

IoT & MACHINE LEARNING INTEGRATED VEHICLE NUMBER PLATE RECOGNISITAION SYSTEM FOR AUTHORISED ACCESS

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

by

B. SARYU	21NU1A0409
D. JASMINE	21NU1A0431
B. CHARMILA	21NU1A0412
K. JYOTHI SWAROOP	21NU1A0448
N. RAJEEV	21NU1A0459

Under the Guidance of

Dr. K. Rajasekhar, M. Tech , Ph.D.

Associate Professor



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NADIMPALLI SATYANARAYANA RAJU INSTITUTE OF TECHNOLOGY**

(Autonomous)

SONTYAM, VISAKHAPATNAM-531173

2024-2025

NADIMPALLI SATYANARAYANA RAJU INSTITUTE OF TECHNOLOGY
(Autonomous)
SONTYAM, VISAKHAPATNAM-531173
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



CERTIFICATE

This is to certify that the Project Work entitled “**IoT & MACHINE LEARNING INTEGRATED VEHICLE NUMBER PLATE RECOGNITION SYSTEM FOR AUTHORISED ACCESS**” that is being submitted by **B. SARYU (21NU1A0409), D. JASMINE (21NU1A0431), B. CHARMILA (21NU1A0412), K. JYOTHI SWAROOP (21NU1A0448), N. RAJEEV (21NU1A0459)** for the fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY in ELECTRONICS AND COMMUNICATION ENGINEERING** to Jawaharlal Nehru Technological University – Gurajada, Vizianagaram is a record of Bonafide work carried out by them under my guidance and supervision.

Project Guide

Dr. K. Rajasekhar

Associate Professor

Head of the Department

Dr. B. Siva Prasad

Professor

INTERNAL EXAMINER

EXTERNAL EXAMINER

EXTERNAL EXAM DATE:_____

DECLARATION

We declare that this project entitled “**IoT & MACHINE LEARNING INTEGRATED VEHICLE NUMBER PLATE RECOGNITION SYSTEM FOR AUTHORISED ACCESS**” has been carried out by us and the contents have been presented in the form of a dissertation in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **ELECTRONICS AND COMMUNICATION ENGINEERING**.

We further declared that this dissertation has not been submitted elsewhere for any degree.

21NU1A0409	B. SARYU
21NU1A0431	D. JASMINE
21NU1A0412	B. CHARMILA
21NU1A0448	K. JYOTHI SWAROOP
21NU1A0459	N. RAJEEV

ACKNOWLEDGEMENT

“Gratitude is the memory of the heart” goes saying. Expressing heart full thanks is always a pleasant duty. It will be a pleasure for us expressing my deep sense of gratitude and sincere thanks to our guide **Dr. K. Rajasekhar**, Associate Professor. Especially the extensive comments, the discussions, and interactions with him had a direct impact on the final form and quality of this thesis.

We are thankful to our beloved Project Coordinator **Dr. K. Rajasekhar**, Associate Professor, for providing facilities and continuous support to complete this project.

We are thankful to Head of the Department **Dr. B. Siva Prasad**, Professor of **N S Raju Institute of Technology**, for his excellent guidance and enthusiastic encouragement in motivating us to take up this challenging task.

We are thankful to our Chief Management officer **Dr. P. S. Raju**, for providing facilities and continuous support to complete this project.

We thank all the Teaching and Non-Teaching Staff for their encouragement and support in the successful completion of this project.

We are thankful to our Principal **Dr. S. Sambhu Prasad** for providing facilities and continuous support to complete this project.

We thank the **Management** of **N S Raju Institute of Technology** for providing the various resources to complete this project.

21NU1A0409	B. SARYU
21NU1A0431	D. JASMINE
21NU1A0412	B. CHARMILA
21NU1A0448	K. JYOTHI SWAROOP
21NU1A0459	N. RAJEEV

ABSTRACT

Vehicle number plate recognition is a crucial component in modern traffic management and security systems. It enables automation in toll collection, access control, and monitoring of traffic violations. In this project, we develop a Raspberry Pi-based Vehicle Number Plate Recognition System that uses image processing and OCR to detect and read vehicle registration numbers automatically. The system hardware includes a Raspberry Pi 3, camera module, RFID reader, and an LCD touchscreen display. The RFID module first detects the presence of a vehicle, triggering the camera to capture live images. The captured image is processed using OpenCV for tasks like grayscale conversion, edge detection, and contour finding to isolate the number plate region.

Once the number plate area is located, it is cropped and enhanced using filtering techniques to reduce noise and improve readability. The processed image is then fed to an OCR engine (Tesseract) which extracts the alphanumeric characters from the plate. The recognized number is displayed on the LCD and can also be logged for future reference or monitoring. The system is designed to be compact, low-cost, and efficient for real-time applications such as gated community access, parking management, and automated surveillance systems. Future improvements may include integration with cloud storage, real-time alert systems, and AI-based character recognition for improved accuracy in diverse conditions.

KEYWORDS: IoT, Machine Learning, Number Plate Recognition, Access Control, Security Systems, Automated Gates.

LIST OF FIGURES

FIGURE NO	FIGURE TITLE	PAGE NO
3.1	Raspberry pi 3 processor	18
3.2	Components of the Raspberry pi 3 processor	19
3.3	Pin description of raspberry pi	20
3.4	GPIO pins of raspberry pi	21
3.5	Physical layout of GPIO pins	22
3.6	Pin description of GPIO pins	22
3.7	Micro SD card socket	23
3.8	Micro SD card	23
3.9	Regulated power supply	24
3.10	Circuit diagram of regulated power supply with led connection	25
3.11	Step-down transformer	26
3.12	Hi-Watt 9V battery	27
3.13	Bridge rectifier: a full-wave rectifier using 4 diodes	28
3.14	DB107	29
3.15	Voltage regulator	29
3.16	Inside a LED	30
3.17	Parts of a LED	30
3.18	USB power cable	30
3.19	LCD Display	32
3.20	USB camera	33
4.1	Circuit Diagram of the system integrated with Raspberry pi 3	35
4.2	Assembled picture	36
5.1	Raspberry Pi Monitor	40
5.2	Raspberry Pi Compilation	41
5.3	Raspberry Pi Configuration	42
5.4	Raspberry Pi setting	42
5.5	Raspberry Pi Configuration	43

FIGURE NO	FIGURE TITLE	PAGE NO
5.6	Configuring Locales	43
5.7	Raspberry Pi Tools	44
5.8	Raspberry Pi Packages	44
5.9	Raspberry Pi Software Tools	45
5.10	Raspberry Pi Configuration Tools	46
5.11	Raspberry pi Bash Script	47
5.12	Program in Terminal	48
5.13	Printing output in the Terminal	49
5.14	Code Output	50
5.15	Python Shell Window	51
5.16	Program in Python	52
5.17	Output of Python Code	52

LIST OF TABLES

TABLE NO	TABLE TITLE	PAGE NO
3.1	Components Required for Vehicle Number Recognition System	34
5.1	Data Types	53

CONTENTS

ABSTRACT	I
LIST OF FIGURES	II
LIST OF TABLES	IV
	1
LIST OF PO 's and PSO's	
S. No.	TITLE
CHAPTER-1	INTRODUCTION
	2 - 7
1.1	Introduction To IoT
1.2	Introduction To Project
1.3	Problem Statement
1.4	Aim and Objectives
	5-7
CHAPTER-2	LITERATURE REVIEW
	8 - 17
CHAPTER-3	HARDWARE DESCRIPTION
	18 - 34
3.1	Raspberry Pi Processor Overview
3.2	Pin Description of Raspberry Pi
3.2.1	Features
3.2.2	Connectors
3.2.3	General Purpose I/O Pins
3.2.4	Key Benefits of Raspberry Pi
3.2.5	Applications of Raspberry Pi
3.3	Hard Disk (SD Card)
3.4	Regulated Power Supply
3.4.1	Introduction
3.4.2	Block Diagram

S. No.	TITLE	PageNo
3.4.4	Battery Power Supply	26
3.4.5	Rectifiers	27
3.4.6	Bridge Full Wave Rectifiers	27
3.4.7	DB107	28
3.4.8	Filters	29
3.4.9	Voltage Regulator	29
3.4.10	LED	30
3.5	USB Power Cable	30
3.6	LCD Monitor	32
3.7	USB Camera	33
3.8	Miscellaneous	34
CHAPTER-4	HARDWARE IMPLEMENTATION	35 – 38
4.1	Block Diagram	35
4.2	Assembled Pictures	36
4.3	Working of the Project	38
CHAPTER-5	SOFTWARE IMPLEMENTATION	39 - 53
5.1	Description	39
5.2	Python	39
5.3	Configuration RPI and Compilation Steps	40
5.3.1	Run the Config Tool	41
5.3.2	Using Python 3 in Raspbian OS	46
5.4	Getting Started with the Interpreter	48
5.4.1	Running the Python Program from a File	49
5.4.2	Development Environments	50
5.4.3	IDLE	51
CHAPTER-6	EXPERIMENTAL RESULTS	55
CHAPTER-7	CONCLUSION & FUTURE SCOPE	56 – 57
7.1	Conclusion	56
7.2	Future Scope	57
	REFERENCES	58 – 60
	APPENDIX	61 – 67

List of Program Outcomes

As Per the Program of Study

- PO1:** Apply the knowledge of basic sciences and fundamental engineering concepts in solving engineering problems (**Engineering Knowledge**)
- PO2:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. (**Problem Analysis**)
- PO3:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. (**Design/Development of Solutions**)
- PO4:** Perform investigations, design and conduct experiments, analyze and interpret the results to provide valid conclusions (**Investigation of Complex Problems**)
- PO5:** Select/develop and apply appropriate techniques and IT tools for the design & analysis of the systems (**Modern Tool Usage**)
- PO6:** Give reasoning and assess societal, health, legal and cultural issues with competency in professional engineering practices (**The Engineer and Society**)
- PO7:** Demonstrate professional skills and contextual reasoning to assess environmental/societal issues for sustainable development (**The Environment and Sustainability**)
- PO8:** Demonstrate Knowledge of professional and ethical practices (**Ethics**)
- PO9:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary situations (**Individual and Team Work**)
- PO10:** Communicate effectively among engineering community, being able to comprehend and write effectively reports, presentation and give / receive clear instructions (**Communication**)
- PO11:** Demonstrate and apply engineering & management principles in their own / team projects in multidisciplinary environment (**Project Finance and Management**)
- PO12:** Recognize the need for, and have the ability to engage in independent and lifelong learning (**Life Long Learning**)
- PSO1:** To demonstrate the ability to design and develop complex systems in the areas of next generation Communication Systems, IoT based Embedded Systems, Advanced Signal and Image Processing, latest Semiconductor technologies, RF and Power Systems.)
- PSO2:** To demonstrate the ability to solve complex Electronics and Communication Engineering problems using the latest hardware and software tools along with analytical skills to contribute to useful, frugal and eco-friendly solutions.

CHAPTER 1

INTRODUCTION

CHAPTER-1

INTRODUCTION

1.1 INTRODUCTION:

The vehicle license plate is a unique identifier that distinguishes each vehicle on the road. Recognizing this identifier plays a crucial role in numerous real-world applications such as traffic flow monitoring, toll booth automation, parking lot access control, urban surveillance, and enforcement of traffic regulations. As urban vehicle density continues to grow, the need for an efficient, automated solution to identify and monitor vehicles has become increasingly significant. To address this demand, this project introduces an Automatic Vehicle License Plate Recognition (AVLPR) System built around the Raspberry Pi 3 platform, integrated with OpenCV for image processing and Optical Character Recognition (OCR) via the Tesseract engine, enabling seamless vehicle identification and monitoring in real time.

In the proposed system, a high-resolution camera module is interfaced with the Raspberry Pi to capture images of approaching or stationary vehicles. Upon detection of a vehicle—triggered by an ultrasonic sensor—the system initiates a multistage image processing workflow. This process begins with the conversion of the captured image into grayscale, followed by noise filtering using median or Gaussian techniques to enhance image clarity. Plate localization algorithms are then applied to isolate the license plate region, which is crucial for accurate extraction. Once the plate is localized, the system proceeds with character segmentation, isolating each alphanumeric character from the plate. Finally, the Tesseract OCR engine analyzes the segmented characters and recognizes the license number with high accuracy.

The recognized license plate number is not only displayed on a touch screen interface for immediate feedback but can also be stored locally or transmitted to a remote server for centralized logging, access control, or law enforcement purposes. An audible buzzer is included in the design to provide real-time alerts for successful recognition or specific system states (such as detection failure or unauthorized vehicle access).

This embedded solution leverages both hardware and software optimization to deliver a low-cost, compact, and scalable AVLPR system. The Raspberry Pi, being power-efficient and highly customizable, serves as the ideal platform for deployment in outdoor environments and constrained urban spaces. Additionally, the system is designed to adapt

to varying lighting and weather conditions, thanks to image enhancement techniques and tunable camera exposure settings. It also adheres to Indian vehicle number plate norms, which include variations in font style, color contrast (black-on-white for private vehicles and black-on-yellow for commercial), and character count.

In summary, the proposed project not only automates the identification of vehicle license plates but also establishes a foundation for smart traffic infrastructure. Its potential applications span across smart cities, automated gates, restricted zone monitoring, and law enforcement surveillance systems, thus contributing significantly to transportation management and public safety.

1.2 INTRODUCTION TO IoT:

The Internet of Things (IoT) is a transformative paradigm wherein everyday physical objects are embedded with sensors, actuators, software, and connectivity features that enable them to collect, transmit, and exchange data over the internet. This digital-physical integration fosters smart environments that respond dynamically to changes, significantly enhancing automation, real-time decision-making, and operational efficiency. IoT has found widespread adoption across multiple domains, including healthcare (remote patient monitoring), agriculture (precision farming), industry (smart manufacturing), urban planning (smart cities), and particularly in transportation, where it is revolutionizing traditional infrastructure.

In the domain of intelligent transportation systems (ITS), IoT technologies serve as the backbone for enabling real-time traffic management, vehicular communication, incident detection, and smart mobility services. One of the prominent applications of IoT in this space is Automatic Vehicle License Plate Recognition (AVLPR). Here, cameras, sensors, and embedded processors are employed in synergy to detect and recognize vehicle registration plates for purposes such as traffic law enforcement, automated toll collection, parking access management, and border control. When integrated with cloud infrastructure, these systems can provide real-time analytics, maintain historical data, and even integrate with national databases for enhanced security and compliance.

The fusion of IoT with image processing and machine learning technologies significantly elevates the capability of AVLPR systems. Image processing techniques help in plate localization, noise reduction, and character segmentation, while machine learning models can enhance optical character recognition (OCR) accuracy under varying lighting, weather, and angle conditions. This hybrid approach enables systems to not only identify vehicles

accurately but also predict traffic patterns, detect anomalies, and issue automated alerts—making roadways safer and smarter.

At the core of this IoT-based AVLPR system is the Raspberry Pi, a compact, energy-efficient, and cost-effective single-board computer that provides just the right balance of computing power and modularity. It is equipped with General Purpose Input/Output (GPIO) pins, USB ports, a CSI camera interface, and networking capabilities (Wi-Fi and Ethernet) making it highly adaptable for edge processing. In such a setup, the Raspberry Pi handles real-time image acquisition, data pre-processing, and OCR computation locally, thereby reducing the need for constant cloud communication and ensuring low-latency responses in mission-critical applications like automatic gate access or emergency route clearance.

Furthermore, its compatibility with various Linux distributions, support for Python, C/C++, and other programming languages, and access to vast open-source libraries (e.g., OpenCV, Tesseract OCR, and TensorFlow Lite) make Raspberry Pi an attractive platform for rapid prototyping and deployment of IoT-based vision systems. When deployed at scale, such systems can form a distributed sensor network, feeding into centralized dashboards for city-level traffic analytics and infrastructure planning.

In conclusion, the synergy of IoT, embedded systems, and intelligent vision technologies is reshaping the way transportation systems operate. Projects like IoT-based vehicle license plate recognition exemplify how modern computing platforms like Raspberry Pi can be harnessed to build real-time, autonomous, and scalable solutions for the smart cities of tomorrow.

1.3 PROBLEM STATEMENT:

With the rapid proliferation of vehicles on urban and rural roadways, the task of effectively monitoring and managing vehicular activities has become increasingly complex for traffic control departments, parking authorities, and law enforcement agencies. Traditional methods involving manual inspection and documentation of vehicle credentials are not only inefficient and labor intensive but also susceptible to errors, delays, and inconsistencies, particularly during peak traffic hours or in high-risk zones. As a result, challenges such as traffic congestion, unauthorized parking, vehicle theft, toll evasion, and traffic law violations have escalated, putting significant pressure on existing infrastructure and personnel.

One of the major technological hurdles lies in the diversity of number plate formats, especially in countries like India, where vehicles exhibit varied fonts, sizes, and color

schemes such as white plates for private vehicles, yellow for commercial, and red or blue plates for specific government or diplomatic purposes. Compounding these variations are practical issues such as dirt accumulation, faded characters, non-standard plate shapes, and inconsistent character spacing, all of which undermine the accuracy of automated recognition systems.

Environmental conditions further add to the complexity. Factors such as low-light environments, nighttime glare from headlights, motion blur from moving vehicles, and shadows or reflections on glossy plate surfaces pose significant barriers to image clarity and reliable OCR (Optical Character Recognition) performance. In such dynamic scenarios, even state-of-the-art ANPR (Automatic Number Plate Recognition) systems often fail to deliver consistent and real-time results without heavy computational resources or high-end surveillance equipment.

Moreover, existing ANPR implementations are often cost-intensive, requiring dedicated servers, high-resolution IP cameras, and centralized data processing units. These systems are typically designed for static infrastructure such as toll booths, highways, or border crossings, making them unsuitable for decentralized, portable, or small-scale applications—such as in housing societies, private parking areas, or smaller municipalities.

