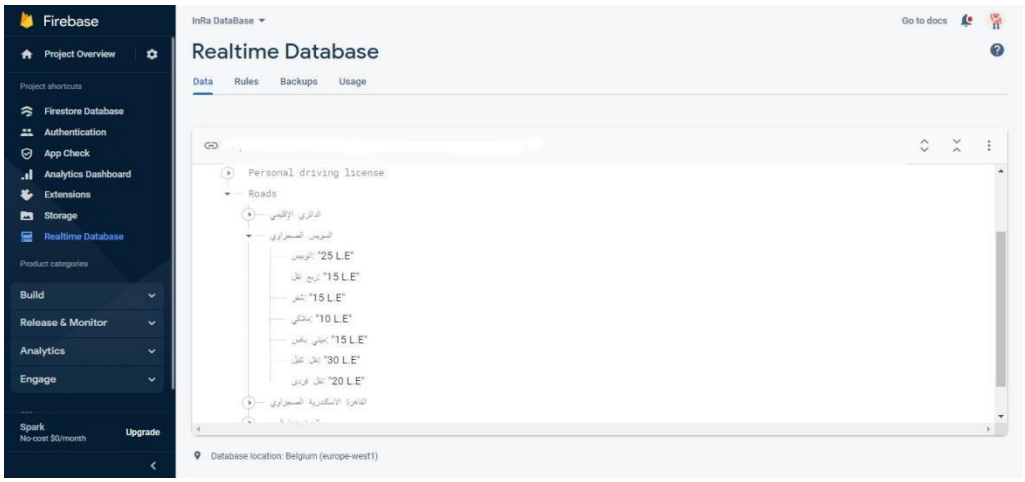


Figure 5-12 Road price information

Information about the price of the Suez Desert Road, for example
Any image I upload to the database, I load from storage:

Figure 5-13 Pictures stored on the storage

Select the car license that contains the car numbers.

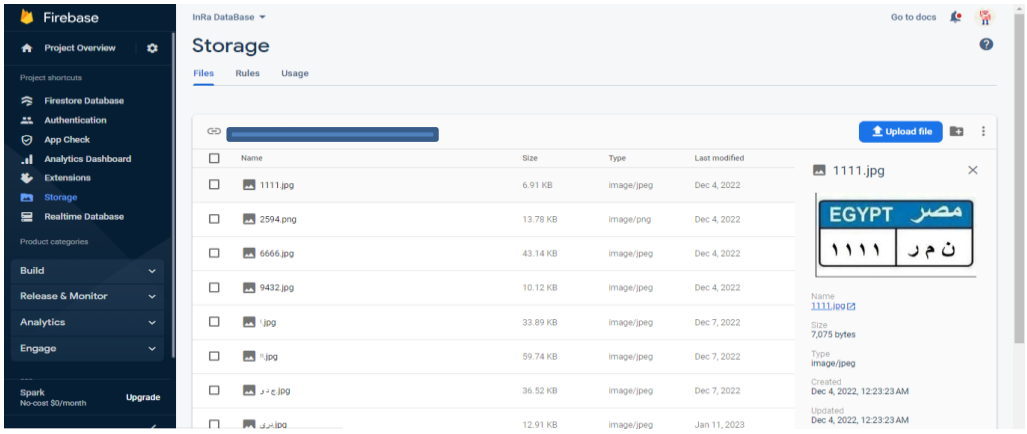


Figure 5-14 Select a photo from the photos stored on the storage

The same applies to a personal driver's license.