Therefore, there is a pressing need for a scalable, affordable, and standalone vehicle license plate recognition system that can adapt to diverse number plate styles, perform robustly in real-world lighting conditions, and operate effectively with minimal infrastructure. Such a solution should also support local data processing, wireless communication, and flexible deployment, making it ideal for modern smart city requirements and next-generation traffic management systems. This project aims to address these challenges by developing a Raspberry Pi-based system integrated with image processing and IoT technologies to deliver a compact, real-time, and autonomous number plate recognition platform.

1.4 AIM AND OBJECTIVES:

Aim

The main goal of this project is to develop a real-time, IoT-based Vehicle License Plate Recognition (VLPR) System using the Raspberry Pi 3 platform. The idea is to create a smart, affordable, and reliable solution that can identify vehicles automatically, without needing constant human input. By combining hardware and software technologies, the system is

designed to work efficiently under real-world conditions and is tailored to meet the license plate standards used in India. This makes it suitable for a wide range of applications like smart parking, toll booths, secure entry gates, and traffic enforcement.

Objectives

- To capture clear and accurate images of number plates, a camera module is connected to the Raspberry Pi to take pictures of vehicles as they approach. This step is critical, as it forms the foundation for everything that follows. Whether the vehicles are moving or stationary, the system should capture high-quality images suitable for processing.
- To process the captured images for useful information, once the image is captured, it goes through several processing steps. First, it's converted to grayscale to simplify the data and reduce load. Then, we apply filtering techniques like median filtering to remove unwanted noise. These steps help in making the plate more visible and easier to analyze.
- To detect and isolate the number plate from the image, using contrast-based detection and morphological operations, the system identifies and extracts just the number plate section from the image. This is a critical part of the process and must work in various lighting conditions and even when the plates are a bit dirty or tilted.
- To break down the number plate into individual characters, once the plate is located, the characters are separated using scanning techniques that move horizontally and vertically. This is necessary because not all number plates have consistent spacing or alignment, especially in real-life scenarios.
- To recognize the individual characters using OCR, the characters on the plate are then interpreted using OCR (Optical Character Recognition). Tools like Tesseract OCR and template matching techniques help read and understand the alphanumeric values. These values can then be used for verification or stored in a database.
- To adapt the system for Indian number plate standards, since number plates in India follow specific color schemes and formats (white background for private vehicles and yellow for commercial ones), the system is fine-tuned to account for these variations. This makes it more accurate and relevant for local use.
- To enhance functionality with supporting hardware, additional components like ultrasonic sensors are used to detect when a vehicle is present. A buzzer is added to give audible alerts, and a touchscreen interface allows users to interact with the system or view results in real time. To ensure the system works well in all lighting conditions, using infrared projectors and smart exposure control, the system adjusts itself to work even at night or in harsh

lighting. It measures how light or dark the image is and fine-tunes the camera settings to get the best result.

- To test the system's performance in real-world conditions, the system will be tested in different environments day, night, sunny, cloudy, fast-moving traffic, and more—to make sure it performs consistently. We'll measure its speed, accuracy, and reliability to identify where improvements can be made.
- To make the system flexible and easy to integrate, the final goal is to build a solution that's easy to expand or plug into other systems. Whether it's a parking gate, a toll booth, or a police database, this system should be able to connect, send data, and receive commands with minimal hassle.

CHAPTER-2

LITERATURE REVIEW

CHAPTER-2

LITERATURE REVIEW

Automatic License Plate Recognition (ALPR) has emerged as a transformative technology at the intersection of computer vision, embedded systems, and automation, reshaping how we manage transportation, enhance security, and streamline urban operations. Its ability to automatically capture, process, and interpret vehicle license plates has made it indispensable in applications ranging from traffic surveillance to parking management, access control, and law enforcement. A pivotal enabler of ALPR's widespread adoption, particularly in resource-constrained environments, is the Raspberry Pi—a low-cost, single-board computer that combines accessibility with sufficient computational power to handle image processing tasks. By serving as the backbone for many ALPR systems, Raspberry Pi has democratized access to this technology, enabling its deployment in settings where expensive hardware would be prohibitive. This democratization is not merely a technical achievement but a societal one, as it extends the benefits of ALPR to developing regions, small businesses, and community-driven initiatives.

The Raspberry Pi's appeal lies in its versatility and affordability, often retailing for under \$50, yet offering the capability to run sophisticated algorithms, interface with cameras, and support real-time applications. In the context of ALPR, it acts as a central platform for capturing license plate images, processing them through a pipeline of detection, segmentation, and recognition, and verifying the extracted data against databases. For instance, a typical Raspberry Pi-based ALPR system might employ a connected camera to capture images, use image processing techniques to isolate the license plate, apply Optical Character Recognition (OCR) to extract alphanumeric characters, and cross-reference the results with a stored dataset to authenticate a vehicle. This workflow, while seemingly straightforward, involves intricate technical challenges, such as handling variable lighting, plate designs, and environmental conditions—all of which the Raspberry Pi manages with surprising efficacy for its size and cost.

One compelling example of Raspberry Pi's role in ALPR comes from the work of Sharma et al., who developed a system that achieved a remarkable 96% recognition accuracy. Their approach leveraged the Raspberry Pi's ability to run OCR algorithms efficiently, processing images captured by a standard camera. The system first detected the license plate region using edge detection and contour analysis, then segmented individual characters through thresholding

and morphological operations. Once segmented, the characters were fed into an OCR module trained to recognize specific fonts and layouts, enabling the system to extract plate numbers with high precision. The final step involved verifying the extracted data against a database, allowing the system to authenticate vehicles in real time. This implementation was particularly impactful in industrial settings, where rapid identification of unauthorized vehicles is critical for security. By automating gate access, the system not only enhanced safety but also alleviated the workload of security personnel, freeing them to focus on strategic tasks rather than routine checks.

The success of such systems underscores the Raspberry Pi's ability to balance performance and practicality. Unlike high-end servers or specialized hardware, which may offer superior processing power but come with prohibitive costs, the Raspberry Pi delivers a cost-effective alternative without sacrificing functionality. Its compact form factor makes it easy to integrate into diverse environments, from fixed installations at parking lots to mobile setups in traffic monitoring units. Moreover, its open-source ecosystem, supported by a vast community of developers, provides access to a wealth of libraries and tools—such as OpenCV for image processing or Tesseract for OCR—that streamline the development of ALPR systems. This accessibility empowers hobbyists, researchers, and small organizations to experiment with ALPR, fostering innovation and expanding its reach.

Beyond industrial security, Raspberry Pi-based ALPR systems have found applications in a variety of domains, each highlighting the technology's adaptability. In parking management, for example, ALPR enables automated systems that recognize license plates at entry and exit points, log timestamps, and calculate parking durations. This eliminates the need for physical tickets or manual interventions, creating a seamless experience for drivers and operators alike. In one implementation, a Raspberry Pi equipped with a high-resolution camera was deployed at a university parking lot, where it processed plates in real time and interfaced with a cloud-based billing system. The system not only reduced administrative overhead but also minimized disputes over parking fees, as all transactions were backed by timestamped records. Such applications demonstrate how ALPR can enhance efficiency in everyday scenarios, transforming mundane tasks into automated workflows.

Traffic surveillance is another area where Raspberry Pi-based ALPR shines, particularly in resource-limited settings. By deploying these systems along roads or highways, authorities can monitor vehicle activity, detect violations such as speeding or driving without registration, and issue alerts or citations automatically. The real-time nature of these systems is critical, as it allows for immediate responses to infractions, thereby improving road safety. For instance, a

rural municipality might use a Raspberry Pi-based ALPR unit to monitor a high-traffic intersection, capturing plates and cross-referencing them with a regional database to identify unregistered vehicles. The low cost of such setups makes them viable for areas where funding for advanced surveillance systems is scarce, ensuring that even underserved communities can benefit from modern enforcement tools.

Despite its strengths, the Raspberry Pi is not without limitations, particularly when it comes to compute-intensive tasks like deep learning. Modern ALPR systems increasingly rely on neural networks, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), to achieve state-of-the-art performance in plate detection and character recognition. These models, while highly accurate, demand significant computational resources, which can strain the Raspberry Pi's modest hardware. To address this, researchers have pursued several strategies. One approach involves optimizing algorithms to run efficiently on low-power devices, using lightweight models like MobileNet or Tiny YOLO that are designed for resource-constrained environments. These models sacrifice some accuracy for speed and efficiency, making them suitable for real-time applications on Raspberry Pi. Another strategy is to offload complex computations to cloud servers, where the Raspberry Pi handles image capture and preprocessing, while a remote server performs tasks like neural network inference. This hybrid approach leverages the strengths of both edge and cloud computing, ensuring robust performance without overburdening the local device.

The interplay between edge and cloud computing is a defining feature of modern ALPR systems, reflecting broader trends in the Internet of Things (IoT) and distributed processing. By performing initial processing on-device, Raspberry Pi-based systems reduce latency and bandwidth requirements, making them ideal for scenarios where connectivity is unreliable. For example, a remote toll booth might use a Raspberry Pi to capture and preprocess license plate images, sending only the extracted data to a central server for verification. This minimizes data transfer costs and ensures functionality even during network outages. Conversely, cloud integration enables scalability, allowing systems to handle large volumes of data or integrate with national databases. The flexibility to operate in either mode or a combination of both makes Raspberry Pi a powerful platform for ALPR, capable of adapting to diverse operational needs.

The technical evolution of ALPR extends far beyond hardware, encompassing a rich array of algorithms and methodologies that have driven its progress. Early ALPR systems relied heavily on traditional image processing techniques, such as edge detection, thresholding, and template matching. These methods, while effective for standardized plates under controlled

conditions, struggled with real-world variability, such as non-standard fonts, reflective surfaces, or partial occlusions. To overcome these limitations, researchers have increasingly turned to machine learning and deep learning, which offer greater robustness and adaptability. For instance, the work of Qadri and Asif demonstrated the efficacy of combining Canny edge detection with Hough transformation and template matching for Islamabad plates. Their system, tested on 102 samples under varying lighting conditions, achieved an 89.7% recognition rate, highlighting the reliability of edge-based methods for well-defined plate formats. However, its performance dipped for non-standard plates, underscoring the need for more flexible approaches.

In contrast, Siddique et al. explored a dynamic scenario, using the Harris corner detection algorithm to capture plates in motion. Their segmentation method, which integrated connected component analysis with pixel-based metrics, excelled in highway surveillance, achieving a 96.92% accuracy. This success illustrates the importance of tailoring algorithms to specific use cases, as motion introduces challenges like blur and perspective distortion that static systems rarely encounter. By leveraging robust segmentation, their system could isolate characters even in noisy images, ensuring reliable recognition under adverse conditions. Such advancements reflect the iterative nature of ALPR research, where each study builds on previous insights to address new challenges.

Neural network-based approaches have further elevated ALPR's capabilities, offering unparalleled accuracy and versatility. Yang et al.'s work combined weighted statistics with a momentum backpropagation neural network, prioritizing prominent features to enhance both speed and accuracy. This method proved particularly effective for real-time applications, such as toll booths, where rapid processing is essential. Similarly, Li et al. advanced the field with a ConvNet-RNN model, achieving a per-character accuracy of 95.1%. By training on diverse datasets, their system could handle complex plate designs, including those with multiple rows or decorative elements, outperforming traditional methods that relied on rigid assumptions about plate structure. These examples highlight the transformative impact of deep learning, which enables ALPR systems to generalize across diverse environments and plate types.

The Sighthound system represents a high-water mark in CNN-based ALPR, leveraging deep learning to achieve near-perfect performance across a range of conditions. Fine-tuned for various plate formats and environmental challenges, it surpassed commercial benchmarks, demonstrating the potential of CNNs to adapt to real-world complexity. However, its reliance on high computational resources poses a barrier to widespread adoption, particularly in low-resource settings. This is where Raspberry Pi-based systems offer a compelling alternative, as

evidenced by Sharma et al.'s OCR-based implementation, which rivaled more expensive solutions with its 96% accuracy. By optimizing algorithms for efficiency, these systems deliver comparable performance at a fraction of the cost, making ALPR accessible to a broader audience.

Other studies have explored innovative combinations of techniques to enhance ALPR's robustness. Duan et al. introduced a color edge detection approach with fuzzy maps, achieving a 93.7% success rate across a large dataset of 1,601 images. Their pipeline integrated binarization, connected component analysis, and OCR with topological sorting, showcasing the value of combining multiple methods to address different aspects of the recognition problem. Similarly, Anagnostopoulos et al. employed Sauvola's method and Sliding Concentric Windows for segmentation, paired with a Probabilistic Neural Network for recognition. Their system achieved a 96.5% segmentation rate but a lower overall success rate of 86%, illustrating the trade-offs between segmentation accuracy and end-to-end performance. These findings emphasize the importance of holistic design, where each stage of the ALPR pipeline—detection, segmentation, and recognition—must be carefully optimized to achieve reliable results.

The work of Kumar et al. further illustrates the synergy of traditional and modern approaches, using artificial neural networks with Canny edge detection and blob coloring to recognize 95.36% of 259 plates. This high accuracy reflects the power of integrating image processing with machine learning, offering a balanced solution for practical deployment. By leveraging Canny edge detection to identify plate boundaries and blob coloring to isolate characters, their system simplified the segmentation process, allowing the neural network to focus on recognition. Such hybrid approaches are particularly well-suited for Raspberry Pi, as they minimize computational overhead while maintaining high performance.

The practical impact of ALPR extends far beyond technical innovation, touching numerous facets of daily life. In traffic surveillance, ALPR systems monitor highways and urban roads, detecting violations like speeding, red-light running, or driving unregistered vehicles. By automating enforcement, these systems improve compliance and reduce the need for manual patrols, allowing law enforcement to allocate resources more effectively. For example, a city might deploy ALPR cameras at key intersections, capturing plates and cross-referencing them with a database of known offenders. If a stolen vehicle is detected, the system can alert nearby officers, enabling a swift response. This real-time capability enhances public safety, deterring criminal activity and ensuring accountability on the roads.

Access control is another critical application, where ALPR ensures that only authorized

vehicles enter restricted areas, such as corporate campuses, government facilities, or residential complexes. In high-security environments, verifying vehicle identities is paramount, as unauthorized access could lead to breaches or attacks. A Raspberry Pi-based ALPR system, for instance, might be installed at the entrance to a military base, scanning plates and granting access only to pre-approved vehicles. The system's ability to operate autonomously reduces the risk of human error, ensuring consistent enforcement of security protocols. Moreover, its low cost makes it feasible to deploy multiple units across large facilities, creating a robust perimeter defense.

Parking management represents one of ALPR's most visible applications, streamlining operations in both commercial and public spaces. By recognizing plates at entry and exit points, ALPR systems calculate parking durations and issue bills automatically, eliminating the need for tickets or cash transactions. This not only enhances convenience for drivers but also reduces operational costs for parking operators. In a shopping mall, for example, a Raspberry Pi-based ALPR system could log entry times, calculate fees based on duration, and integrate with a mobile app for seamless payments. The system might also detect VIP or disabled parking permits, ensuring compliance with regulations and improving accessibility. Such applications highlight ALPR's ability to transform routine tasks into efficient, user-friendly processes.

Toll payment systems similarly benefit from ALPR, enabling electronic collection that reduces congestion and improves traffic flow. On highways, ALPR cameras capture plates and link them to user accounts, deducting fees without requiring vehicles to stop. This seamless experience is particularly valuable in high-traffic corridors, where delays at toll booths can cause significant bottlenecks. A Raspberry Pi-based toll system, for instance, could be deployed in rural areas, where traditional toll infrastructure is impractical. By leveraging solar power and cellular connectivity, the system could operate independently, bringing modern tolling to underserved regions. These examples illustrate ALPR's role in enhancing transportation efficiency, making travel smoother and more equitable.

ALPR also plays a vital role in theft prevention and vehicle document verification. By cross-referencing plates with databases of stolen or unregistered vehicles, law enforcement can identify suspicious activity in real time. For instance, a patrol car equipped with an ALPR camera might scan plates during routine operations, flagging any matches with a stolen vehicle list. This proactive approach increases the likelihood of recovering stolen property and apprehending offenders. Similarly, ALPR supports document verification by ensuring that vehicles comply with licensing and insurance requirements. During roadside checks, an ALPR system could instantly verify a vehicle's registration status, reducing the time officers spend on

manual lookups. These applications contribute to safer roads and stronger regulatory frameworks, reinforcing public trust in transportation systems.

Despite its advancements, ALPR faces several challenges that limit its universal adoption. Environmental variability such as poor lighting, fog, or rain can degrade image quality, affecting detection and recognition accuracy. For example, a plate obscured by mud or snow may be unreadable, leading to false negatives. To mitigate this, researchers have developed preprocessing techniques, such as histogram equalization and adaptive thresholding, that enhance image clarity under adverse conditions. Plate diversity across countries further complicates matters, as algorithms must adapt to varying fonts, colors, and layouts. A system trained on European plates, for instance, may struggle with Arabic or Chinese characters, necessitating region-specific models. This challenge is particularly pronounced for Raspberry Pi-based systems, where computational constraints limit the complexity of algorithms.

Privacy concerns represent a significant ethical hurdle, as ALPR systems often collect and store sensitive data about vehicle movements. Without proper safeguards, this data could be misused, leading to unwarranted surveillance or profiling. For example, a poorly secured ALPR database might be hacked, exposing drivers' travel patterns to malicious actors. To address this, organizations must implement robust encryption, anonymization, and access controls, ensuring that data is used only for legitimate purposes. Public transparency is equally important, as communities have a right to know how ALPR systems operate and what protections are in place. By balancing security with privacy, stakeholders can build trust and encourage broader acceptance of ALPR.

Algorithmic bias is another concern, particularly when models are trained on limited or unrepresentative datasets. If a system is primarily tested on plates from urban areas, it may underperform in rural settings, where plate designs or vehicle types differ. This could lead to inequitable outcomes, such as higher error rates for certain demographics. To mitigate bias, researchers are prioritizing diverse datasets that encompass a wide range of plate formats, lighting conditions, and geographic contexts. Community engagement is also critical, as it ensures that ALPR systems address the needs of all stakeholders, not just those in dominant regions. These efforts align with broader calls for fairness in AI, emphasizing inclusivity and accountability.

Regulatory compliance adds further complexity, as ALPR systems must adhere to local laws governing data collection, storage, and usage. In some countries, for instance, storing license plate data beyond a certain period may be prohibited, requiring systems to implement automatic deletion protocols. Failure to comply could result in legal penalties or public

backlash, undermining the technology's credibility. To navigate this landscape, developers must work closely with policymakers, ensuring that ALPR systems align with legal and ethical standards. This collaborative approach is essential for scaling ALPR globally, as it fosters harmonization across jurisdictions.

Looking to the future, ALPR is poised for significant advancements, driven by emerging technologies and innovative paradigms. Deep learning continues to dominate, with models like YOLO and Faster R-CNN offering real-time detection and recognition in unconstrained environments. These models excel at handling diverse plates, occlusions, and motion blur, making them ideal for dynamic scenarios like highway surveillance. Attention mechanisms, inspired by transformer architectures, are further improving character recognition by focusing on salient features, such as the shape of a digit or the color of a plate. This reduces errors for ambiguous characters, enhancing overall accuracy.

Edge computing is another transformative trend, enabling ALPR systems to process data locally rather than relying on cloud infrastructure. Raspberry Pi is at the forefront of this shift, supporting lightweight models that deliver real-time performance on-device. For example, a Raspberry Pi equipped with a quantized neural network could detect and recognize plates without internet access, making it suitable for remote areas. This decentralization reduces latency, lowers bandwidth costs, and enhances resilience, as systems can operate independently of network conditions. As edge devices become more powerful, we can expect ALPR to become even more ubiquitous, embedded in everything from traffic lights to drones.

Federated learning offers a promising avenue for addressing privacy concerns, allowing ALPR models to train across multiple institutions without sharing sensitive data. In a federated setup, each organization such as a police department or parking operator—trains a local model on its own dataset, sharing only model updates with a central server. This preserves data sovereignty while enabling collaborative improvement of ALPR algorithms. For instance, a federated ALPR system could learn to recognize plates from different countries without ever exchanging raw images, ensuring compliance with privacy laws. This approach is particularly relevant for Raspberry Pi-based systems, as it allows low-power devices to contribute to global model training without requiring extensive resources.

Multimodal ALPR is an exciting frontier, combining visual data with contextual information to enhance accuracy. By integrating vehicle type, GPS location, or time of day, these systems can make more informed decisions. For example, a multimodal ALPR system might prioritize certain plate formats based on the region it's operating in, reducing false positives. Similarly, it could use vehicle color or make to disambiguate similar plates,

improving reliability in crowded scenes. Raspberry Pi's ability to interface with sensors like GPS modules or accelerometers makes it well-suited for multimodal applications, enabling richer data collection at the edge.

Synthetic data generation, powered by Generative Adversarial Networks (GANs), is addressing data scarcity by creating realistic plate images for training. This is particularly valuable for underrepresented plate types, which are often missing from real-world datasets. By simulating diverse designs, lighting conditions, and occlusions, GANs can produce balanced datasets that mitigate bias and improve inclusivity. For instance, a GAN-trained ALPR system might perform equally well on rural and urban plates, ensuring equitable outcomes. Raspberry Pi-based systems can leverage synthetic data during development, allowing developers to test algorithms without collecting extensive real-world samples.

IoT integration is expanding ALPR's role in smart cities, where connected cameras, sensors, and databases enable real-time analytics. In a smart city ecosystem, ALPR could optimize traffic flow by identifying congested routes and suggesting alternatives to drivers. It could also support predictive policing by analyzing vehicle movement patterns to anticipate crime hotspots. Raspberry Pi's compatibility with IoT protocols, such as MQTT or Zigbee, makes it a natural fit for these applications, allowing it to communicate with other devices in a networked environment. By embedding ALPR into urban infrastructure, cities can become safer, more efficient, and more responsive to residents' needs.

Blockchain technology is being explored to secure ALPR data, ensuring tamper-proof records for applications like tolling or law enforcement. By storing plate data on a decentralized ledger, blockchain can prevent unauthorized modifications, enhancing trust in the system. For example, a blockchain-based ALPR system could record toll transactions immutably, allowing drivers to verify charges independently. While blockchain's computational demands pose challenges for Raspberry Pi, lightweight protocols like IOTA are making it feasible to integrate distributed ledgers into edge devices. This convergence of blockchain and ALPR could redefine how we manage transportation data, prioritizing security and transparency.

The societal implications of ALPR are profound, raising both opportunities and challenges. On one hand, ALPR enhances safety by deterring crime, enforcing traffic laws, and streamlining operations. On the other, it risks enabling mass surveillance if not governed responsibly. Constant monitoring of vehicle movements could erode privacy, particularly if data is shared without consent. To address this, policymakers must establish clear guidelines on data retention, access, and usage, ensuring that ALPR serves the public good without compromising rights. Techniques like data anonymization, where plate numbers are hashed

before storage, can further protect privacy while preserving functionality.

Equity is another priority, as ALPR must serve all communities fairly. Biases in training data or algorithm design could lead to disparities, such as higher error rates for minority groups or rural regions. To prevent this, developers must validate models across diverse scenarios, engaging with stakeholders to identify blind spots. Initiatives like open-source ALPR frameworks, built on platforms like Raspberry Pi, can foster inclusivity by inviting contributions from global developers. By prioritizing fairness, ALPR can become a tool for empowerment, not exclusion.

Accessibility remains a core strength of ALPR, particularly through low-cost solutions like Raspberry Pi. By enabling small businesses, local governments, and even individuals to deploy ALPR, these platforms ensure that the technology's benefits are widely shared. A community center, for instance, might use a Raspberry Pi-based ALPR system to manage visitor parking, enhancing security without breaking the budget. Similarly, a developing country could deploy ALPR at border checkpoints, improving trade efficiency with minimal investment. These examples highlight the transformative potential of accessible technology, which levels the playing field and fosters innovation.

In conclusion, Automatic License Plate Recognition represents a remarkable fusion of vision, computation, and application, with Raspberry Pi serving as a catalyst for its global adoption. From edge-based detection to neural network-driven recognition, ALPR has evolved into a versatile tool that enhances safety, efficiency, and connectivity. Its integration into parking, tolling, surveillance, and security systems demonstrates its practical value, while its affordability ensures that these benefits reach diverse communities. However, realizing ALPR's full potential requires addressing technical challenges like environmental variability, ethical concerns like privacy, and societal issues like equity. Through continued innovation spanning deep learning, edge computing, and IoT ALPR can become a cornerstone of intelligent transportation, delivering secure, inclusive, and sustainable solutions for an increasingly mobile world. By fostering collaboration among technologists, policymakers, and citizens, we can ensure that ALPR not only meets today's needs but also anticipates tomorrow's challenges, paving the way for a smarter, safer future.

CHAPTER 3

HARDWARE DESCRIPTION

CHAPTER-3

HARDWARE DESCRIPTION

3.1 RASPBERRY-PI PROCESSOR OVERVIEW

The Raspberry Pi 4 is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 4 brings you a more powerful processor, 10x faster than the generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.

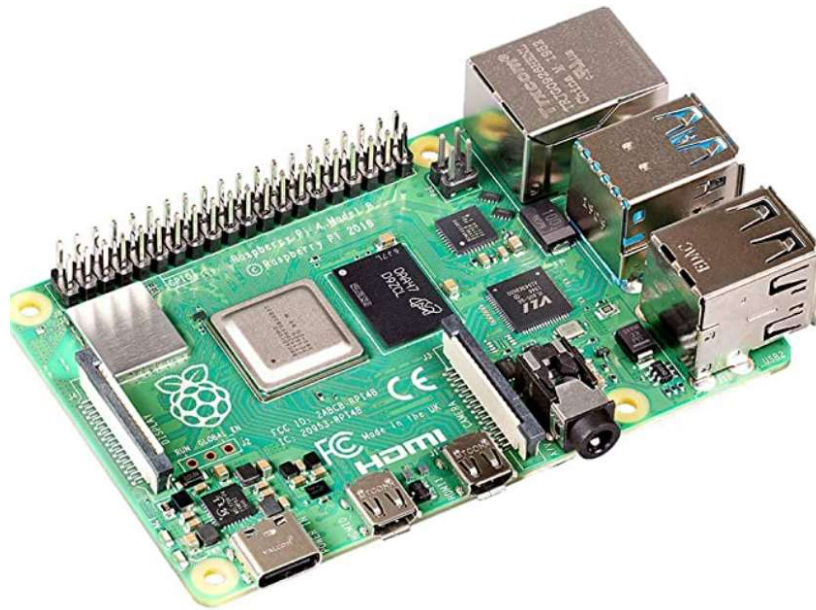


Fig 3.1: Raspberry-Pi3 Processor

The Raspberry Pi has made quite a splash since it was first announced. The credit-card sized computer is capable of many of the things that your desktop PC does, like spreadsheets, word-processing and games. It also plays high-definition video. It can run several flavors of Linux and is being used to teach kids all over the world how to program and it does all that for under \$50.

The Model B+'s FOUR built-in USB ports provide enough connectivity for a mouse, keyboard, or anything else that you feel the R-Pi needs, but if you want to add even more you can still use a USB hub. It is recommended that you use a powered hub so as not to overtax the on-board v

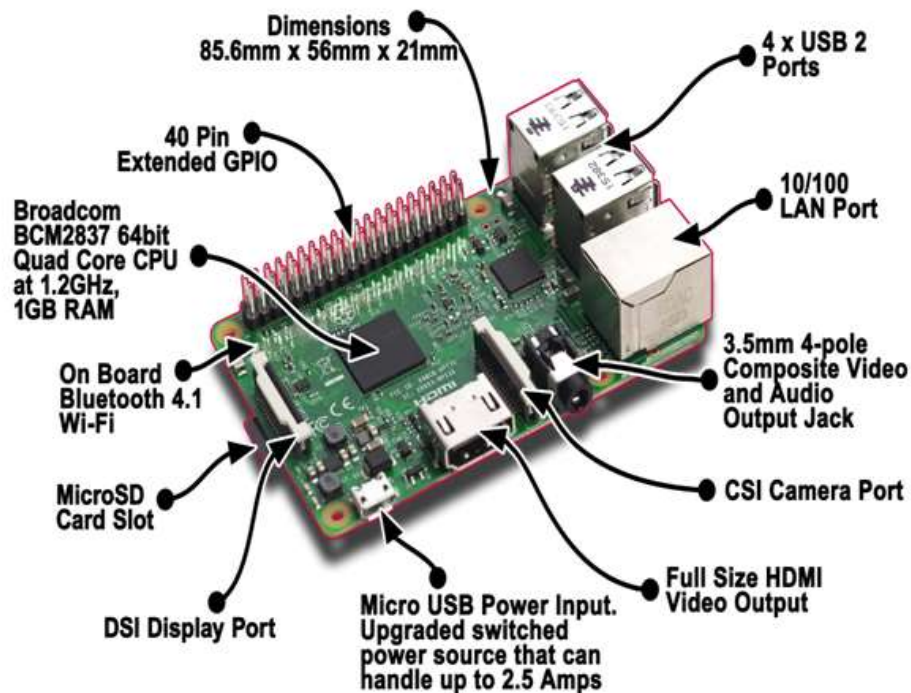


Fig 3.2: Componentss of the Raspberry-Pi Processor

The Model B+'s FOUR built-in USB ports provide enough connectivity for a mouse, keyboard, or anything else that you feel the R-Pi needs, but if you want to add even more you can still use a USB hub. It is recommended that you use a powered hub so as not to overtax the on-board voltage regulator. Powering the Raspberry Pi is easy, just plug any USB power supply into the micro-USB port. There's no power button so the Pi will begin to boot as soon as power is applied, to turn it off simply remove power. The four built-in USB ports can even output up to 1.2A enabling you to connect more USB devices (This does require a 2Amp micro USB Power Supply).

3.2 PIN DESCRIPTION OF RASPBERRY PI:

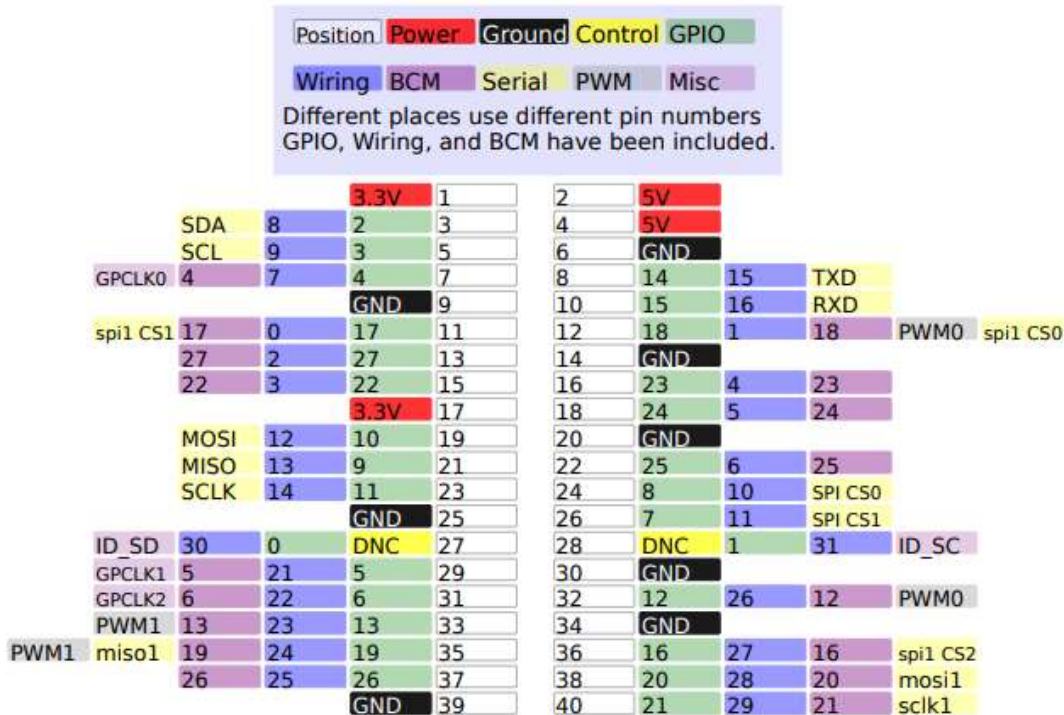


Fig 3.3: Pin Description of Raspberry Pi

3.2.1 FEATURES

- Processor- Broadcom BCM2837 chipset. 1.2GHz Quad-Core ARM Cortex-A53802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
- GPU- Dual Core Video Core IV® Multimedia Co-Processor. Provides Open GLES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
- Memory-1GB LPDDR2
- Operating System Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
- Dimensions- 85 x 56 x 17mm
- Power- Micro USB socket 5V1, 2.5A

3.2.2 CONNECTORS

Ethernet- 10/100 BaseT Ethernet socket

Video Output- HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC).

Audio Output- Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector.

GPIO Connector- 40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines.

Camera Connector- 15-pin MIPI Camera Serial Interface (CSI-2).

Display Connector- Display Serial Interface (DSI) flat flex cable connector with two data lanes and a clock lane.

Memory Card Slot- Push/pull Micro SDIO

3.2.3 GENERAL PURPOSE I/O (GPIO) PINS

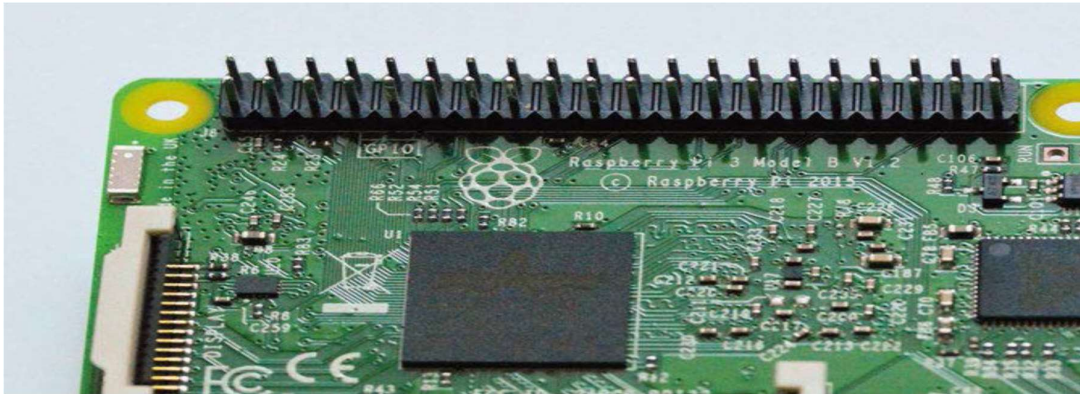


Fig 3.4: GPIO Pins of Raspberry-pi

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior including whether it is an input or output pin is controllable by the user at run time.

GPIO pins have no predefined purpose, and go unused by default. The idea is that sometimes a system integrator who is building a full system might need a handful of additional digital control lines and having these available from a chip avoids having to arrange additional circuitry to provide them.

GPIO capabilities may include

- GPIO pins can be configured to be input or output
- GPIO pins can be enabled/disabled
- Input values are readable (typically high or low)
- Output values are writable/readable

GPIO peripherals vary widely. In some cases, they are simple a group of pins that can switch as a group to either input or output. In others, each pin can be set up to accept or source different logic voltages, with configurable drive strengths and pull ups/downs. Input and output voltages are typically though not always limited to the supply voltage of the device with the GPIOs, and may be damaged by greater voltages.