5.3 INRA-Application:

1. Home Page (English - Arabic):

Figure 5-15 Home page in English

Figure 5-16 Home page in Arabic

2. Car-ID (English - Arabic):

Figure 5-17 Car ID in English

Figure 5-18 Car ID in Arabic

3. Car Details (English - Arabic):

Figure 5-19 Car Details in Englis

Figure 5-20 Car Details in Arabic

4. Fines Page (English - Arabic):

Figure 5-21 Fines Page in English

Figure 5-22 Fines Page in Arabic

5. Roads Page (English - Arabic):

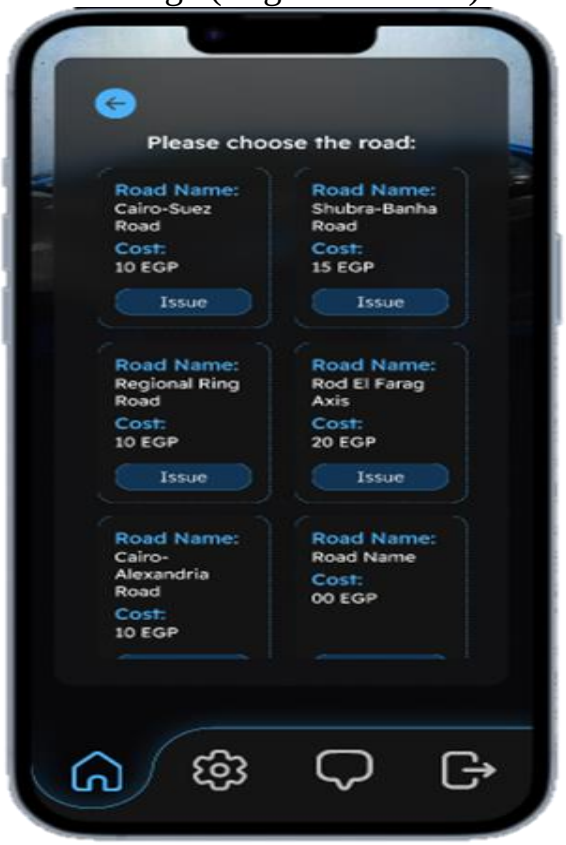


Figure 5-23 Road Page in English

Figure 5-24 Road Page in Arabic

6. Payment Methodes (English - Arabic):

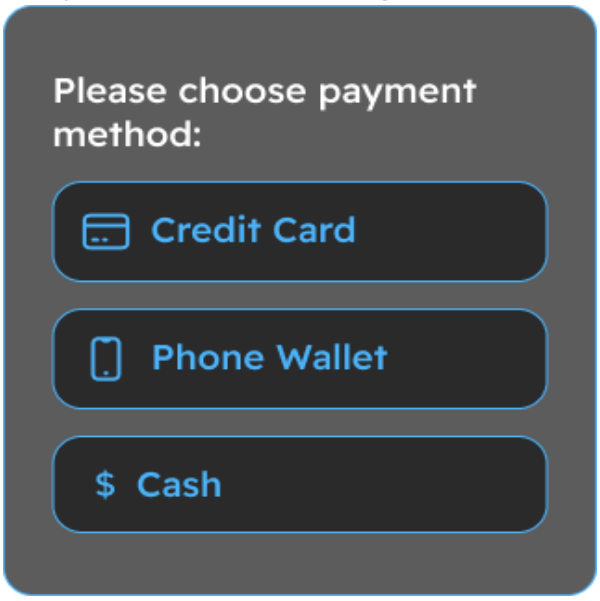


Figure 5-25 Payment Methodes in English

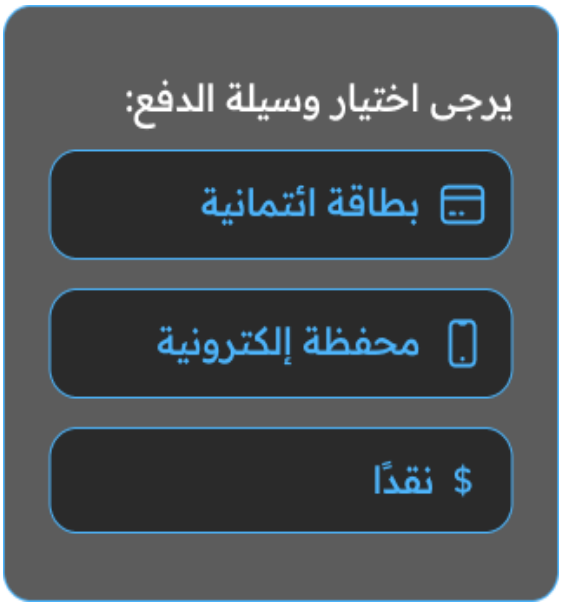


Figure 5-26 Payment Method in Arabic

7. Settings (English - Arabic):

Figure 5-27 Settinges in English

Figure 5-28 Settinges in Arabic

8. Support (English - Arabic):

Figure 5-29 Support in English

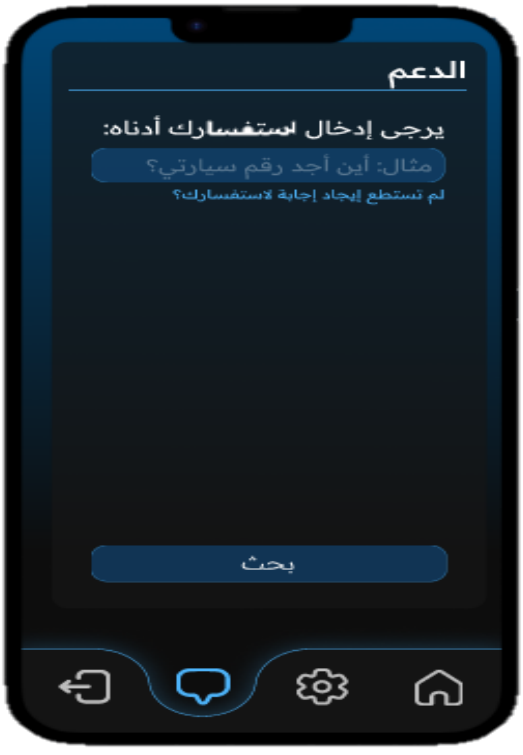


Figure 5-30 Support in Arabic

5.4 Project phases:

1) Project Start Page

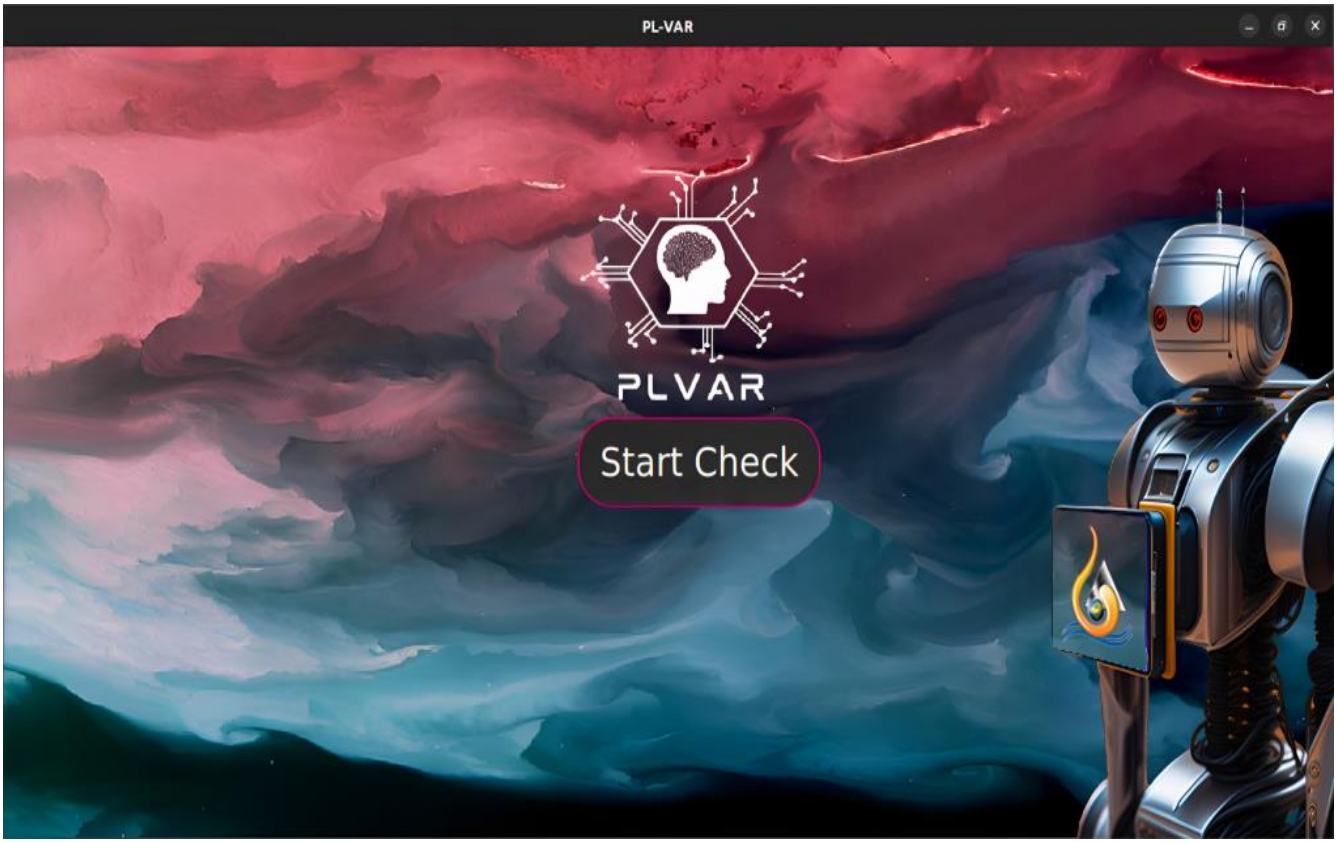


Figure 5-31 Start Check

When the user clicks on Start Check, the transition to the next page will begin to check the plate and color of the car.

2)

Figure 5-32 License plate and color

On this page, the license plate and the color of the car are examined, and after the examination, it will move to check the driver's seat belt.

3)

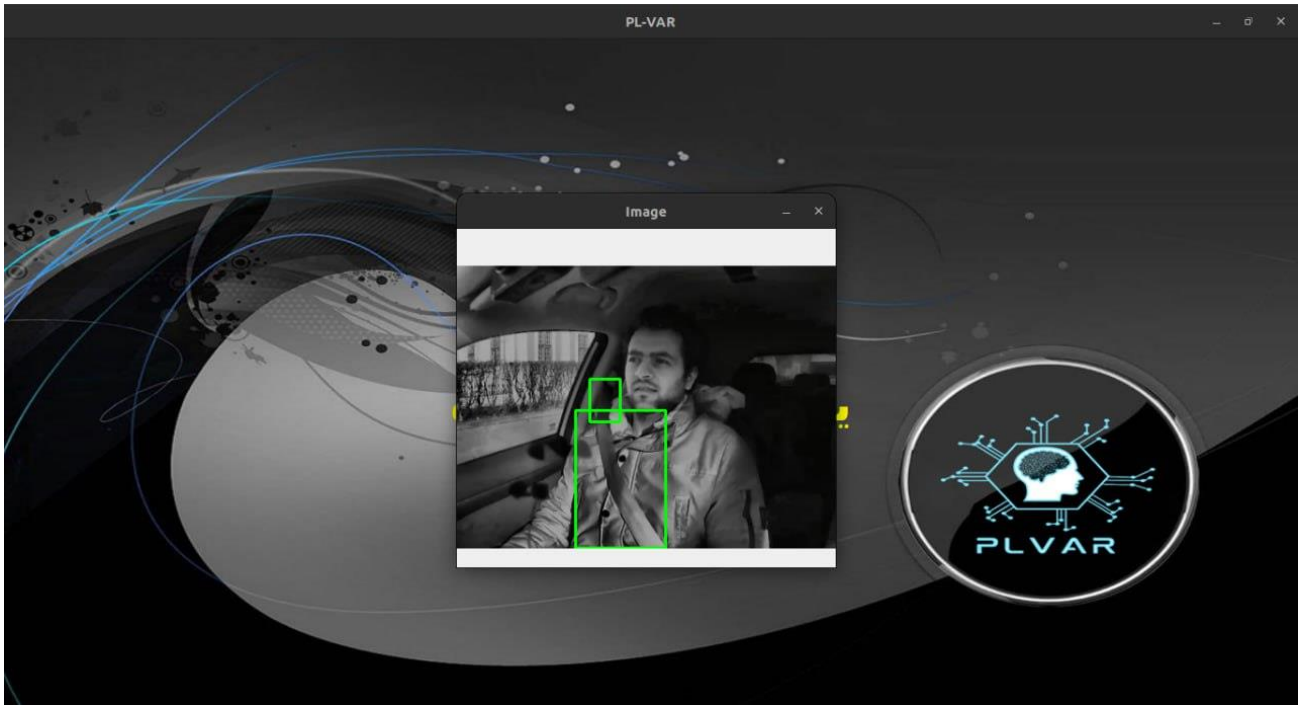


Figure 5-33seat belt

Here, the driver's seat belts are checked, and after the test, he will go to the car plate display page and check its data.

4)

Figure 5-34 Car plate

Here, view the car plate and verify its data, and then it will move to the barcode scan page to check the driver's license.