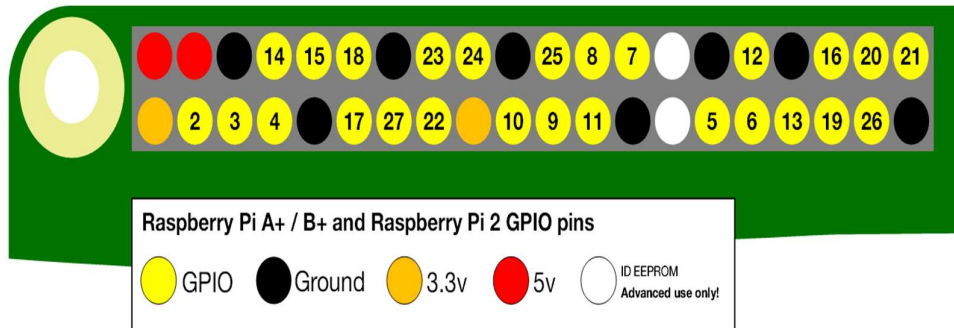


Fig 3.5: Physical layout of GPIO Pins

Raspberry Pi 3 Model B (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 TxD (UART) 15
	Ground	9		10	GPIO 16 RxD (UART) 16
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4 4
	3.3 VDC Power	17		18	GPIO 5 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI) 10
	Ground	25		26	GPIO 11 CE1 (SPI) 11
30	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM) 31
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0 26
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN 28
	Ground	39		40	GPIO 29 PCM_DOUT 29

Fig 3.6: Pin Description of GPIO pins

A GPIO pin's state may be exposed to the software developer through one of a number of different interfaces, such as a memory mapped peripheral, or through dedicated IO port instructions. Some GPIOs have 5 V tolerant inputs: even when the device has a low supply voltage (such as 2 V), the device can accept 5 V without damage.

3.2.4 KEY BENEFITS OF RASPBERRY PI 3

- Low cost
- Consistent board format
- 10x faster processing
- Added connectivity

3.2.5 APPLICATIONS OF RASPBERRY PI:

- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Environmental sensing/monitoring (e.g. weather station)
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming & Wireless access point

3.3 HARD DISK (SD CARD):

Here like the Model A & B, the B+ requires a FLASH card to store the operating system and files. The Model A and B had an SD socket on the bottom. The SD card stuck out the end of the Pi, and could come loose or snap off by accident with enough force. The good news for any users, we always had dual Micro/Standard SD cards so we can still use the same cards.

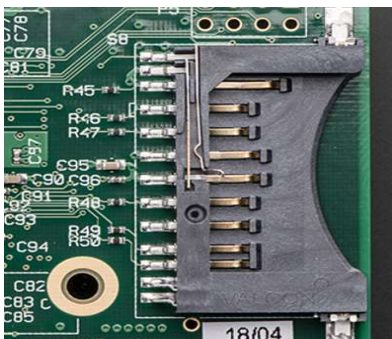


Fig 3.7: Micro SD Card Socket



Fig 3.8: Micro SD Card

The new raspberry pi B+ replaces the large SD socket with a new Micro SD socket, in that we just dump the coding in sdcard and these sdcard is put into the micro sdcard socket in raspberry-pi.

3.4 REGULATED POWER SUPPLY:

3.4.1 INTRODUCTION:

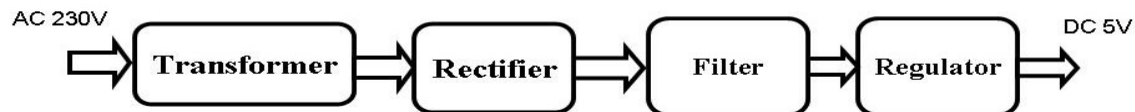
Power supply is a supply of electrical power. A device or system that supplies electrical or other types of energy to an output load or group of loads is called a power supply unit or PSU. The term is most commonly applied to electrical energy supplies, less often to mechanical ones, and rarely to others.

A power supply may include a power distribution system as well as primary or secondary sources of energy such as

- Conversion of one form of electrical power to another desired form and voltage, typically involving converting AC line voltage to a well-regulated lower-voltage DC for electronic devices. Low voltage, low power DC power supply units are commonly integrated with the devices they supply, such as computers and household electronics.
- Batteries.
- Chemical fuel cells and other forms of energy storage systems.
- Solar power.
- Generators or alternators.

3.4.2 BLOCK DIAGRAM:

Regulated Power supply



The basic circuit diagram of a regulated power supply (DC O/P) with led connected as load is shown in fig: 3.9.

Fig 3.9: Regulated Power Supply

REGULATED POWER SUPPLY

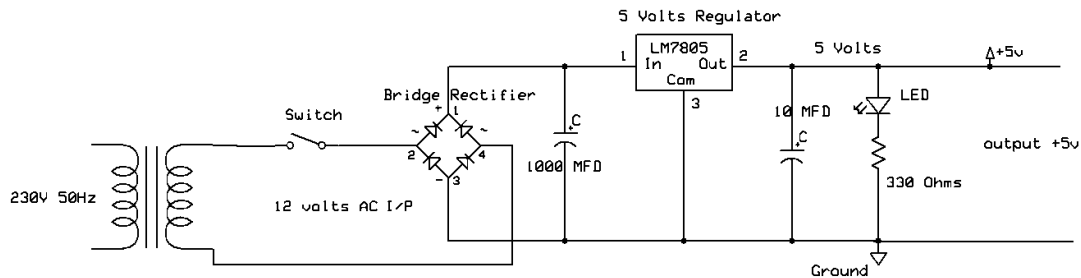


Fig 3.10: Circuit diagram of Regulated Power Supply with Led connection

The components mainly used in above figure are

- 230V AC MAINS
- TRANSFORMER
- BRIDGE RECTIFIER(DIODES)
- CAPACITOR
- VOLTAGE REGULATOR
- RESISTOR
- LIGHT EMITTING DIODE

The detailed explanation of each and every component mentioned above is as follows:

3.4.3 TRANSFORMER:

A transformer is a device that transfers electrical energy from one circuit to another through inductively coupled conductors without changing its frequency. A varying current in the first or primary winding creates a varying magnetic flux in the transformer's core, and thus a varying magnetic field through the secondary winding. This varying magnetic field induces a varying electromotive force (EMF) or "voltage" in the secondary winding. This effect is called mutual induction.

If a load is connected to the secondary, an electric current will flow in the secondary winding and electrical energy will be transferred from the primary circuit through the transformer to the load. This field is made up from lines of force and has the same shape as a bar magnet.

If the current is increased, the lines of force move outwards from the coil. If the current is reduced, the lines of force move inwards.

If another coil is placed adjacent to the first coil then, as the field moves out or in, the moving lines of force will "cut" the turns of the second coil. As it does this, a voltage is induced in the second coil. With the 50 Hz AC mains supply, this will happen 50 times a second. This is called MUTUAL INDUCTION and forms the basis of the transformer.

The input coil is called the PRIMARY WINDING; the output coil is the SECONDARY WINDING. Fig: 3.11 shows step-down transformer.

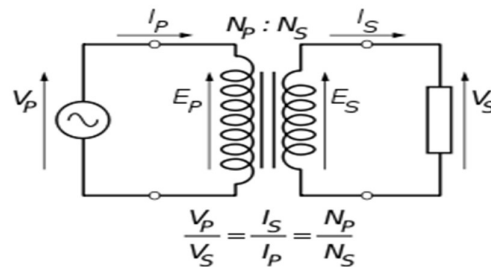


Fig 3.11: Step-Down Transformer

The voltage induced in the secondary is determined by the TURNS RATIO.

$$\frac{\text{primary voltage}}{\text{secondary voltage}} = \frac{\text{number of primary turns}}{\text{number of secondary turns}}$$

Step down transformer:

In case of stepdown transformer, Primary winding induces more flux than the secondary winding, and secondary winding is having less number of turns because of that it accepts less number of flux, and releases less amount of voltage.

3.4.4 BATTERY POWER SUPPLY:

A battery is a type of linear power supply that offers benefits that traditional line-operated power supplies lack: mobility, portability and reliability. A battery consists of multiple electrochemical cells connected to provide the voltage desired. Fig: 3.3.5 shows Hi-Watt 9V battery



Fig 3.12: Hi-Watt 9V Battery

The most commonly used dry-cell battery is the carbon-zinc dry cell battery. Dry-cell batteries are made by stacking a carbon plate, a layer of electrolyte paste, and a zinc plate alternately until the desired total voltage is achieved. The most common dry-cell batteries have one of the following voltages: 1.5, 3, 6, 9, 22.5, 45, and 90. During the discharge of a carbon-zinc battery, the zinc metal is converted to a zinc salt in the electrolyte, and magnesium dioxide is reduced at the carbon electrode. These actions establish a voltage of approximately 1.5 V.

3.4.5 RECTIFIERS:

A rectifier is an electrical device that converts alternating current (AC) to direct current (DC), a process known as rectification. Rectifiers have many uses including as components of power supplies and as detectors of radio signals. Rectifiers may be made of solid-state diodes, vacuum tube diodes, mercury arc valves, and other components.

A device that it can perform the opposite function (converting DC to AC) is known as an inverter.

When only one diode is used to rectify AC (by blocking the negative or positive portion of the waveform), the difference between the term diode and the term rectifier is merely one of usage, i.e., the term rectifier describes a diode that is being used to convert AC to DC. Almost all rectifiers comprise a number of diodes in a specific arrangement for more efficiently converting AC to DC than is possible with only one diode. Before the development of silicon semiconductor rectifiers, vacuum tube diodes and copper (I) oxide or selenium rectifier stacks were used.

3.4.6 BRIDGE FULL WAVE RECTIFIER:

The Bridge rectifier circuit is shown in fig: 3.13, which converts an ac voltage to dc voltage using both half cycles of the input ac voltage. The Bridge rectifier circuit is shown in the figure. The circuit has four diodes connected to form a bridge. The ac input voltage is applied to the

diagonally opposite ends of the bridge. The load resistance is connected between the other two ends of the bridge.

For the positive half cycle of the input ac voltage, diodes D1 and D3 conduct, whereas diodes D2 and D4 remain in the OFF state. The conducting diodes will be in series with the load resistance R_L and hence the load current flows through R_L .

For the negative half cycle of the input ac voltage, diodes D2 and D4 conduct whereas, D1 and D3 remain OFF. The conducting diodes D2 and D4 will be in series with the load resistance R_L and hence the current flows through R_L in the same direction as in the previous half cycle. a bi-directional wave is converted into a unidirectional wave.

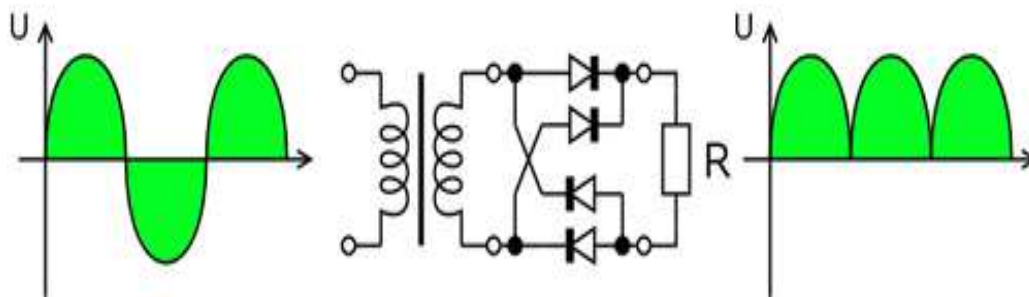


Fig 3.13: Bridge rectifier: a full-wave rectifier using 4 diodes

3.4.7 DB107:

Now -a -days Bridge rectifier is available in IC with a number of DB107. In our project we are using an IC in place of bridge rectifier. The picture of DB 107 is shown in fig: 3.3.8.

Features:

- Good for automation insertion
- Surge overload rating - 30 amperes peak
- Ideal for printed circuit board
- Reliable construction utilizing molded
- Glass passivated device
- Polarity symbols molded on body

- Mounting position: Any
- Weight: 1.0 gram



Fig 3.14: DB107

3.4.8 FILTERS:

Electronic filters are electronic circuits, which perform signal-processing functions, specifically to remove unwanted frequency components from the signal, to enhance wanted ones.

3.4.9 VOLTAGE REGULATOR:

A voltage regulator (also called a ‘regulator’) with only three terminals appears to be a simple device, but it is in fact a very complex integrated circuit. It converts a varying input voltage into a constant ‘regulated’ output voltage. Voltage Regulators are available in a variety of outputs like 5V, 6V, 9V, 12V and 15V. The LM78XX series of voltage regulators are designed for positive input. For applications requiring negative input, the LM79XX series is used. Using a pair of ‘voltage-divider’ resistors can increase the output voltage of a regulator circuit.

It is not possible to obtain a voltage lower than the stated rating. You cannot use a 12V regulator to make a 5V power supply. Voltage regulators are very robust. These can withstand over-current draw due to short circuits and also over-heating. In both cases, the regulator will cut off before any damage occurs. The only way to destroy a regulator is to apply reverse voltage to its input. Reverse polarity destroys the regulator almost instantly. Fig: 3.3.11 shows voltage regulator.

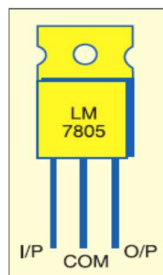


Fig 3.15: Voltage Regulator

3.4.10 LED:

A light-emitting diode (LED) is a semiconductor light source. LED's are used as indicator lamps in many devices, and are increasingly used for lighting. Introduced as a practical electronic component in 1962, early LED's emitted low-intensity red light, but modern versions are available across the visible, ultraviolet and infrared wavelengths, with very high brightness. The internal structure and parts of a led are shown in figures 3.16 and 3.17 respectively.

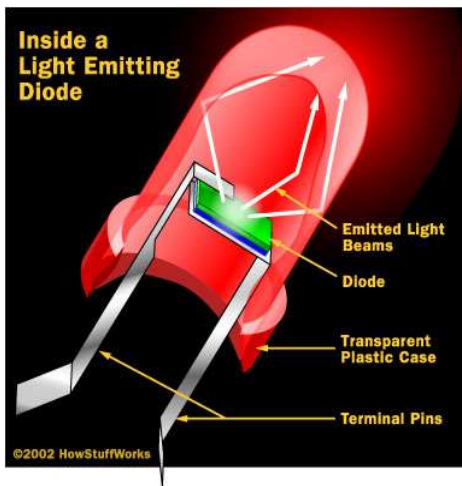


Fig 3.16: Inside a LED

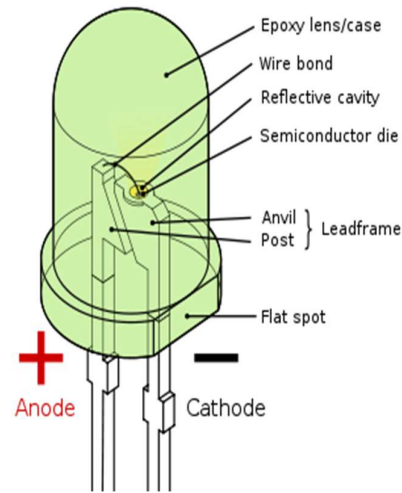


Fig 3.17: Parts of a LED

3.5 USB POWER CABLE:



Fig 3.18: USB Power Cable

USB data using to provide supply to raspberry pi in this project. USB supplies bus power across V_{BUS} and GND at a nominal voltage $5\text{ V} \pm 5\%$, at supply, to power USB devices. Power is sourced solely from upstream devices or hosts, and is consumed solely by downstream devices. USB provides for various voltage drops and losses in providing bus power. As such, the voltage at the hub port is specified in the range $5.00+0.25-0.60\text{ V}$ by USB 2.0, and $5.00+0.25-0.55\text{ V}$ by USB 3.0. It is specified that devices' configuration and low-power functions must operate down to 4.40 V at the hub port by USB 2.0 and that devices' configuration, low-power, and high-power functions must operate down to 4.00 V at the device port by USB 3.0.

There are limits on the power a device may draw, stated in terms of a *unit load*, which is 100 mA or 150 mA for SuperSpeed devices. There are low-power and high-power devices. Low-power devices may draw at most 1 unit load, and all devices must act as low-power devices when starting out as unconfigured. High-power devices draw at least 1 unit load and at most 5 unit loads (500 mA) or 6 unit loads (900 mA) for SuperSpeed devices. A high-powered device must be configured, and may only draw as much power as specified in its configuration. I.e., the maximum power may not be available.

A bus-powered hub is a high-power device providing low-power ports. It draws 1 unit load for the hub controller and 1 unit load for each of at most 4 ports. The hub may also have some non-removable functions in place of ports. A self-powered hub is a device that provides high-power ports. Optionally, the hub controller draw power for its operation as a low-power device, but all high-power ports draw from the hub's self-power.

Where devices (for example, high-speed disk drives) require more power than a high-power device can draw, they function erratically, if at all, from bus power of a single port. USB provides for these devices as being self-powered. However, such devices may come with a Y-shaped cable that has two USB plugs (one for power and data, the other for only power), so as to draw power as two devices. Such a cable is non-standard, with the USB compliance specification stating that "use of a 'Y' cable (a cable with two A-plugs) is prohibited on any USB peripheral", meaning that "if a USB peripheral requires more power than allowed by the USB specification to which it is designed, then it must be self-powered.

The standard connectors were deliberately intended to enforce the directed topology of a USB network: type-A receptacles on host devices that supply power and type-B receptacles on target devices that draw power. This prevents users from accidentally connecting two USB power supplies to each other, which could lead to short circuits and dangerously high currents, circuit failures, or even fire. USB does not support cyclic networks and the standard connectors from incompatible USB devices are themselves incompatible.

However, some of this directed topology is lost with the advent of multi-purpose USB connections (such as USB On-The-Go in smartphones, and USB-powered Wi-Fi routers), which require A-to-A, B-to-B, and sometimes Y/splitter cables. See the USB On-The-Go connectors section below for a more detailed summary description.

3.6 LCD MONITOR:

A computer monitor is an output device that displays information in pictorial form. A monitor usually comprises the visual display, circuitry, casing, and power supply. The display device in modern monitors is typically a thin film transistor liquid crystal display (TFT-LCD) with LED backlighting having replaced cold-cathode fluorescent lamp (CCFL) backlighting. Older monitors used a cathode ray tube (CRT). Monitors are connected to the computer via VGA, Digital Visual Interface (DVI), HDMI, DisplayPort, Thunderbolt, low-voltage differential signaling (LVDS) or other proprietary connectors and signals. Originally, computer monitors were used for data processing while television sets were used for entertainment. From the 1980s onwards, computers (and their monitors) have been used for both data processing and entertainment, while televisions have implemented some computer functionality. The common aspect ratio of televisions, and computer monitors, has changed from 4:3 to 16:10, to 16:9. Modern computer monitors are easily interchangeable with conventional television sets.



Fig 3.19: LCD Monitor

However, as computer monitors do not necessarily include integrated speakers, it may not be possible to use a computer monitor without external components

The estimation of the monitor size by the distance between opposite corners does not take into account the display aspect ratio, so that for example a 16:9 21-inch (53 cm) widescreen display has less area, than a 21-inch (53 cm) 4:3 screen. The 4:3 screen has dimensions of 16.8 in \times 12.6 in (43 cm \times 32 cm) and area 211 sq in (1,360 cm²), while the widescreen is 18.3 in \times 10.3 in (46 cm \times 26 cm)

The resolution for computer monitors has increased over time. From 320x200 during the early 1980s, to 1024x768 during the late 1990s. Since 2009, the most commonly sold resolution for computer monitors is 1920x1080.^[12] Before 2013 top-end consumer LCD monitors were limited to 2560x1600 at 30 in (76 cm), excluding Apple products and CRT monitors. Apple introduced 2880x1800 with Retina MacBook Pro at 15.4 in (39 cm) on June 12, 2012, and introduced a 5120x2880 Retina iMac at 27 in (69 cm) on October 16, 2014. By 2015 most major display manufacturers had released 3840x2160 resolution displays.

3.7 USB CAMERA:

Logitech® Webcam C170. The easy way to start video calling and send photos (5MP). With simple plug-and-play setup, you'll be making video calls in exceptional VGA resolution in no time on Logitech Vid™ HD. You can take and send beautiful, high-resolution photos at up to 5MP (software enhanced), too.



Fig 3.20: Logitech 5MegaPixel USB Camera

A built-in noise-reducing mike helps loved ones hear you clearly on calls. You can also record lively, colorful videos in XVGA (1024 x 768) resolution and share them with friends, family and the world. Also, the universal clip makes it easy to use with your desktop or laptop.

3.8 MISCELLANEOUS:

We have used the Raspberry Pi 3 board as the main processing unit, along with a module for capturing vehicle number plates for the Authorization access.

Table 3.1 Components required for Vehicle Number Plate Recognition

S.No	Parts	Type	Quantity
1	Raspberry Pi 3	Processor	1
2	USB	-	1
3	Regulated Power Supply	Input	-
4	Micro SD Card	Flash memory card used for storage.	1

CHAPTER 4

HARDWARE IMPLEMENTATION

CHAPTER-4

HARDWARE IMPLEMENTATION

4.1 BLOCK DIAGRAM

4.1.1 BLOCK DIAGRAM FOR THE SYSTEM INTEGRATED WITH RASPBERRY PI 4

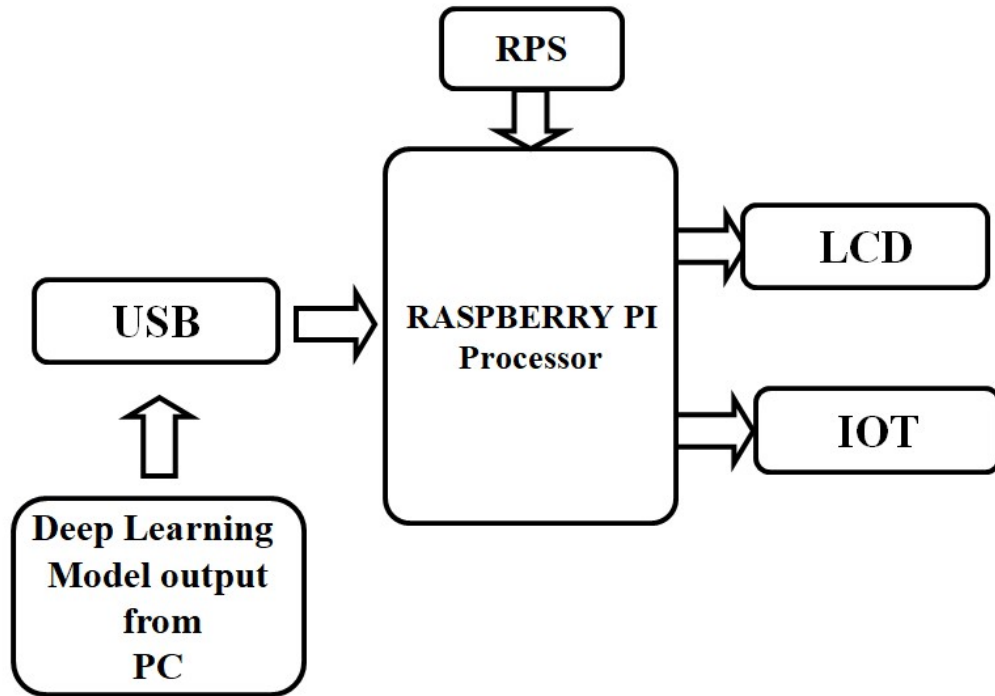


Fig 4.1 Circuit Diagram of the system integrated with Raspberry pi 4

The overall system architecture of the Skin Cancer Detection System using Raspberry Pi, highlighting the interaction between hardware components and the flow of data from input to output. At the heart of the system is the Raspberry Pi 4, which acts as the central processing unit. It is responsible for handling inputs, processing data, and coordinating communication with the display and IoT modules.

The system begins with the deep learning model, which is trained on a personal computer using large datasets of skin lesion images. Due to the computational requirements of training deep learning models, this step is performed externally. Once the model is trained, it is used to make predictions on new skin lesion images. The output of this model — the classification of the lesion as either benign or malignant is sent to the Raspberry Pi through USB serial

communication.

The Raspberry Pi receives this classification result via the USB port and then performs two critical operations. First, it displays the prediction on a 20x4 LCD screen that is directly connected to it. This provides the user with real-time feedback about the condition of the analyzed skin lesion. Second, the Raspberry Pi sends the same output to an IoT module, which further transmits the data to an Android web application and stores it in an Excel sheet for record-keeping and future analysis.

To power the entire setup, a Regulated Power Supply (RPS) is used. It ensures that the Raspberry Pi receives a stable voltage, thereby maintaining consistent and reliable operation of the system. The integration of the deep learning model with the Raspberry Pi and additional modules like LCD and IoT makes the system portable, low-cost, and efficient for use in remote or under-resourced areas where skin cancer diagnosis might otherwise be inaccessible.

4.2 ASSEMBLED PICTURE:

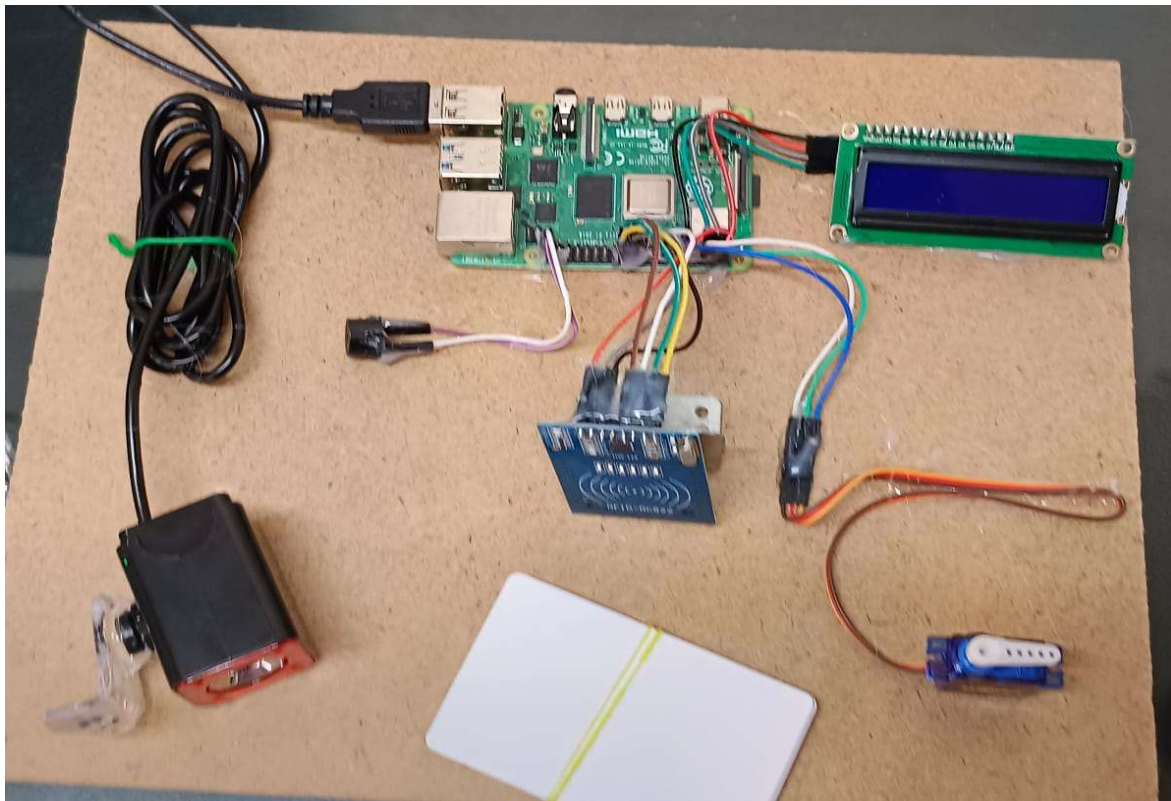


Fig 4.2: Assembled Picture

4.3 WORKING OF THE PROJECT:

The design and implementation of the skin cancer detection system are driven by the need for accessible and accurate diagnostic tools, especially in rural or remote areas where dermatological services are scarce. Leveraging deep learning and embedded system technologies, the project aims to create a cost-effective, portable solution capable of classifying skin lesions using image-based analysis. At the heart of this system lies the Raspberry Pi 4 Model B, which serves as the central processing hub, interfacing with camera modules, display screens, and IoT communication platforms to deliver an end-to-end skin lesion classification workflow.

Upon startup, the Raspberry Pi boots into the Raspbian OS and loads all essential libraries such as TensorFlow Lite or PyTorch Mobile for neural network inference, OpenCV and PIL for image handling, and Flask or Django for optional GUI/API integration. A camera module, either a dedicated Pi Cam v2 or a standard USB webcam, is connected directly to the Raspberry Pi to capture real-time skin images. For locations lacking stable power supply, the system can be powered by a battery pack or portable power bank, making it suitable for on-field screenings.

As soon as the system is powered on and initialized, the user can interact with it through a connected touchscreen display. The interface allows a healthcare worker or user to capture a clear image of the skin lesion in question. The captured image undergoes several preprocessing stages including resizing, contrast enhancement, noise reduction, and grayscale conversion all implemented using OpenCV and NumPy. These steps ensure that the image is optimized for accurate analysis by the deep learning model.

Once preprocessing is complete, the image is passed to the trained convolutional neural network (CNN), which has been converted to a lightweight model using TensorFlow Lite or PyTorch Mobile. This deep learning model has been trained on a comprehensive dataset of dermatological images and is capable of classifying different types of skin cancer, including melanoma, basal cell carcinoma, and benign moles. The model processes the image and outputs a prediction along with a confidence score indicating the likelihood of malignancy.

The prediction result is then displayed on the touchscreen, allowing the user to take note of the result and, if necessary, seek medical follow-up. For remote diagnostics and continuous monitoring, the system can be integrated with IoT platforms such as ThingSpeak, Firebase, or MQTT over Wi-Fi. This functionality enables secure transmission of prediction data, time-stamped records, and even environmental sensor inputs like UV exposure levels, directly to healthcare providers or cloud-based databases for long-term patient monitoring.

To ensure user-friendliness and autonomy, the entire pipeline from image capture to inference to result display is automated and highly responsive. Once the user initiates the image capture process, the system handles all subsequent steps without requiring manual intervention. The interface, built using Flask or Django, offers a clean and intuitive platform for local or remote access. It enables users to interact with the system via a touchscreen or browser-based interface, view live camera feeds, trigger new scans, and monitor system status in real time.

Lightweight and efficient communication protocols such as MQTT are employed for sending and receiving data when internet connectivity is available, particularly in settings with limited bandwidth. These protocols allow for the seamless integration of the system into larger healthcare monitoring infrastructures, enabling real-time alerts, logging of patient data, and even periodic transmission of health metrics to cloud databases or centralized health portals for long-term analysis and follow-up.

Furthermore, optional integration of environmental sensors, such as UV exposure detectors, adds a valuable contextual dimension to the system. This data, when correlated with skin lesion classification outcomes, can help in identifying potential environmental risk factors for skin damage or cancer development. These insights can be recorded and visualized through the user interface, providing a more holistic health-monitoring experience.

In summary, the system operates through a logical and well-orchestrated flow that begins with high-resolution image acquisition, moves through a robust preprocessing stage involving contrast normalization and resizing, followed by deep learning inference via TensorFlow Lite or PyTorch Mobile. The final prediction, along with the confidence score, is then clearly displayed on the touchscreen or made accessible through the GUI. Optional cloud integration and sensor data provide additional utility, enhancing the diagnostic value of the system.

All these operations are efficiently managed by the Raspberry Pi 4, which serves as the computational core of the system. Its versatile GPIO pins handle peripheral interfacing, its USB ports support external camera and display connectivity, and its onboard Wi-Fi (or external dongle) enables IoT communication. Despite its small form factor, the Raspberry Pi offers sufficient processing power to handle deep learning inference and peripheral management simultaneously, making it an ideal choice for portable, low-cost, AI-powered healthcare solutions.

Ultimately, this skin cancer detection system represents a practical and impactful application of embedded AI, bridging the gap between advanced diagnostic tools and underserved communities. Its design ensures portability, cost-efficiency, and ease of use, offering a scalable approach to early cancer detection and public health improvement.

CHAPTER 5

SOFTWARE IMPLEMENTATION

CHAPTER-5

SOFTWARE IMPLEMENTATION

5.1 DESCRIPTION

Raspbian is a Debian-based (32 bit) computer operating system for Raspberry Pi. There are several versions of Raspbian including Raspbian Buster and Raspbian Stretch. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers. Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

Raspbian uses PIXEL, Pi Improved X-Window Environment, Lightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft Pi as well as a lightweight version of Chromium as of the latest version.

Though I did not intend to run the Raspberry Pi as a desktop computer, the Raspbian operating system does provide users with the LXDE desktop environment. The Pi does not have a great deal of processor speed or memory, but it does have enough resources to run LXDE and a handful of applications. So long as the user does not wish to do a lot at once, the Pi offers a fairly responsive desktop interface. I probably would not run heavier programs such as LibreOffice or Firefox on the Pi, but Raspbian does provide the Epiphany web browser and a few other desktop programs.

5.2 PYTHON:

Python is a very useful programming language that has an easy to read syntax, and allows programmers to use fewer lines of code than would be possible in languages such as assembly, C, or Java. The Python programming language actually started as a scripting language for Linux. Python programs are similar to shell scripts in that the files contain a series of commands that the computer executes from top to bottom. Like shell scripts, Python can automate tasks like batch renaming and moving large amounts of files. It can be used just like a command line

with IDLE, Python's REPL (read, eval, print, loop) function. However, there are more useful things you can do with Python. For example, you can use Python to program things like:

- Web applications
- Desktop applications and utilities
- Special GUIs
- Small databases
- 2D games

Python also has a large collection of libraries, which speeds up the development process. There are libraries for everything you can think of – game programming, rendering graphics, GUI interfaces, web frameworks, and scientific computing.

Many (but not all) of the things you can do in C can be done in Python. Python is generally slower at computations than C, but its ease of use makes Python an ideal language for prototyping programs and designing applications that aren't computationally intensive.

5.3 CONFIGURE RPI AND COMPILATION STEPS:

Regardless of whether you are using the full desktop or a headless setup, you will need to perform some basic configuration steps on your new Raspberry Pi installation. These steps can from a *terminal* (a text input/output environment).

You should be automatically logged into the X windows manager (otherwise known as the *desktop*). To open a terminal, simply click on the Terminal icon on the top left of the desktop. You should be immediately presented with a command prompt in a terminal window.