5)

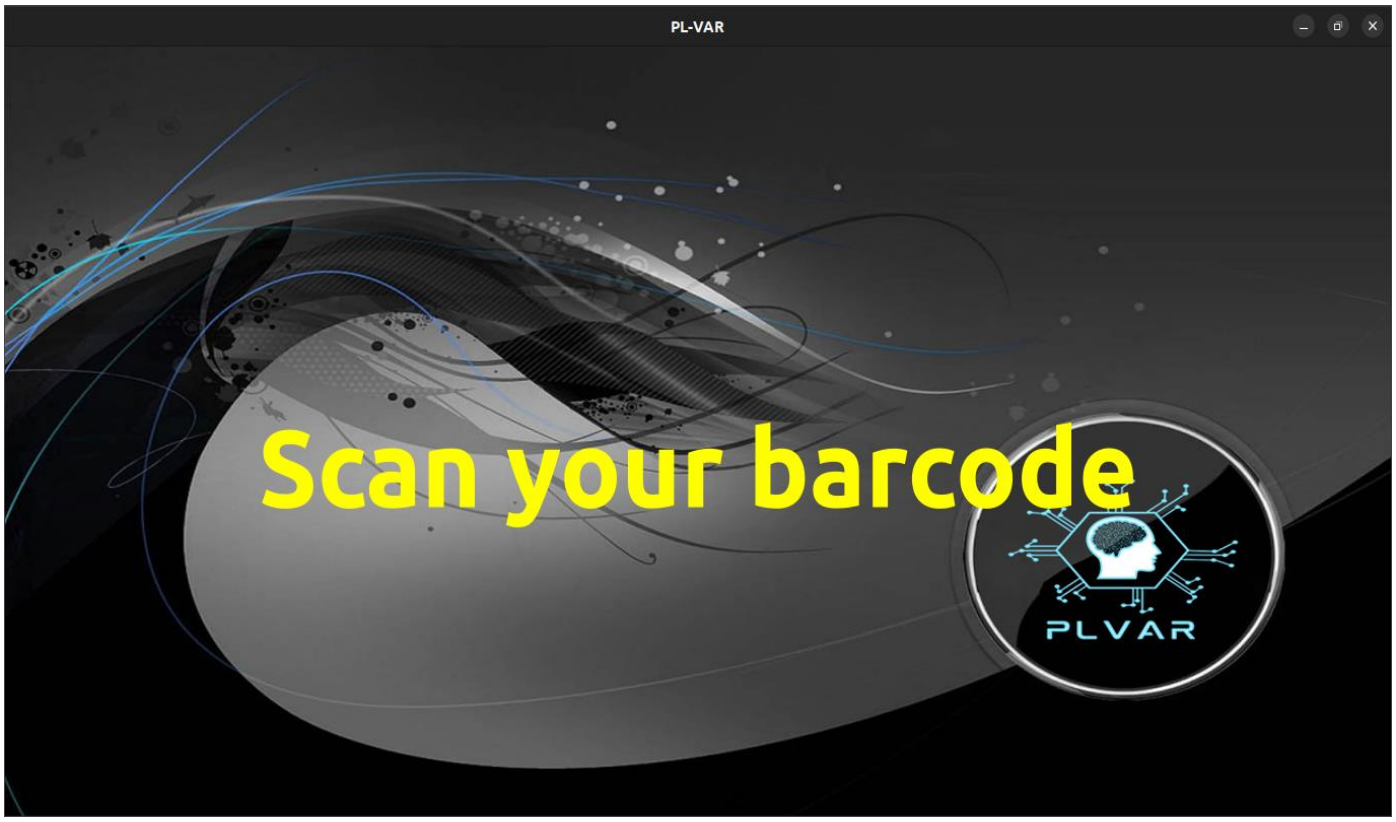


Figure 5-35 Scan your barcode

Here, he reads the barcode of the driver's license, then goes to the page to view and examine the driver's data.

6)

Figure 5-36 driver's data

Here the driver's data is displayed and verified and the face is checked if it matches or not, and whether it has committed crimes or is wanted by the police. If the person's data matches and is unwanted, then he is allowed to pass. If it does not match, a violation of its value will be recorded. If the person is wanted by the police, they will be reported.

5.5 Design Robot:

5.5.1 Head:

Size:
width: 181.0815 mm depth: 196.4366mm Height: 165.4533mm

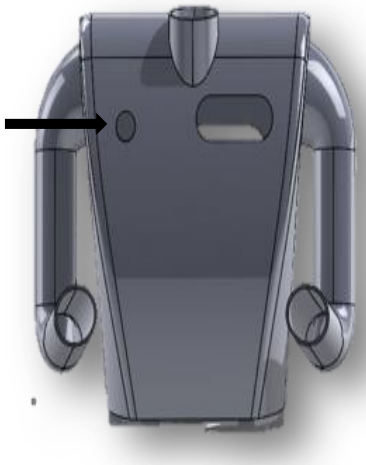
Figure 5-37 Head

Figure 5-38 Head From another angle

5.5.2 Chest

Size:

width: 334.3517mm depth:280.009mm Height; 181.5881mm



Refers to the camera that scans the license

Figure 5-39 Chest1 it's has a Camera that scan the license

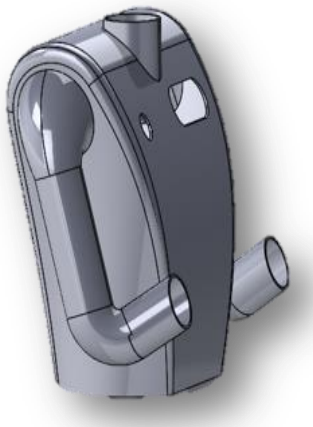


Figure 5-40 chest From another angle

Refers to the place where the motors are placed

Figure 5-41 chest it's has a place where the moters are placed

5.5.3 Frame of The Screen

Size:

width:399 mm depth: 346.1524mm Height: 135.7717mm

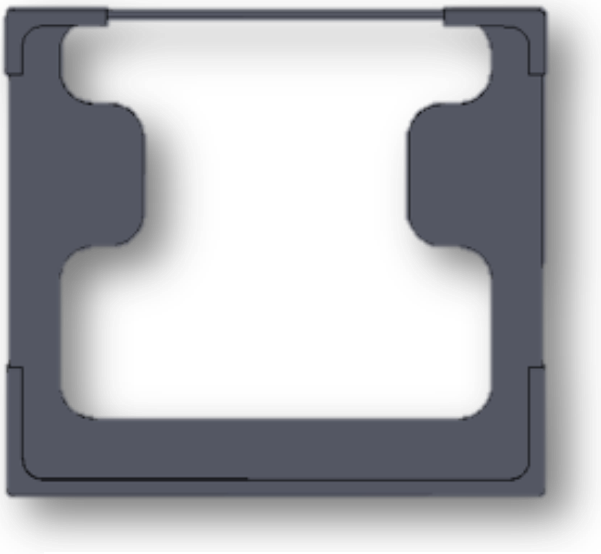


Figure 5-42 Frame of the screen

Figure 5-43 Frame of the screen from another angle

The screen that shows the driver's license and data

5.5.4 Cover of the chest

Size:

width: 198.329 mm depth: 280.0064mm Height: 166.274mm

Figure 5-44 Cover of the chest

Figure 5-45 Cover of the chest from another angle

5.5.5 The Waist1

Size:
width: 198.329 mm depth: 280.0064mm Height: 166.274mm

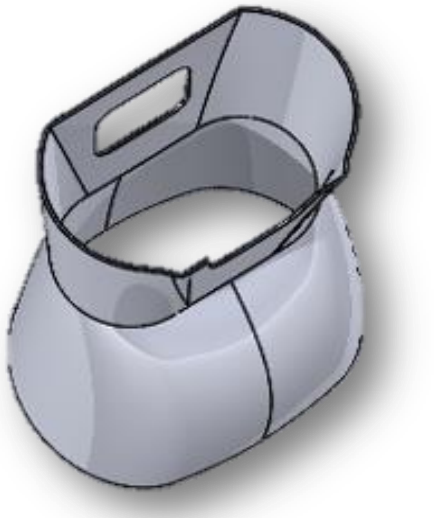


Figure 5-46 The Waist

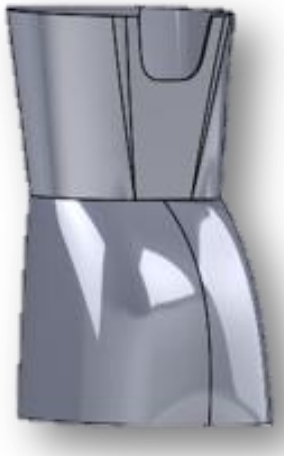


Figure 5-47 The waist from the another angle

5.5.6 The Waist 2

Size:
width:242.533 mm depth: 280.0021mm Height: 150.1977mm

Figure 5-49 The waist2

Figure 5-48 The waist2 from the another angle

5.5.7 The Base

Size:

width:441.6643 mm depth: 181.6031mm Height: 402.5279mm

Figure 5-50 The Base

5.5.8 Full Body

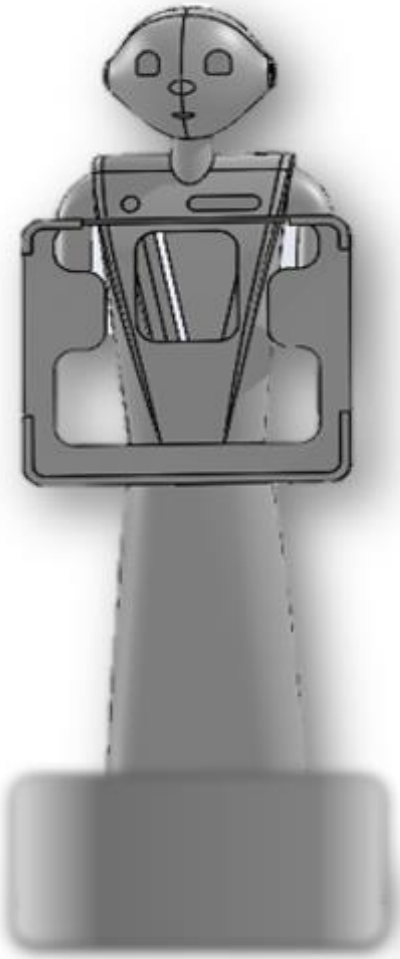


Figure 5-51 Full Body

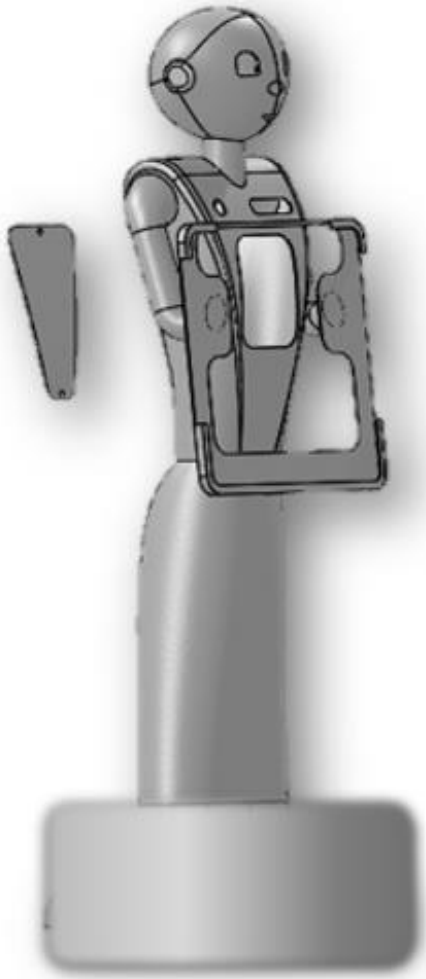


Figure 5-52 Full Body from the another angle

5.5.9 Actual Robot

Figure 5-53 Actual Robot

6 Future work and Conclusion:

6.1 Future work:

Future work for PLVAR could involve further enhancing and expanding the capabilities of the system to provide even more advanced features and benefits. Here are some potential areas of focus:

1. Integration with additional databases: Consider integrating the system with more comprehensive and up-to-date databases to improve the accuracy of license verification and violation detection. Collaborate with relevant authorities and organizations to access their databases and ensure the system has access to the most relevant and recent information.
2. Enhanced facial recognition and object detection: Invest in research and development to improve the facial recognition and object detection capabilities of the system. This could include refining the algorithms and leveraging advancements in computer vision technology to accurately detect and identify faces, licenses, and vehicles.
3. Expansion to other identification documents: Extend the system's capabilities beyond driver's licenses and incorporate the ability to verify other identification documents, such as passports or national ID cards. This would broaden the system's potential applications and increase its value for various industries and sectors.
4. Integration with additional payment methods: Explore partnerships and collaborations with payment service providers to expand the range of payment methods supported by the system. This could involve integrating with digital wallets, mobile payment platforms, or emerging technologies like cryptocurrency.

5. Advanced analytics and reporting: Develop analytical tools and reporting features within the system to provide insights and data visualization. This could help authorities and organizations gain valuable insights into driver behavior, violation patterns, and payment trends, enabling them to make informed decisions and improve overall operations.
6. Localization and internationalization: Adapt the system to meet the specific requirements and regulations of different regions and countries. This may involve translating the mobile app into multiple languages, accommodating local payment methods, and ensuring compliance with local data privacy and security regulations.
7. Continuous security enhancements: Maintain a strong focus on security and regularly update the system to address emerging threats and vulnerabilities. Conduct regular security audits, implement encryption and secure protocols, and stay up-to-date with best practices in data protection.
8. User feedback and usability improvements: Collect user feedback and conduct usability studies to identify areas for improvement in the mobile app and user experience. Continuously iterate and enhance the interface and functionality based on user insights to ensure a seamless and intuitive experience for drivers.

By focusing on these areas, PLVAR can stay at the forefront of license verification and payment processing technology, providing an even more robust and user-friendly solution for drivers and relevant authorities.

6.2 Conclusion:

PLVAR is a powerful system that combines hardware and software solutions to revolutionize the way we verify driver licenses and process payments. The system is designed to be user-friendly and intuitive, providing drivers with an easy way to pay their fees and verify their licenses. The real-time database feature ensures that changes made to the data are immediately reflected in the app, providing drivers with a seamless and hassle-free experience. PLVAR has truly revolutionized the way we verify driver licenses and process payments, making it easier and more convenient for drivers while also prioritizing their safety and security.

Commitment to designing the entire project according to **the IEEE SA - IEEE 1857.10-2021 voice +video, IEEE SA global standard Programming, IEEE CertifAIEd and AI.**

7 References

1. A. K. Emo, G. Matthews, and G. J. Funke, “The slow and the furious: Anger, stress and risky passing in simulated traffic congestion,” Transp. Res. Part F Traffic Psychol. Behav., vol. 42, pp. 1–14, 2016, doi: 10.1016/j.trf.2016.05.002.
2. R. Pfleiderer and M. Dietrich, “New roads generate new traffic,” Earthscan Read. World Transp. Policy Pract., vol. 1, no. 1, pp. 146–150, 2017, doi: 10.1108/13527619510075657.
3. S. Faye, C. Chaudet, and I. Demeure, “A distributed algorithm for adaptive traffic lights control,” IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC, no. September, pp. 1572–1577, 2012, doi: 10.1109/ITSC.2012.6338671.
4. M. Collotta, G. Pau, G. Scatà, and T. Campisi, “A dynamic traffic light management system based on wireless sensor networks for the reduction of the red-light running phenomenon,” J. Transp. Eng., vol. 15, no. 1, pp. 1–11, 2014, doi: 10.2478/ttj-2014-0001.
5. C. L. Philip Chen, J. Zhou, and W. Zhao, “A real-time vehicle navigation algorithm in sensor network environments,” IEEE Trans. Intell. Transp. Syst., vol. 13, no. 4, pp. 1657–1666, 2012, doi: 10.1109/TITS.2012.2201478.
6. M. G. H. Bell, V. Trozzi, S. H. Hosseinloo, G. Gentile, and A. Fonzone, “Time-dependent Hyperstar algorithm for robust vehicle navigation,” Transp. Res. Part A Policy Pract., vol. 46, no. 5, pp. 790–800, 2012, doi: 10.1016/j.tra.2012.02.002.
7. W. Min and L. Wynter, “Real-time road traffic prediction with spatio-temporal correlations,” Transp. Res. Part C Emerg. Technol., vol. 19, no. 4, pp. 606–616, 2011, doi: 10.1016/j.trc.2010.10.002.
8. Z. Liang and Y. Wakahara, “City traffic prediction based on real-time traffic information for intelligent transport systems,” 2013 13th Int. Conf. ITS Telecommun. ITST 2013, pp. 378–383, 2013, doi: 10.1109/ITST.2013.6685576.
9. Intelligent Transportation Systems – Problems and. .
10. J. Barros, M. Araujo, and R. J. F. Rossetti, “Short-term real-time traffic prediction methods: A survey,” 2015 Int. Conf. Model. Technol. Intell. Transp. Syst. MT-ITS 2015, no. June, pp. 132–139, 2015, doi: 10.1109/MTITS.2015.7223248.
11. X. Chen and R. Chen, “A Review on Traffic Prediction Methods for Intelligent Transportation System in Smart Cities,” Proc. - 2019 12th Int. Congr. Image Signal Process. Biomed. Eng. Informatics, CISP-BMEI 2019, no. 4, 2019, doi: 10.1109/CISP-BMEI48845.2019.8965742.
12. P. Chhatpar, N. Doolani, S. Shahani, and R. L. Priya, “Machine Learning Solutions to Vehicular Traffic Congestion,” 2018 Int. Conf. Smart City Emerg. Technol. ICSCET 2018, no. December 2020, pp. 1–4, 2018, doi: 10.1109/ICSCET.2018.8537260.