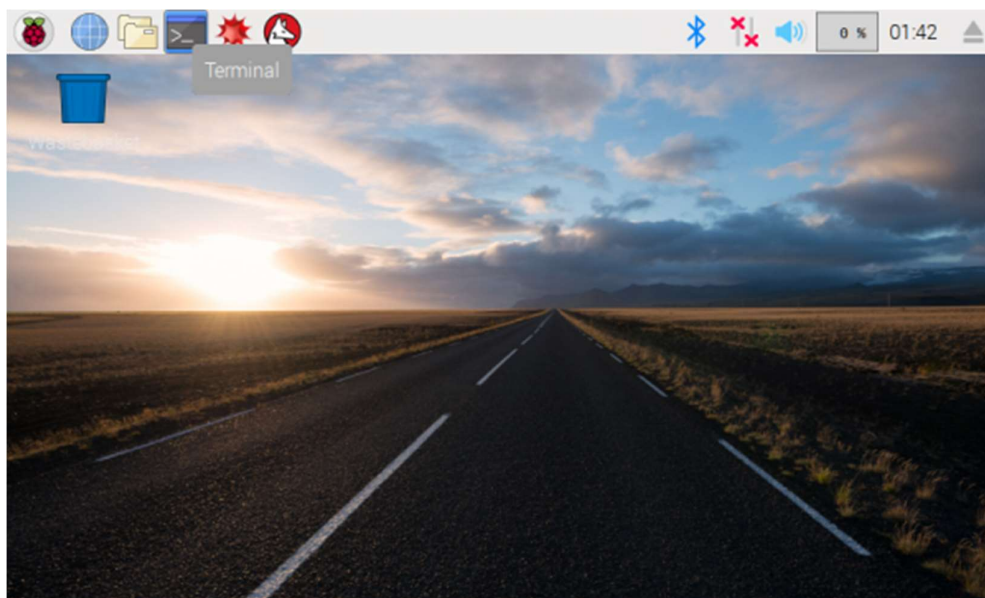
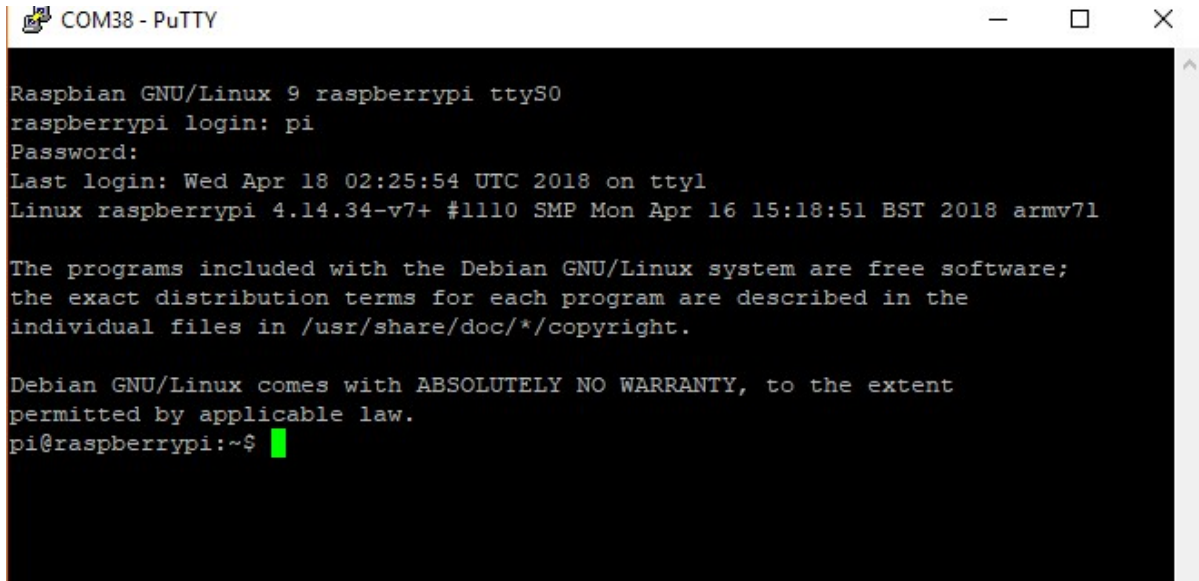


Fig 5.1: Raspberry Pi Monitor

With a headless setup, everything you do will be through a terminal. When you connect to your Pi through Serial or SSH, you will be presented with a login prompt on the terminal. Enter the default credentials:

- Username: pi
- Password: raspberry

You will be presented with a command prompt.



```
COM38 - PuTTY

Raspbian GNU/Linux 9 raspberrypi ttyS0
raspberrypi login: pi
Password:
Last login: Wed Apr 18 02:25:54 UTC 2018 on tty1
Linux raspberrypi 4.14.34-v7+ #1110 SMP Mon Apr 16 15:18:51 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
pi@raspberrypi:~$
```

Fig 5.2: Raspberry Pi Compilation

5.3.1 RUN THE CONFIG TOOL:

From your command prompt, enter the command:

```
COPY CODEsudo raspi-config
```

If asked to enter a password, type out the default password: `raspberry`.

You will be given several options on how to configure your Raspberry Pi.

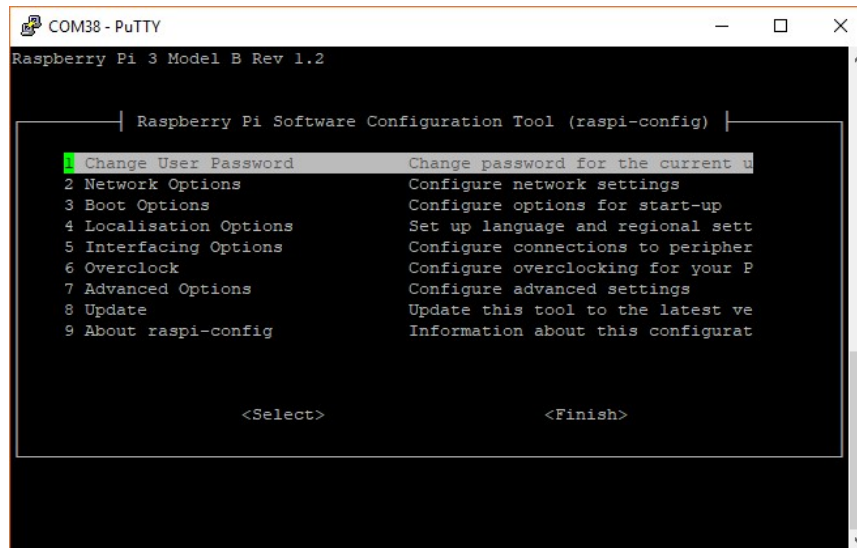


Fig 5.3: Raspberry Pi Configuration

- Use the arrow keys to select 1 Change User Password and follow the on-screen prompts to change your default password.
- Next, select 2 Network Options
- In the following screen, select N2 Wi-fi, and follow the prompts to connect your Pi to a local WiFi network (assuming you have one available).

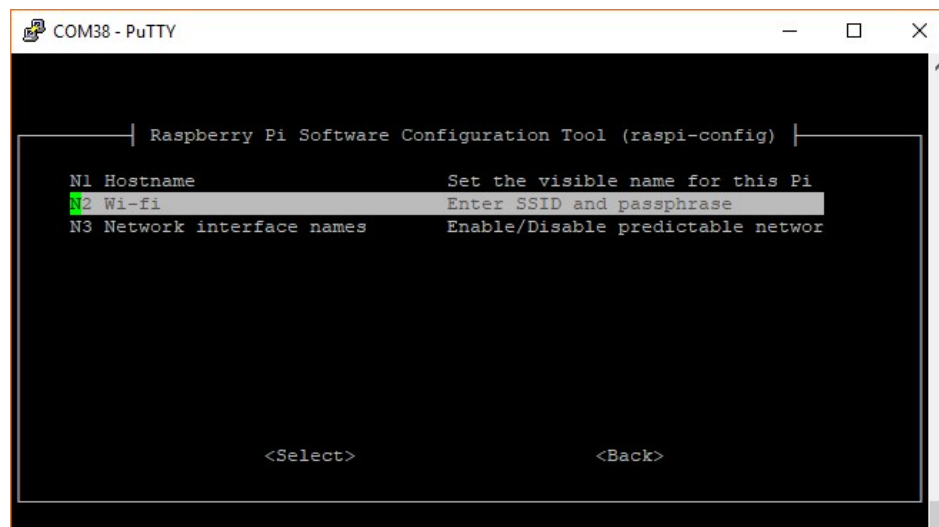


Fig 5.4: Raspberry Pi setting

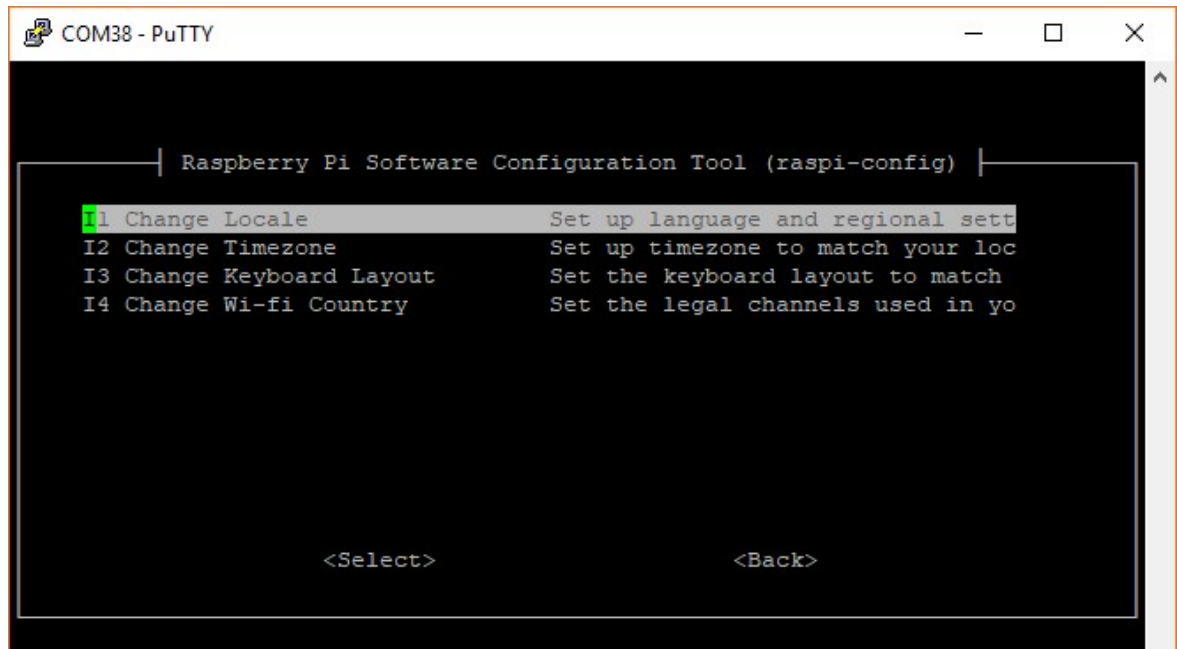


Fig 5.5: Raspberry Pi Configuration

- Select I1 Change Local
- Scroll down to highlight en_GB.UTF-8 UTF-8, and press the spacebar to deselect it (the asterisk '*' will disappear)
- Scroll to find your language/country and press space to select it (an asterisk '*' will appear next to your selection)
- If you live in Great Britain, you can just leave en_GB.UTF-8 UTF-8 selected
- If you live in the United States, you will probably want to select en_US.UTF-8 UTF-8.

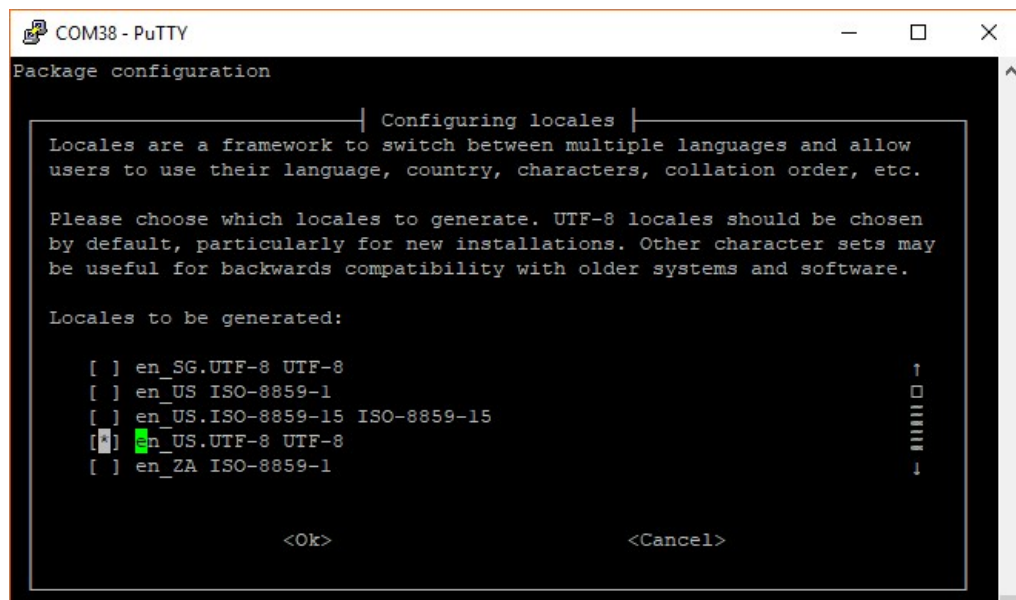


Fig 5.6: Configuring Locales

- Press *enter* to save the changes
- On the next screen, highlight your chosen localization option (e.g. en_US.UTF-8 if you're in the United States) and press *enter*.

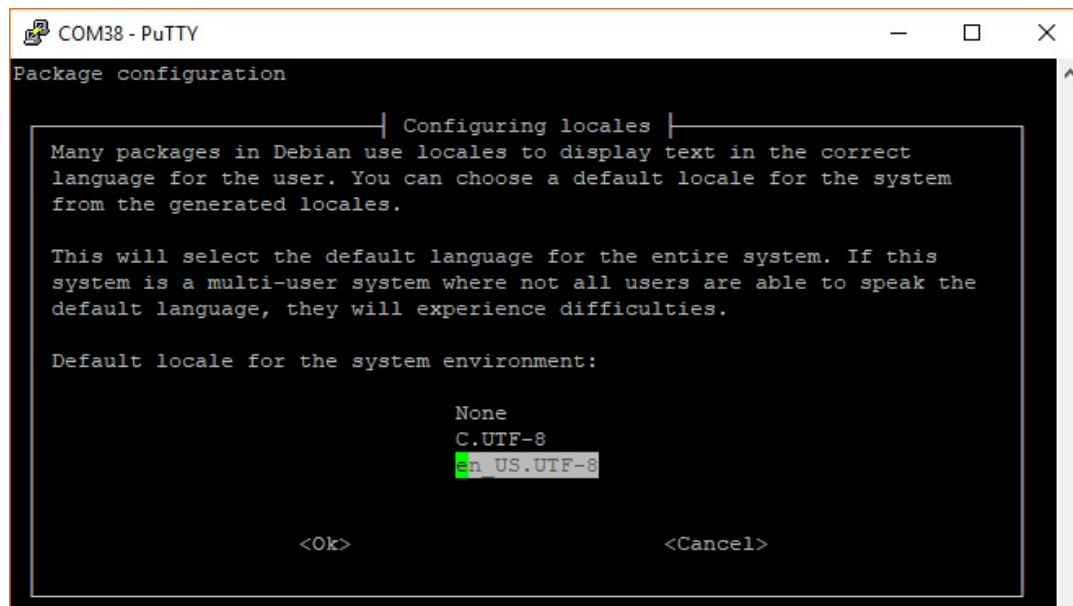


Fig 5.7: Raspberry Pi Tools

- Go back into the 4 Localisations Options, and select I2 Change Timezone
- Follow the prompts to select your timezone.
- Back in 4 Localisations Options, select I3 Change Keyboard Layout
- Choose your preferred layout (the default *Generic 105-key (Intl) PC* seems to work well in most situations)
- On the next screen, select the layout for your language/country

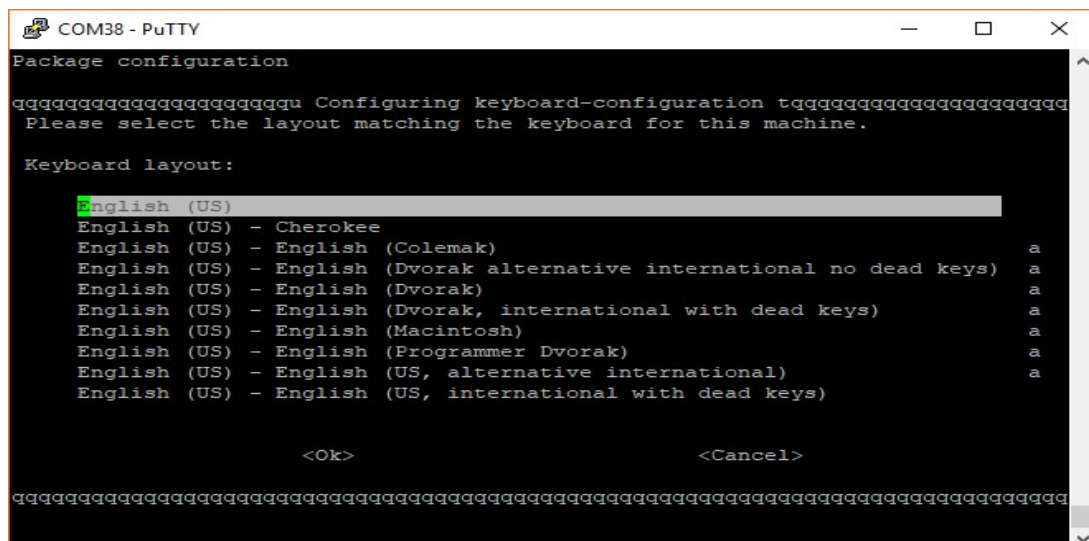


Fig 5.8: Raspberry Pi Packages

- If you live in Great Britain, you can leave English (UK) selected. Otherwise, select Other, press *enter*, and follow the prompts to select your language/country. If you live in the United States, select English (US), and on the next screen, scroll up to highlight English (US). Pre
- Leave The default for the keyboard layout selected, and press *enter*
- Once again, leave the default No compose key selected, and press *enter*
- Leave *No* selected when asked about using *Control+Alt+Backspace*, and press *enter*
- After a few moments, you will be dropped back into the main Configuration Tool menu.

Select 5 Interfacing Options.

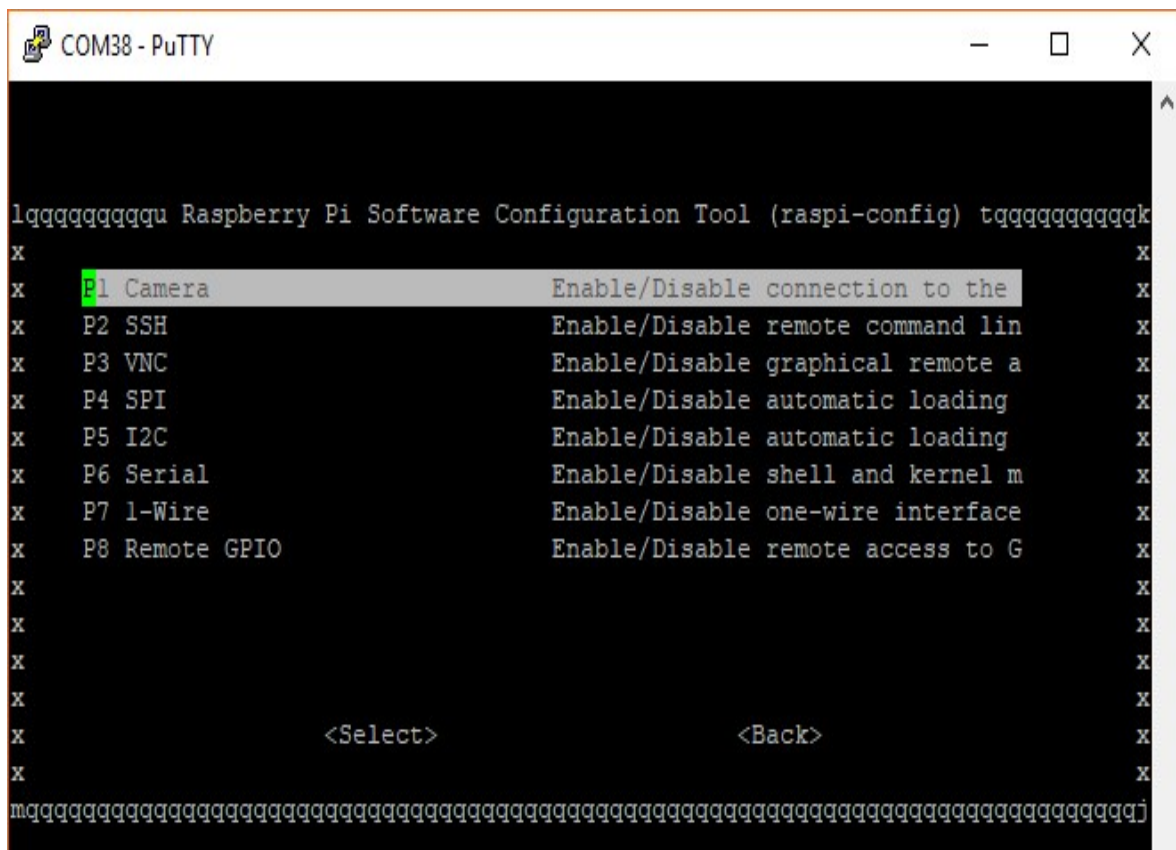


Fig 5.9: Raspberry Pi Software Tools

- Feel free to enable the *Camera* interface and *SSH* if you think you'll need them
- Select *SPI*, select **yes** on the following screen, press *enter*
- Repeat for *I2C*
- Repeat for *Serial*

Back in the main screen, select **7** Advanced Options.

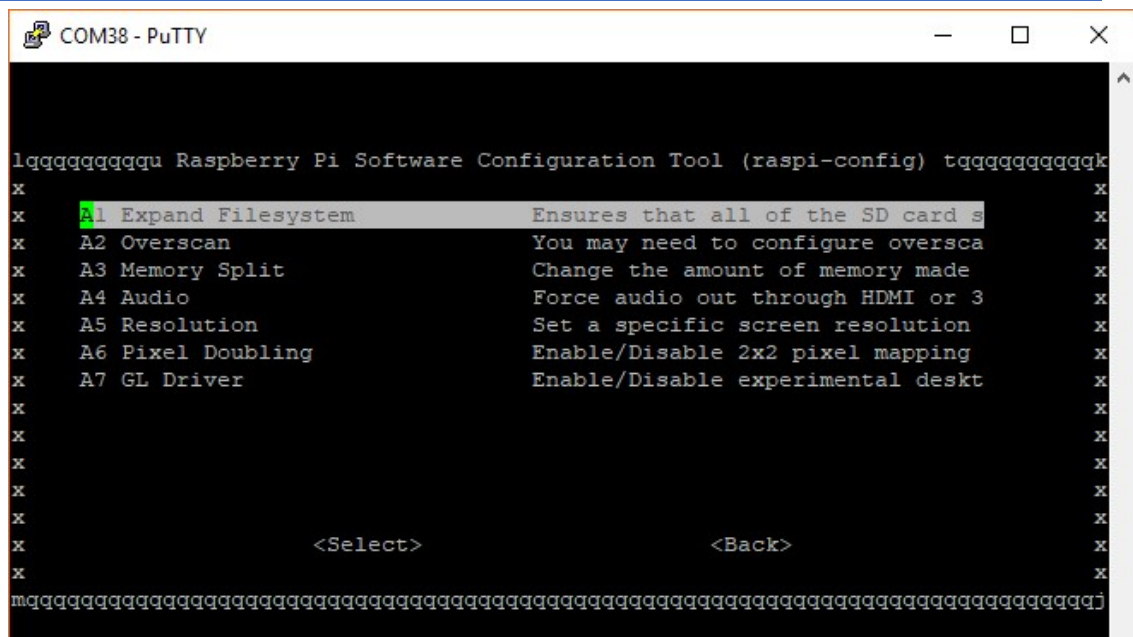


Fig 5.10: Raspberry Pi Configuration Tools

- Select A1 Expand Filesystem, and press *enter*
- Go back into 7 Advanced Options, select A4 Audio, highlight 1 Force 3.5mm ('headphone') jack, and press *enter*
- Use the *right arrow* key to select Finish, and press *enter*. If asked to reboot, select Yes and press *enter*. Wait while your Raspberry Pi restarts. If you are using a Serial or SSH terminal, log back in using the username `pi` and the password you created earlier.

5.3.2 USING PYTHON 3 IN RASPBIAN OS:

By default, Raspbian (Stretch version April 2018 and earlier) uses Python 2. However, versions 2 and 3 come installed by default. We just have to make 1 minor change so that the Pi uses Python 3 whenever we type `python` into a terminal.

In a terminal window, enter the following command:

```
COPY CODEpython --version
```

You should see which version is being used by default. For example, you might see `Python 2.7.13`. If you see that your OS is using Python 2 by default, you'll need to change it to use the Python 3 installation. We want to this so that Python 3 is used every time we log in.

Enter the command:

```
COPY CODEnano ~/.bashrc
```

`.bashrc` is a file that resides in the user's home directory (the user *pi* in this case). The file acts as a shell script that is run each time that specific user opens a terminal (or logs in over SSH, Serial, etc.). It can help to customize the user environment, and you will likely see a number of other commands already in there.

If you are using Raspbian Lite, the Python package manager, *pip*, does not come pre-installed. As a result, you will need to install it with the commands:

```
COPY CODEsudo apt-get update
sudo apt-get install python3-pip
```

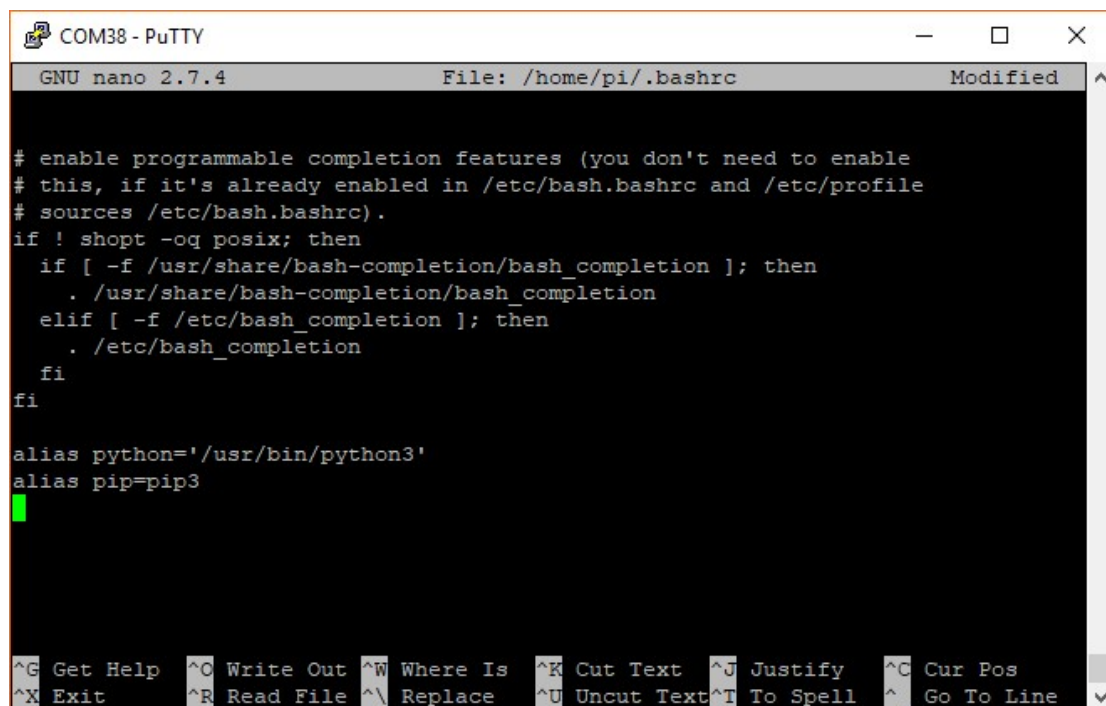
Press *y* when prompted.

Note that to use *pip* for Python 3, you will need to use the command `pip3`. However, we can `.bashrc` file to use *pip* instead of *pip3*, as the rest of the tutorial will show examples using *pip*:

```
COPY CODEnano ~/.bashrc
```

Scroll down to the bottom, and add the following command to the file:

```
COPY CODEalias pip=pip3
```



```
COM38 - PuTTY
GNU nano 2.7.4      File: /home/pi/.bashrc      Modified
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

alias python='/usr/bin/python3'
alias pip=pip3
```

Fig 5.11: Raspberry Pi Bash Script

COPY CODEsource

One of the coolest features of Python is that it is an *interpreted* language (OK, in reality, Python scripts are first compiled to some bytecode, and that bytecode is interpreted). This means that we don't need to run a separate *compile* step (i.e. translate our program into machine code) in order to run our program. In fact, we can even run the interpreter in what's known as *interactive mode*. This will allow us to test out commands one line at a time!

To start, we'll tell Python to print the phrase, "Hello, World!" to the terminal. We'll do this first from the interpreter and then we'll create a file and run it as a program. This will show you two of the main ways to interact with Python.

5.4 GETTING STARTED WITH THE INTERPRETER

From a terminal, enter the following command to start the Python interpreter:

COPY CODEpython

You should be presented with a different command prompt, consisting of 3 greater-than signs `>>>`. Type the following command:

COPY CODEprint("Hello, World!")

Once you press *enter*, you should see the phrase `Hello, World!` repeated back to you.

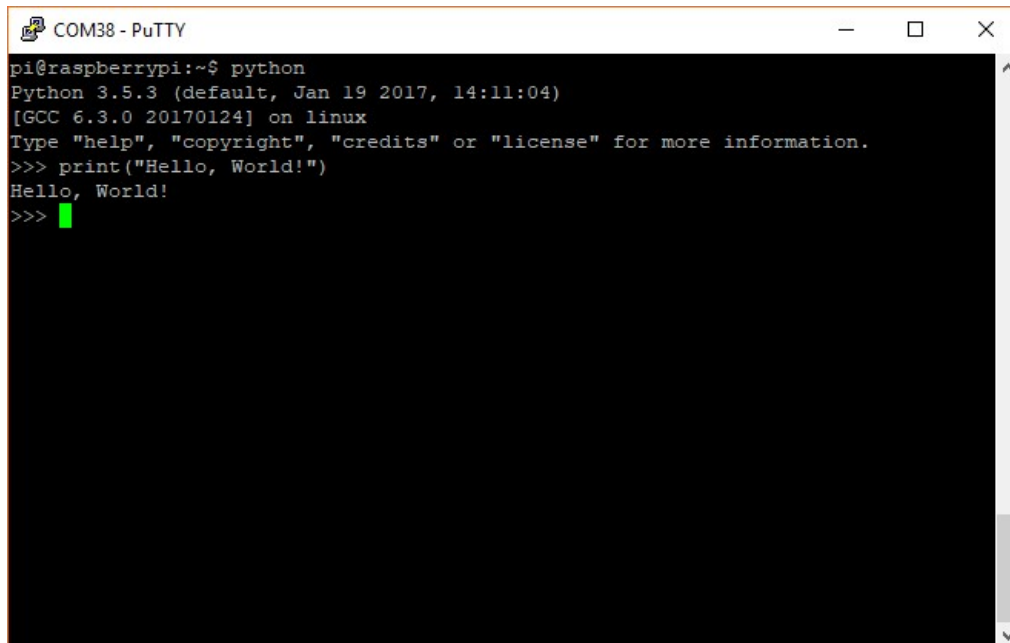
A screenshot of a terminal window titled "COM38 - PuTTY". The terminal shows the command `python` being executed at the prompt `pi@raspberrypi:~$`. The output shows the Python version `Python 3.5.3 (default, Jan 19 2017, 14:11:04)` and the compiler `[GCC 6.3.0 20170124] on linux`. It then prompts the user to type `"help"`, `"copyright"`, `"credits"` or `"license"` for more information. The user enters `>>> print("Hello, World!")`, and the terminal outputs `Hello, World!`. The prompt `>>>` is followed by a green cursor.

Fig 5.12: Program in Terminal

And that's it! You just ran your first Python program!

Exit out of the interpreter by entering:

```
COPY CODEexit()
```

5.4.1 RUNNING THE PYTHON PROGRAM FROM A FILE:

You can individually enter and run commands one line at a time in the Python interpreter, which is incredibly useful for trying out different commands (or using it as a calculator!). Often, you will want to save your commands together in one or more files so that you can run them all at once.

The simplest way to do this is to create a file from the terminal, although you are welcome to use the Raspbian graphical editor, *Leafpad*, as well (found under *Accessories > Text Editor* when you click on the Raspberry Pi icon in the top left).

Still in a terminal, enter the command:

```
COPY CODEnano hello.py
```

This creates a file named *hello.py* in your home directory (*/home/pi*) and starts editing it with the *nano* program.

In this file, enter the following on the first line:

```
COPY CODEprint("Hello, World!")
```

Save and exit (*ctrl+x* followed by *y* and then *enter*). Back in the Linux command prompt, enter the command:

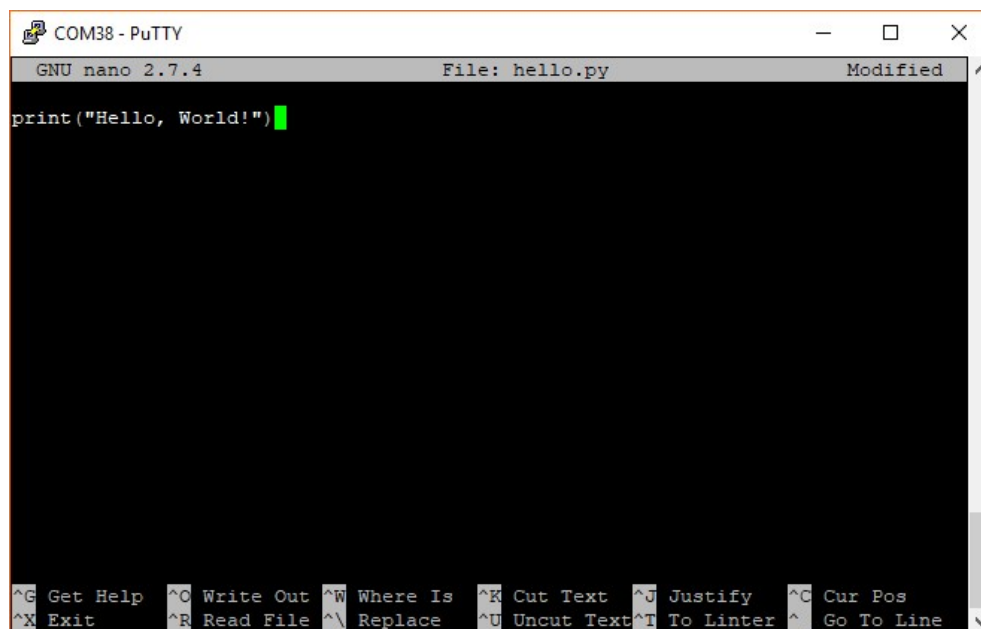
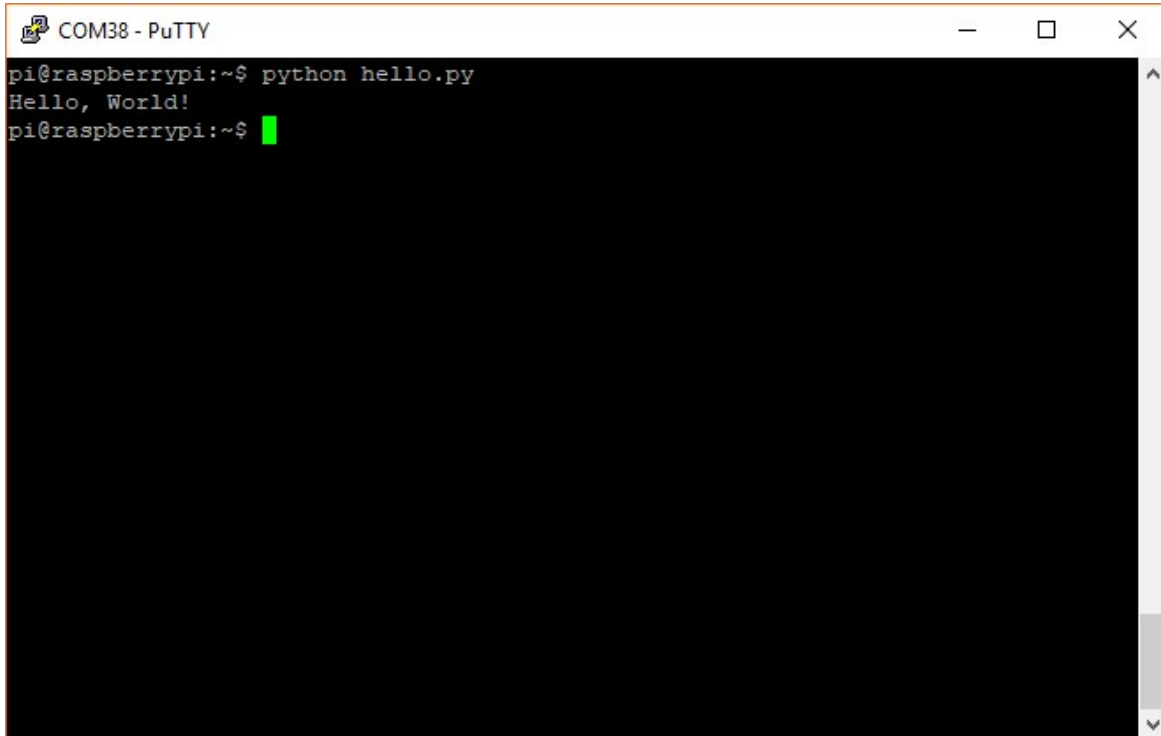


Fig 5.13: Printing output in the Terminal


```
COPY CODEpython hello.py
```

This should run the code found in the file *hello.py*. In our case, you should see the iconic phrase `Hello, World!` printed out in the console.



```
COM38 - PuTTY
pi@raspberrypi:~$ python hello.py
Hello, World!
pi@raspberrypi:~$
```

Fig 5.14: Code Output

To summarize what we just did, you can use the `python` command on its own to begin an *interactive interpreter session* that allows you to type commands in real time. If you specify a file, such as `python <FILE>.py`, the Python interpreter will run the commands found in the file without giving you an interactive session.