13. A. M. De Souza, R. S. Yokoyama, L. C. Botega, R. I. Meneguette, and L. A. Villas, "SCORPION: A solution using cooperative rerouting to prevent congestion and improve traffic condition," Proc. - 15th IEEE Int. Conf. Comput. Inf. Technol. CIT 2015, 14th IEEE Int. Conf. Ubiquitous Comput. Commun. IUCC 2015, 13th IEEE Int. Conf. Dependable, Auton. Se, no. October, pp. 497–503, 2015, doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.71.
14. O. Avatefipour and F. Sadry, "Traffic Management System Using IoT Technology - A Comparative Review," IEEE Int. Conf. Electro Inf. Technol., vol. 2018-May, pp. 1041–1047, 2018, doi: 10.1109/EIT.2018.8500246.
15. S. K. S. Fan, C. J. Su, H. T. Nien, P. F. Tsai, and C. Y. Cheng, "Using machine learning and big data approaches to predict travel time based on historical and real-time data from Taiwan electronic toll collection," Soft Comput., vol. 22, no. 17, pp. 5707–5718, 2018, doi: 10.1007/s00500-017-2610-y.
16. R. Bruno and M. Nurchis, "Robust and efficient data collection schemes for vehicular multimedia sensor Networks," 2013 IEEE 14th Int. Symp. a World Wireless, Mob. Multimed. Networks, WoWMoM 2013, 2013, doi: 10.1109/WoWMoM.2013.6583399.
17. M. Saqib and C. Lee, "Traffic control system using wireless sensor network," Int. Conf. Adv. Commun. Technol. ICACT, vol. 1, pp. 352–357, 2010.
18. M. Won, S. Zhang, and S. H. Son, "WiTraffic : Low-cost and Non-intrusive Traffic Monitoring System Using WiFi," 2017.
19. P. Perez-murueta, G. Alfonso, and C. Cardenas, "applied sciences Deep Learning System for Vehicular Re-Routing and Congestion Avoidance," 2019.
20. O. Choi, S. Kim, J. Jeong, H. Lee, and S. Chong, "Delay-Optimal Data Forwarding in Vehicular Sensor Networks."

21. Raspberry Pi Foundation: About Us. Retrieved from https://www.raspberrypi.org/about/
22. Raspberry Pi. (2022, April 13). In Wikipedia. Retrieved from https://en.wikipedia.org/wiki/Raspberry_Pi
23. "Electric Motors and Drives: Fundamentals, Types and Applications" by Austin Hughes and Bill Drury (ISBN: 0081004024)
24. Upton, Eben (24 June 2019). "Raspberry Pi 4 on sale now from $35". Raspberry Pi Foundation.
25. Upton, Eben (28 May 2020). "8GB Raspberry Pi 4 on sale now at $75". Raspberry Pi Blog. Retrieved 28 May 2020.
26. Hattersley, Lucy (15 November 2018). "Raspberry Pi 4, 3A+, Zero W – specs, benchmarks & thermal tests". The MagPi magazine. Raspberry Pi Trading Ltd. Retrieved 28 May 2020.
27. "DATASHEET – Raspberry Pi Compute Module 3+" (PDF). Raspberrypi.org. 1 January

2019. Retrieved 28 May 2020.
28. "Raspberry Pi 4 Tech Specs". Raspberry Pi. Retrieved 26 June 2019.
29. "Raspberry Pi Zero: the $5 Computer". Raspberry Pi Foundation. 26 November 2015. Retrieved 26 November 2015.
30. "Pi Bootmodes". Only Supports Pi 2 v1.2 and up.
31. "The Raspberry Pi 4 doesn't work with all USB-C cables".
32. "New $10 Raspberry Pi Zero comes with Wi-Fi and Bluetooth". Ars Technica. Retrieved 28 February 2017.
33. "Meet Raspberry Silicon: Raspberry Pi Pico now on sale at $4". 21 January 2021.
34. "Raspberry Pi Documentation". Raspberrypi.com. Retrieved 2 March 2022.
35. "Flashing the Compute Module eMMC – Raspberry Pi Documentation". www.raspberrypi.org. Retrieved 14 July 2021.
36. "Performance – measures of the Raspberry Pi's performance". RPi Performance. eLinux.org. Retrieved 30 March 2014.
37. Tamplin, James. "Firebase is Joining Google!". Firebase, Inc. Retrieved October 21, 2014.
38. "Google Cloud Messaging - official website". Retrieved July 20, 2016.
39. Firebase Realtime Database documentation: https://firebase.google.com/docs/database
40. Firebase Authentication documentation: https://firebase.google.com/docs/auth
41. Firebase Cloud Storage documentation: https://firebase.google.com/docs/storage
42. Firebase Cloud Messaging documentation: https://firebase.google.com/docs/cloud-messaging
43. Firebase Analytics documentation: https://firebase.google.com/docs/analytics
44. Firebase Local Emulator Suite Documentation: https://firebase.google.com/docs/emulator-suite
45. Cloud Firestore Emulator Documentation: https://firebase.google.com/docs/emulator-suite/install_and_configure#installing_the_cloud_firestore_emulator
46. Cloud Storage Emulator Documentation: https://firebase.google.com/docs/emulator-suite/install_and_configure#installing_the_cloud_storage_emulator
47. Authentication Emulator Documentation: https://firebase.google.com/docs/emulator-suite/install_and_configure#installing_the_authentication_emulator
48. JSON (JavaScript Object Notation) official website: https://www.json.org/json-en.html
49. Structuring your data for Cloud Firestore: https://firebase.google.com/docs/firestore/data-model
50. Differences between Realtime Database and Cloud Firestore: https://firebase.google.com/docs/database/rtdb-vs-firestore#differences_between_realtime_database_and_cloud_firestore
51. Firebase documentation on Realtime Database triggers for Cloud Functions: https://firebase.google.com/docs/functions/database-events
52. "FCM Architectural Overview". Firebase. Retrieved November 17, 2020.

53. Sharma, R. (2020). What is GCM and FCM? (Differences and Limitations). Retrieved 2 October 2020, from https://www.izooto.com/blog/everything-that-you-need-to-know-about-firebase-cloud-messaging-platform
54. "Understanding message delivery". Firebase. Retrieved November 17, 2020.
55. Esposito, Christian; Palmieri, Francesco; Choo, Kim-Kwang Raymond (March 2018). "Cloud Message Queueing and Notification: Challenges and Opportunities". IEEE Cloud Computing. **5** (2): 11–16. doi:10.1109/mcc.2018.022171662. ISSN 2325-6095. S2CID 19248242.
56. Li, Na; Du, Yanhui; Chen, Guangxuan (December 2013). "Survey of Cloud Messaging Push Notification Service". 2013 International Conference on Information Science and Cloud Computing Companion. IEEE: 273–279. doi:10.1109/iscc-c.2013.132. ISBN 978-1-4799-5245-8. S2CID 15771293.
57. Mokar, M. A., Fageeri, S. O., & Fattoh, S. E. (2019, September). Using Firebase Cloud Messaging to Control Mobile Applications. In 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE) (pp. 1-5). IEEE
58. https://firebase.google.com/docs/cloud-messaging/concept-options Firebase. Retrieved November 17, 2020.
59. https://www.v7labs.com/blog/object-detection-guide
60. https://www.baeldung.com/cs/yolo-algorithm
61. https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/
62. Diep, Eric (December 5, 2012). "Drake Wants Royalties for "YOLO"". XXL Magazine. Retrieved March 3, 2013.
63. "YOLO". USPTO.report. Retrieved Jan 9, 2020.
64. https://en.wikipedia.org/wiki/YOLO_(The_Lonely_Island_song) iTunes. January 27, 2013. Retrieved December 31, 2014.
65. Pellejero, Sebastian; Quiroz-Gutierrez, Marco (2021-01-26). https://pt.wikipedia.org/wiki/YOLO . Retrieved 2021-01-28.
66. Burns, Ashley (January 6, 2013). "We Have Some Bad News For Drake Regarding The YOLO Wars". Uproxx.com. Retrieved March 3, 2013.
67. "Flutter is a cutting-edge open-source platform" - Flutter website: https://flutter.dev/
68. "Flutter's fast development cycle, streamlined testing process, and comprehensive documentation" - Flutter documentation: https://flutter.dev/docs
69. https://stackoverflow.com/questions/67581144/how-to-create-a-drop-down-view-text-faq-flutter-ui . Retrieved 2018-08-08.
70. Amadeo, Ron (8 May 2017). "Google's "Fuchsia" smartphone OS dumps Linux, has a wild new UI". Ars Technica. Retrieved 18 March 2018.
71. Amadeo, Ron (2018-02-27). "Google starts a push for cross-platform app development with Flutter SDK". Ars Technica. Retrieved 2021-06-11.

72. https://flutternews.medium.com/with-flutter-google-aims-dart-to-mobile-app-cross-development-274a9ff1fec8 . InfoQ. Retrieved 2022-03-17.
73. https://docs.flutter.dev/platform-integration/web/building . docs.flutter.dev. Retrieved 2022-10-06.
74. https://docs.flutter.dev/packages-and-plugins/using-packages . docs.flutter.dev. Retrieved 2022-10-06.
75. https://api.flutter.dev/flutter/foundation/foundation-library.html . docs.flutter.dev. Retrieved 2017-12-13.
76. "Material Design Widgets - Flutter". flutter.dev. Retrieved 2017-12-13.
77. "Cupertino (iOS-style) Widgets - Flutter". flutter.dev. Retrieved 2017-12-13.
78. "Google announces Flutter 1.0, the first stable release of its cross-platform mobile development toolkit"
79. https://www.androidpolice.com/2018/12/05/google-announces-flutter-1-0-the-first-stable-release-of-its-cross-platform-mobile-development-toolkit/ . Android Police. 2018-12-05. Retrieved 2022-03-17.
80. OpenCV documentation on cv2.VideoCapture function: https://docs.opencv.org/master/d8/dfe/classcv_1_1VideoCapture.html
81. OpenCV documentation on color histograms: https://docs.opencv.org/master/d1/db7/tutorial_py_histogram_begins.html
82. Scikit-learn documentation on k-nearest neighbors classifier: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
83. Color histograms for image analysis:Swain, M. J., & Ballard, D. H. (1991). Color indexing. International Journal of Computer Vision, 7(1), 11-32.
84. Machine learning for color classification:Jain, V., & Learned-Miller, E. (2013). FV-CNN: Fisher vectors for convolutional neural networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.
85. CLAHE (Contrast Limited Adaptive Histogram Equalization) in OpenCV: https://docs.opencv.org/master/d5/daf/tutorial_py_histogram_equalization.html
86. Non-maximum suppression in object detection: https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/
87. Database https://firebase.google.com/products/realtime-database
88. https://firebase.google.com/docs/firestore
89. https://wiki.gdevelop.io/gdevelop5/all-features/firebase/firestore/#regulating-access
90. "The Dart Team Welcomes TypeScript". Retrieved 22 February 2020.
91. "Dart SDK Tags". GitHub.
92. "A Tour of the Dart Language". dart.dev. Retrieved 2018-08-09.
93. "Dart, a new programming language for structured web programming", GOTO conference (presentation) (opening keynote), Århus conference, 2011-10-10

94. Ladd, Seth. "What is Dart". What is Dart?. O'Reilly. Retrieved August 16, 2014.
95. "Dart 1.0: A stable SDK for structured web apps". news.dartlang.org. Retrieved 2018-08-08.
96. Seth Ladd. "Dart News & Updates". dartlang.org.
97. Moore, Kevin (2018-08-07). "Announcing Dart 2 Stable and the Dart Web Platform". Dart. Retrieved 2018-08-08.
98. "Concurrency in Dart". dart.dev. Retrieved 2023-05-12.
99. "Google Releases Dart Editor for Windows, Mac OS X, and Linux". Archived from the original on 2013-12-03. Retrieved 2011-11-29.
100. "Dart plugin for Eclipse is Ready for Preview".
101. Ladd, Seth (2015-04-30). "The present and future of editors and IDEs for Dart". Dart News & Updates. Retrieved 2015-05-18.
102. Michael Barr. "Embedded Systems Glossary". Neutrino Technical Library. Retrieved 2007-04-21.
103. Heath, Steve (2003). Embedded systems design. EDN series for design engineers (2 ed.). Newnes. p. 2. ISBN 978-0-7506-5546-0. An embedded system is a microprocessor based system that is built to control a function or a range of functions.
104. Michael Barr; Anthony J. Massa (2006). "Introduction". Programming embedded systems: with C and GNU development tools. O'Reilly. pp. 1–2. ISBN 978-0-596-00983-0.
105. Embedded Systems Dell OEM Solutions | Dell. Content.dell.com (2011-01-04). Retrieved on 2013-02-06.
106. "Working across Multiple Embedded Platforms" (PDF). clarinox. Archived (PDF) from the original on 2011-02-19. Retrieved 2010-08-17.
107. https://www.knightscope.com/
108. https://www.cobaltrobotics.com/
109. https://www.pal-robotics.com/
110. https://www.softbankrobotics.com/emea/en/pepper

8 Appendix

8.1 Face recognition

8.1.1 Train model

```
import cv2
import face_recognition
import pickle
import os
from imutils import paths

# Define the path to the dataset folder where images are stored
dataset_path = "dataset"

# Function to train the facial recognition model
def train_model():
    print("[INFO] Start processing faces...")

    # Get the paths of all images in the dataset folder
    image_paths = list(paths.list_images(dataset_path))

    # Initialize lists to store encodings and corresponding names
    known_encodings = []
    known_names = []

    # Loop over the image paths
    for (i, image_path) in enumerate(image_paths):
        # Extract the person name from the image path
        print("[INFO] Processing image {}/{}".format(i + 1, len(image_paths)))
        name = image_path.split(os.path.sep)[-2]

        # Load the input image and convert it from BGR to RGB
        image = cv2.imread(image_path)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Detect the bounding boxes of faces in the image
        boxes = face_recognition.face_locations(rgb, model="hog")

        # Compute the facial encodings for the detected faces
        encodings = face_recognition.face_encodings(rgb, boxes)

        # Loop over the encodings
```

```
        for encoding in encodings:
            # Add the encoding and corresponding name to the lists
            known_encodings.append(encoding)
            known_names.append(name)

    # Serialize the encodings and names to a pickle file
    print("[INFO] Serializing encodings...")
    data = {"encodings": known_encodings, "names": known_names}
    with open("encodings.pickle", "wb") as f:
        f.write(pickle.dumps(data))

# Train the facial recognition model
train_model()
```

8.1.2 Recognition

```
import cv2
import face_recognition

# Load the known encodings and names from a database or file
known_encodings = load_known_encodings()
known_names = load_known_names()

# Initialize the camera
video_capture = cv2.VideoCapture(0)

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    # Convert the frame from BGR to RGB
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Detect the faces in the frame
    face_locations = face_recognition.face_locations(rgb_frame, model="hog")

    # Iterate over the detected faces
    for (top, right, bottom, left) in face_locations:
        # Draw a rectangle around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

        # Encode the face
        face_encoding = face_recognition.face_encodings(rgb_frame, [(top, right, bottom,
```

```
left)])[0]

        # Compare the face encoding with the known encodings
        matches = face_recognition.compare_faces(known_encodings, face_encoding)
        name = "Unknown"

        # Find the best match
        if True in matches:
            matched_indexes = [i for (i, match) in enumerate(matches) if match]
            face_distances = face_recognition.face_distance(known_encodings, face_encoding)
            best_match_index = min(matched_indexes, key=lambda x: face_distances[x])
            name = known_names[best_match_index]

        # Display the name on the frame
        cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0,
255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Facial Recognition', frame)

    # Break the loop if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the video capture object and close the windows
video_capture.release()
cv2.destroyAllWindows()
```

8.2 ELPER

```
import cv2
import pytesseract

# Set the path to the license plate classifier file
plate_cascade = cv2.CascadeClassifier('haarcascade_russian_plate_number.xml')

# Initialize the video camera
video_capture = cv2.VideoCapture(0)

while True:
    # Read the current video frame
    ret, frame = video_capture.read()

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Search for license plates in the frame using the license plate classifier
    plates = plate_cascade.detectMultiScale(gray, 1.1, 4)

    # Iterate over the detected license plates
    for (x, y, w, h) in plates:
        # Extract the license plate region from the current frame
        plate_image = frame[y:y + h, x:x + w]

        # Convert the image to grayscale
        plate_image_gray = cv2.cvtColor(plate_image, cv2.COLOR_BGR2GRAY)

        # Apply OCR technique on the image to extract the text from it
        plate_text = pytesseract.image_to_string(plate_image_gray, config='--psm 7')

        # Display the extracted text
        print("License Plate Number:", plate_text)

        # Display a bounding box around the license plate in the current frame
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)

    # Display the current frame
    cv2.imshow('Car Plate Recognition', frame)

    # Wait for 'q' key press to exit the application
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the video camera and close the windows
video_capture.release()
cv2.destroyAllWindows()
```

8.3 Seat Belt Detection

python
import cv2
import numpy as np

```
# Load YOLOv8 model
model = cv2.dnn.readNetFromDarknet('yolov8.cfg', 'yolov8.weights')

# Load image
image = cv2.imread('image.jpg')

# Preprocess image
blob = cv2.dnn.blobFromImage(image, 1/255.0, (416, 416), swapRB=True, crop=False)
model.setInput(blob)

# Perform object detection
layer_names = model.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in model.getUnconnectedOutLayers()]
outputs = model.forward(output_layers)

# Post-processing
boxes = []
confidences = []
class_ids = []

for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]

        if confidence > 0.5 and class_id == 0:  # Assuming class_id 0 represents seat belts
            center_x = int(detection[0] * image.shape[1])
            center_y = int(detection[1] * image.shape[0])
            width = int(detection[2] * image.shape[1])
            height = int(detection[3] * image.shape[0])

            x = int(center_x - width / 2)
            y = int(center_y - height / 2)

            boxes.append([x, y, width, height])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# Non-maximum suppression
indices = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5, nms_threshold=0.4)
```

```
# Draw bounding boxes
for i in indices:
    x, y, w, h = boxes[i[0]]
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the result
cv2.imshow('Seat Belt Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Example 2: Image Preprocessing with OpenCV

```
python
import cv2

# Load image
image = cv2.imread('image.jpg')

# Resize image
resized_image = cv2.resize(image, (416, 416))

# Convert image to grayscale
gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)

# Apply Gaussian blur
blurred_image = cv2.GaussianBlur(resized_image, (5, 5), 0)

# Apply thresholding
_, thresholded_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)

# Display the preprocessed images
cv2.imshow('Resized Image', resized_image)
cv2.imshow('Grayscale Image', gray_image)
cv2.imshow('Blurred Image', blurred_image)
cv2.imshow('Thresholded Image', thresholded_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8.4 car color recognition

python
```
import cv2
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

# Function to extract color histogram features from an image
def extract_color_histogram(image, bins):
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv_image], [0, 1], None, bins, [0, 180, 0, 256])
    cv2.normalize(hist, hist)
    return hist.flatten()

# Loading the dataset and preprocessing the images
dataset = [...]  # List of tuples (image, color_label)
preprocessed_dataset = []

for image, color_label in dataset:
    # Preprocess the image (resize, convert to HSV, etc.)
    preprocessed_image = preprocess_image(image)
    preprocessed_dataset.append((preprocessed_image, color_label))

# Extracting color histogram features from the preprocessed dataset
histogram_features = []
labels = []

for image, color_label in preprocessed_dataset:
    histogram = extract_color_histogram(image, bins=16)  # Adjust the number of bins as needed
    histogram_features.append(histogram)
    labels.append(color_label)

# Converting the features and labels to numpy arrays
X = np.array(histogram_features)
y = np.array(labels)

# Training the KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # Adjust the value of K as needed
knn_classifier.fit(X, y)

# Testing the car color recognition system on a new image
test_image = cv2.imread('test_image.jpg')
preprocessed_test_image = preprocess_image(test_image)
test_histogram = extract_color_histogram(preprocessed_test_image, bins=16)  # Adjust the number
of bins as needed

predicted_color = knn_classifier.predict([test_histogram])[0]
print("Predicted color:", predicted_color)
```

8.5 GUI

PYQT5 We used this library in the GUI

1)

```
#Create button
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

# Function to be executed when the button is clicked
def button_clicked():
    print('Button clicked!')

if name == 'main':
    app = QApplication(sys.argv)

    # Create the main window
    window = QWidget()
    window.setWindowTitle('PyQt5 Button Example')

    # Create a button
    button = QPushButton('Click Here', window)
    button.setGeometry(50, 50, 100, 30)  # Set position and size
    button.clicked.connect(button_clicked)  # Connect the function to the button click

    # Show the window
    window.show()

    # Run the application
    sys.exit(app.exec_())
```

2)

```
#Move between pages using button
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton,
QLabel, QStackedWidget

if name == 'main':
    app = QApplication(sys.argv)

    # Create the main window
    window = QWidget()
    window.setWindowTitle('Page Navigation with PyQt5')
```

```
# Create a QStackedWidget to manage the pages
stacked_widget = QStackedWidget()

# Create the first page
page1 = QWidget()
layout1 = QVBoxLayout()
label1 = QLabel('Page 1')
layout1.addWidget(label1)
page1.setLayout(layout1)

# Create the second page
page2 = QWidget()
layout2 = QVBoxLayout()
label2 = QLabel('Page 2')
layout2.addWidget(label2)
page2.setLayout(layout2)

# Add the pages to the QStackedWidget
stacked_widget.addWidget(page1)
stacked_widget.addWidget(page2)

# Create buttons for page navigation
button1 = QPushButton('Page 1')
button1.clicked.connect(lambda: stacked_widget.setCurrentIndex(0))

button2 = QPushButton('Page 2')
button2.clicked.connect(lambda: stacked_widget.setCurrentIndex(1))

# Set the first page as the initially visible page
stacked_widget.setCurrentIndex(0)

# Set up the layout of the main window
layout = QVBoxLayout()
layout.addWidget(stacked_widget)
layout.addWidget(button1)
layout.addWidget(button2)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())
```

3)

```
#Move between pages without button
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel,
QStackedWidget

if name == 'main':
    app = QApplication(sys.argv)

    # Create the main window
    window = QWidget()
    window.setWindowTitle('Page Navigation with PyQt5')

    # Create a QStackedWidget to manage the pages
    stacked_widget = QStackedWidget()

    # Create the first page
    page1 = QWidget()
    layout1 = QVBoxLayout()
    label1 = QLabel('Page 1')
    layout1.addWidget(label1)
    page1.setLayout(layout1)

    # Create the second page
    page2 = QWidget()
    layout2 = QVBoxLayout()
    label2 = QLabel('Page 2')
    layout2.addWidget(label2)
    page2.setLayout(layout2)
    # Add the pages to the QStackedWidget
    stacked_widget.addWidget(page1)
    stacked_widget.addWidget(page2)

    # Set the first page as the initially visible page
    stacked_widget.setCurrentIndex(0)

    # Set up the layout of the main window
    layout = QVBoxLayout()
    layout.addWidget(stacked_widget)
    window.setLayout(layout)

    # Show the window
    window.show()
```

```
    # Switch to the second page after a certain delay (in milliseconds)
    QTimer.singleShot(2000, lambda: stacked_widget.setCurrentIndex(1))

    # Run the application
    sys.exit(app.exec_())
```

4)

```
#View an image
import sys
from PyQt5.QtWidgets import QApplication, QLabel, QWidget
from PyQt5.QtGui import QPixmap

if name == 'main':
    app = QApplication(sys.argv)

    # Create the main window
    window = QWidget()
    window.setWindowTitle('Display Image')

    # Load the image
    image_path = 'path/to/your/image.jpg'
    pixmap = QPixmap(image_path)

    # Create a QLabel widget and set the image pixmap
    label = QLabel(window)
    label.setPixmap(pixmap)

    # Show the window
    window.show()

    # Run the application
    sys.exit(app.exec_())
```

5)

```
#Create Table
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QTableWidget,
QTableWidgetItem

if name == 'main':
    app = QApplication(sys.argv)

    # Create the main window
    window = QWidget()
```

```
window.setWindowTitle('PyQt5 Table Example')

# Create a table widget
table = QTableWidget()

# Set the number of rows and columns in the table
table.setRowCount(3)
table.setColumnCount(2)

# Set the table headers
table.setHorizontalHeaderLabels(['Column 1', 'Column 2'])

# Set the table data
table.setItem(0, 0, QTableWidgetItem('Cell 1,1'))
table.setItem(0, 1, QTableWidgetItem('Cell 1,2'))
table.setItem(1, 0, QTableWidgetItem('Cell 2,1'))
table.setItem(1, 1, QTableWidgetItem('Cell 2,2'))
table.setItem(2, 0, QTableWidgetItem('Cell 3,1'))
table.setItem(2, 1, QTableWidgetItem('Cell 3,2'))

# Set the table to resize to fit its contents
table.resizeColumnsToContents()
table.resizeRowsToContents()

# Set up the layout of the main window
layout = QVBoxLayout()
layout.addWidget(table)
window.setLayout(layout)

# Show the window
window.show()

# Run the application
sys.exit(app.exec_())
```

6)

```
import cv2
import face_recognition
from PyQt5.QtCore import Qt, QTimer
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import QApplication, QLabel, QVBoxLayout, QWidget

class FacialRecognitionApp(QWidget):
```

```
def init(self):
    super().init()
    self.setWindowTitle("Facial Recognition")

    # Create a label to display the video feed
    self.video_label = QLabel(self)

    # Create a layout and set it as the main layout for the window
    layout = QVBoxLayout(self)
    layout.addWidget(self.video_label)
    self.setLayout(layout)

    # Load the known encodings and names from a database or file
    self.known_encodings = load_known_encodings()
    self.known_names = load_known_names()

    # Initialize the video capture object
    self.video_capture = cv2.VideoCapture(0)

    # Start the timer to update the video feed
    self.timer = QTimer(self)
    self.timer.timeout.connect(self.update_frame)
    self.timer.start(100)  # Update every 100 milliseconds

def load_known_encodings(self):
    # Load the known encodings from a database or file
    # and return the encodings
    pass

def load_known_names(self):
    # Load the known names from a database or file
    # and return the names
    pass

def update_frame(self):
    # Capture frame-by-frame
    ret, frame = self.video_capture.read()

    # Convert the frame from BGR to RGB
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Detect the faces in the frame
    face_locations = face_recognition.face_locations(rgb_frame, model="hog")
    # Iterate over the detected faces
```

```
        for (top, right, bottom, left) in face_locations:
            # Draw a rectangle around the face
            cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

            # Encode the face
            face_encoding = face_recognition.face_encodings(rgb_frame, [(top, right, bottom, left)])[0]

            # Compare the face encoding with the known encodings
            matches = face_recognition.compare_faces(self.known_encodings, face_encoding)
            name = "Unknown"

            # Find the best match
            if True in matches:
                matched_indexes = [i for (i, match) in enumerate(matches) if match]
                face_distances = face_recognition.face_distance(self.known_encodings, face_encoding)
                best_match_index = min(matched_indexes, key=lambda x: face_distances[x])
                name = self.known_names[best_match_index]

            # Display the name on the frame
            cv2.putText(frame, name, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)
        # Convert the frame to QImage and set it as the image for the QLabel
        image = QImage(frame, frame.shape[1], frame.shape[0], QImage.Format_RGB888)
        pixmap = QPixmap.fromImage(image)
        self.video_label.setPixmap(pixmap)

    def closeEvent(self, event):
        # Release the video capture object when the window is closed
        self.video_capture.release()
        event.accept()

if name == "main":
    # Create the application instance
    app = QApplication([])

    # Create the main window instance
    window = FacialRecognitionApp()

    # Show the main window
    window.show()

    # Run the application event loop
    app.exec_()
```

8.6 DataBase Codes

1) This code is explained in the database in header 5.18.2

```
import pyrebase
# Firebase project configuration
config = {
  "apiKey": "YOUR_API_KEY",
  "authDomain": "YOUR_AUTH_DOMAIN",
  "projectId": "YOUR_PROJECT_ID",
  "storageBucket": "YOUR_STORAGE_BUCKET",
  "messagingSenderId": "YOUR_MESSAGING_SENDER_ID",
  "appId": "YOUR_APP_ID",
  "databaseURL": "YOUR_DATABASE_URL"
}
# Initialize Pyrebase with Firebase project configuration
firebase = pyrebase.initialize_app(config)

 #Create an instance of the Firebase Realtime Database
db = firebase.database()
#Create an instance of the Firebase Realtime Database
db = firebase.database()
```

2) The code that is found in the database in header 5.18.3

```
#Read data from the Firebase Realtime Database
data = db.child("path/to/data").get()
#Access the retrieved data
retrieved_data = data.val()
#Order the data by a specific child
keyordered_data = db.child("path/to/data").order_by_child("child_key").get()
#Filter the data by a specific value
filtered_data = db.child("path/to/data").equal_to("value", "child_key").get()
#Example of advanced querying options
Query=db.child("path/to/data").start_at(10).end_at(100).limit_to_first(5).get()
```

3) The code that is found in the database in header 5.18.4

```
#Write data to the Firebase Realtime Database
db.child("path/to/data").set({"key": "value"})
#Append new data with a generated unique key
db.child("path/to/data").push({"key": "value"})
```

#Update existing data in the Firebase Realtime Database
db.child("path/to/data").update({"key": "new_value"})
#Remove data from the Firebase Realtime Database
db.child("path/to/data").remove()

4) This code is explained in the database in header 5.18.5

#Initialize Firebase app configuration
config = {
"apiKey": "YOUR_API_KEY",
"authDomain": "YOUR_AUTH_DOMAIN",
"databaseURL": "YOUR_DATABASE_URL",
"storageBucket": "YOUR_STORAGE_BUCKET"
}
# Initialize Pyrebase4
firebase = pyrebase.initialize_app(config)
Get the Firebase Realtime Database reference
db = firebase.database()
#Write data to the Firebase Realtime Database
data = {"name": "John", "age": 30}
db.child("users").child("1").set(data)
#Read data from the Firebase Realtime Database
user = db.child("users").child("1").get()
print(user.val()) # {'name': 'John', 'age': 30}
#Update data in the Firebase Realtime Database
db.child("users").child("1").update({"age": 31})
Delete data from the Firebase Realtime Database
db.child("users").child("1").remove()
#Listen for changes in data and receive updates in real-time
def on_data_change(event):
print(event["data"])
db.child("users").child("1").stream(on_data_change)

5) This code is explained in the database in header 5.18.7

#Sign up a new user
auth.create_user_with_email_and_password("example@example.com", "password")
#Sign in an existing user
auth.sign_in_with_email_and_password(“example@example.com”, “password”)
#Get the currently signed-in user

user = auth.get_account_info(token)['users'][0]
#Update user profile
auth.update_account_info(token,{"displayName":"John","photoURL":"https://example.com/profile.jpg"})

6) This code is explained in the database in header 5.18.8
Upload a file to Firebase Storage
storage.child("images/profile.jpg").put("profile.jpg")
Download a file from Firebase Storage
storage.child("images/profile.jpg").download("profile.jpg")
Get the URL of a file in Firebase Storage
url = storage.child("images/profile.jpg").get_url(None)

7) This code is explained in the database in header 5.18.9
#Listen for changes in authentication state
```
def on_auth_state_change(user):
    if user is None:
        print("User signed out")
    else:
        print("User signed in:", user["localId"])
auth.add_auth_state_listener(on_auth_state_change)
```

8) This code is explained in the database in header 5.18.10
#Custom query example: Retrieve users with age greater than 25, ordered by name
users=db.child("users").order_by_child("age").start_at(25).order_by_child("name").get()

9) This code is explained in the database in header 5.18.11
#Batch operation example: Update multiple data in a single batch
```
batch = db.batch()
batch.update(db.child("users").child("1"), {"name": "John"})
batch.update(db.child("users").child("2"), {"age": 32})
batch.commit()
```

8.7 Mobile Application

8.7.1 How to create homePage

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
// This widget is the root of your application.
@override Widget build(BuildContext context) {
return MaterialApp( title: 'Hello World DemApplication', theme: ThemeData(
primarySwatch: Colors.blue, ),
home: MyHomePage(title: 'Home page'), ); } }
class MyHomePage extends StatelessWidget { MyHomePage({Key key,
this.title}) : super(key: key); final String title;
@override Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar( title: Text(this.title), )
, body: Center( child: Text( 'Hello World', ) ), );
}
}
```

8.7.2 Text

```
Text('Hello World!', style: TextStyle(fontWeight: FontWeight.bold))
```

8.7.3 Image

1. Create a folder, assets in the project folder and place the
necessary images.
2. Specify the assets in the pubspec.yaml as shown below:

```
flutter:
assets:
- assets/smiley.png
```

3.Now, load and display the image in the application.

```
Image.asset('assets/smiley.png')
```

8.7.4 Icon

```
Icon(Icons.email)
```

8.7.5 tingNavigation and Rou

MaterialPageRoute

```
MaterialPageRoute(builder: (context) => Widget())
```

Navigation.push

```
Navigator.push( context, MaterialPageRoute(builder: (context) => Widget()), );
```

Navigation.pop
Navigator.push(context);

8.7.6 CurvedAnimation

controller = AnimationController(duration: const Duration(seconds: 2), vsync: this);
animation = CurvedAnimation(parent: controller, curve: Curves.easeIn)

8.7.7 Button

body: Center(
child: GestureDetector
(onTap: () { _showDialog(context); },
child: Text('Hello World',))
Box by inheritingCreate the widget, Rating
StatefulWidget
class RatingBox extends StatefulWidget { }

8.7.8 Create a state for RatingBox, _RatingBoxState byinheriting Stat <T>

class _RatingBoxState extends State { }