Note that the filename suffix `.py` is not required for the interpreter to run the code found inside. However, it can be very helpful to keep your files organized so that when you see a file ending in `.py`, you will know that it contains Python code. The `.py` suffix is also necessary when making *modules*, which we'll cover later.

5.4.2 DEVELOPMENT ENVIRONMENTS:

The simplest way to create Python programs is to write your code in a text editor (e.g. nano, vim, emacs, Midnight Commander, Leafpad, etc.), save it, and then run it from the terminal with the command `python <FILE>.py`. You are welcome to continue working through this guide using a text editor and command line.

Some users prefer to use an *integrated development environment* (IDE) when developing code. IDEs offer a number of benefits including syntax highlighting, code completion, one-click running, debugging hints, etc. However, most IDEs require a graphical interface to use, which means you will need to be on the full desktop version of Raspbian.

5.4.3 IDLE:

IDLE is the default Python editor that has been available on Raspbian for many generations. The good news is that it has a built-in interpreter, which allows you to run commands one at a time to test code. The bad news is that it doesn't show line numbers, and it only works with Python (but you're only here for Python anyway, right?).

Open IDLE by selecting the Raspberry Pi logo in the top-left, and click *Programming > Python 3 (IDLE)*. You should be presented with the Python interactive interpreter.

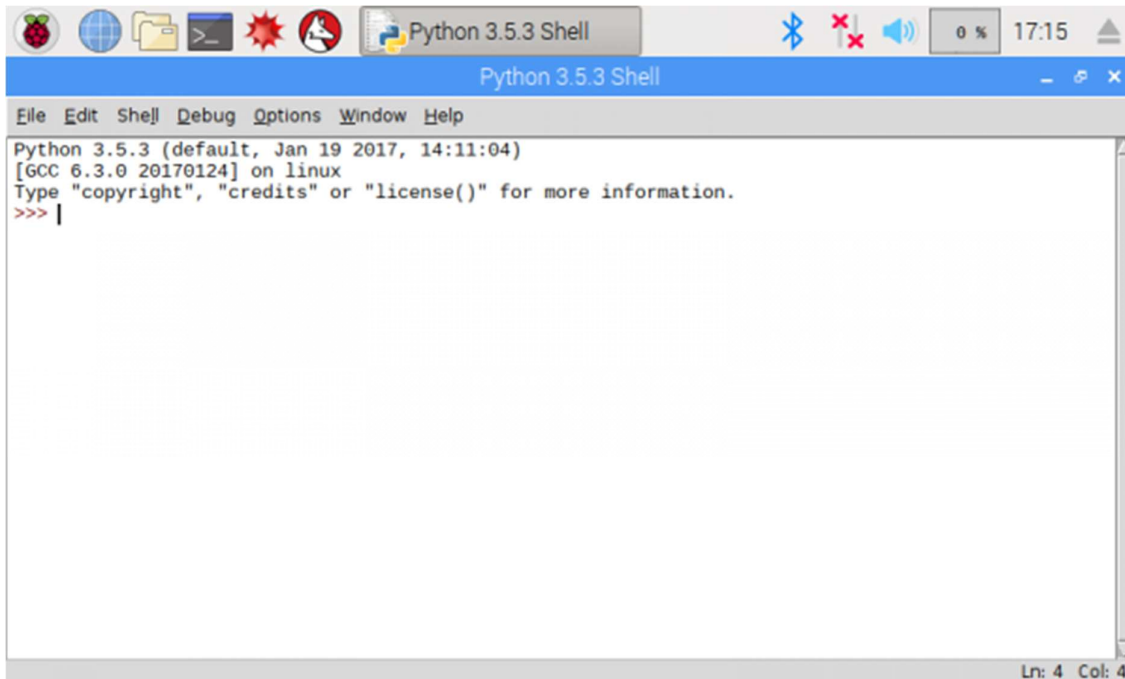


Fig 5.15: Python Shell Window

To write a program, go to *File > New File*. Enter in your code.

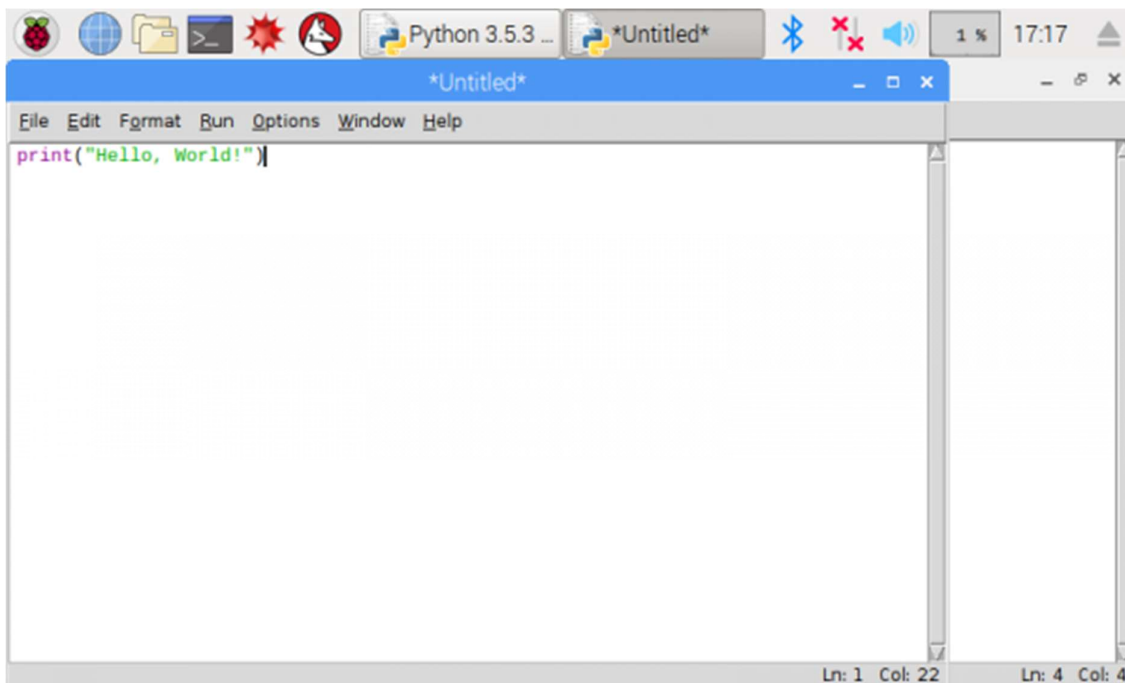


Fig 5.16: Program in Python

Click *File > Save As...* to save your code to a Python file (don't forget the *.py* suffix!). Click *Run > Run Module* to run your program.

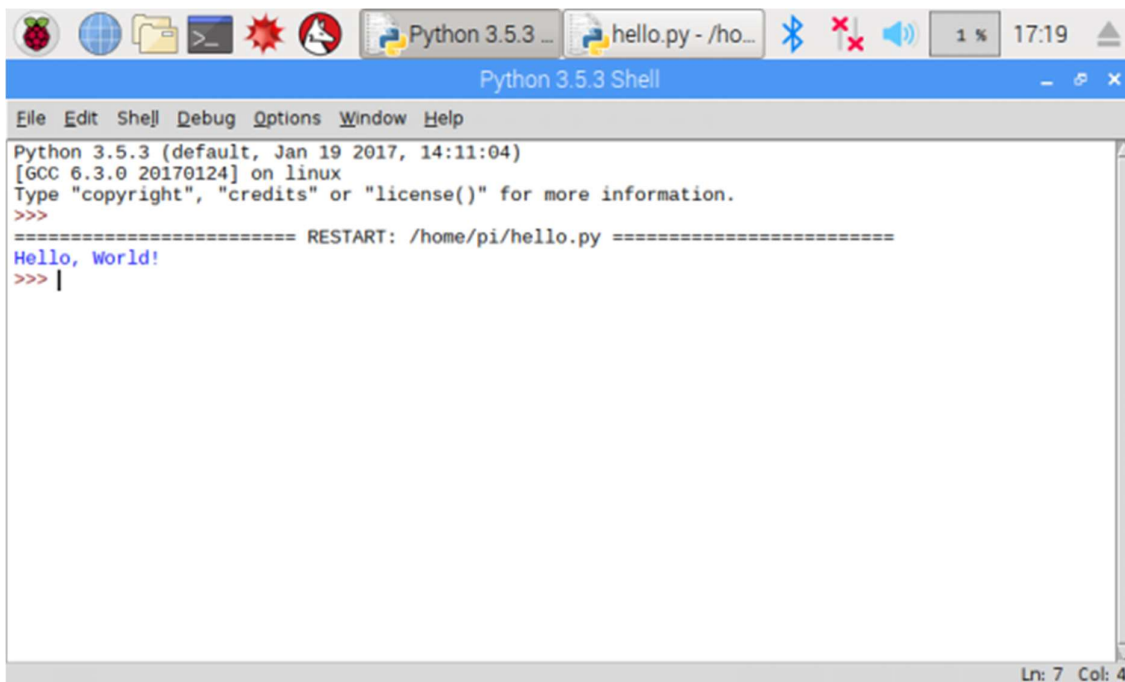


Fig 5.17: Output of Python Code

To ensure efficient memory usage and seamless data processing within our Raspberry Pi-based system, it was essential to understand and manage the various data types used in Python and associated libraries like NumPy. Since our project involves image processing, machine learning inference, and hardware integration all of which demand precision, performance, and optimization we compiled a data type table highlighting the bit-size and data range of commonly used types. This table serves as a reference for selecting appropriate data types for variables such as pixel values, sensor inputs, and model parameters, ultimately contributing to improved processing speed, reduced memory overhead, and accurate real-time performance on resource-constrained embedded hardware like the Raspberry Pi.

Table 5.1 Data Types

S.NO	Data types	Size in Bits	Data Range
1	int	32 / 64 bits	Unlimited (bounded by available memory)
2	float	64 bits (double)	$\pm 1.8 \times 10^{308}$, precision ~15 decimal digits
3	bool	8 bits (internally)	True or False
4	str	Variable	Unicode text; 1–4 bytes per character
5	numpy.int8	8 bits	–128 to +127
6	numpy.uint8	8 bits	0 to 255
7	numpy.int16	16 bits	–32,768 to +32,767
8	numpy.uint16	16 bits	0 to 65,535
9	numpy.int32	32 bits	–2,147,483,648 to +2,147,483,647
10	numpy.uint32	32 bits	0 to 4,294,967,295
11	numpy.float32	32 bits	$\sim \pm 3.4 \times 10^{38}$, precision ~7 decimal digits
12	numpy.float64	64 bits	$\sim \pm 1.8 \times 10^{308}$, precision ~15 decimal digits

CHAPTER 7

CONCLUSION & FUTURE SCOPE

CHAPTER-7

CONCLUSION & FUTURE SCOPE

7.1 CONCLUSION:

This project not only addresses a practical problem in vehicle access control but also lays the groundwork for future developments in automated number plate recognition systems. The successful integration of IoT and machine learning technologies highlights the potential for innovative solutions in security and access management. This project paves the way for further research and enhancements, contributing to the evolution of smart transportation systems and automated security solutions.

The IoT-integrated machine learning number plate recognition system using Raspberry Pi and Python is a pioneering project that showcases the intersection of technology and practical applications in security and access control. This system is designed to automate the process of recognizing vehicle number plates, thereby facilitating authorized access to restricted areas such as parking lots, residential complexes, and secured institutional premises. By leveraging the capabilities of Raspberry Pi a low-cost, powerful, and portable computing platform — the project delivers a cost-effective yet efficient solution to a persistent challenge in modern security infrastructure.

The entire system operates through a seamless interaction between hardware and software. The camera module captures the number plate image when a vehicle is detected within a certain range using an ultrasonic sensor. This image is processed using OpenCV, which performs tasks like grayscale conversion, noise removal, and character segmentation. Following preprocessing, Optical Character Recognition (OCR) extracts the alphanumeric characters from the plate. These characters are then matched against a database of authorized numbers. If a match is found, access is granted through a gate mechanism driven by a servo motor. In case of a mismatch, a buzzer alerts the operator of unauthorized access. This smooth workflow ensures that the system is both intelligent and responsive, capable of operating in real-time with minimal delay.

Ultimately, this work signifies a small yet impactful step toward the broader vision of smart cities. By automating and optimizing vehicle access control, such systems not only enhance security but also contribute to traffic management, resource efficiency, and user convenience. As technology continues to evolve, solutions like this will become increasingly indispensable in building safer, smarter, and more connected environments for everyone.

7.2 FUTURE SCOPE:

While the current implementation of the system provides a strong foundation for automated vehicle access control, there remain numerous opportunities for further enhancement and innovation. One promising direction is the development of a custom OCR model, designed specifically to recognize the font styles, sizes, and number plate formats commonly used in specific regions. This would greatly improve the accuracy and speed of plate recognition, especially under challenging conditions such as poor lighting, low-resolution images, or weathered license plates.

Another area of improvement lies in the integration of intelligent sensors like Passive Infrared (PIR) or Infrared (IR) detectors. These sensors can act as triggers for the camera, activating it only when a vehicle approaches, thus reducing unnecessary data capture and conserving power and processing resources. This enhancement would contribute to a more efficient and responsive system, especially in high-traffic environments.

The incorporation of AI and machine learning can further elevate the system's capabilities. By analyzing patterns in access behavior or license plate data over time, the system could make predictive decisions, such as identifying suspicious entries or automatically flagging unauthorized vehicles. This would shift the system from being purely reactive to becoming more proactive and intelligent.

To ensure long-term viability and ease of development, adopting a microservices architecture would offer great benefits. Breaking down the system into modular components—such as image capture, preprocessing, OCR, and database management—allows for individual updates, testing, and scaling without disrupting the entire application. This modularity would also support easier integration with external systems or future hardware upgrades.

To further enhance security, the system could evolve to support multi-factor authentication, such as pairing license plate recognition with RFID tags, facial recognition, or even smartphone-based digital access passes. This would provide an additional layer of verification, suitable for high-security applications.

Lastly, ensuring scalability and adaptability will be essential for real-world deployment. Designing the system in a modular and extensible way would allow it to be implemented in various contexts—from residential gated communities and university campuses to industrial zones and government premises. As the system continues to develop, its ability to evolve with emerging technologies and integrate seamlessly with broader smart city infrastructure will determine its long-term impact and success.

REFERENCES

REFERENCES

- [1] Zang, D., Chai, Z., Zhang, J., Zhang, D., & Cheng, J. (2015). Vehicle license plate recognition using visual attention model and deep learning. *Journal of Electronic Imaging*, 24(3), 033001
- [2] Sirithinaphong, T., & Chamnongthai, K. (1999). The recognition of car license plate for automatic parking system. In *ISSPA'99. Proceedings of the Fifth International Symposium on Signal Processing and its Applications* (IEEE Cat. No. 99EX359) (Vol. 1, pp. 455-457). IEEE.
- [3] Wang, J., Basic, B., & Yan, W. Q. (2018). An effective method for plate number recognition. *Multimedia Tools and Applications*, 77(2), 1679-1692.
- [4] Al-Ghaili, A. M., Mashohor, S., Ramli, A. R., & Ismail, A. (2012). Vertical-edge-based car-license-plate detection method. *IEEE transactions on vehicular technology*, 62(1), 26-38.
- [5] Qin, Z., Shi, S., Xu, J., & Fu, H. (2006). Method of license plate location based on corner feature. In *2006 6th World Congress on Intelligent Control and Automation* (Vol. 2, pp. 8645-8649). IEEE.
- [6] Rasheed, S., Naeem, A., & Ishaq, O. (2012). Automated number plate recognition using hough lines and template matching. In *Proceedings of the World Congress on Engineering and Computer Science* (Vol. 1, pp. 24-26).
- [7] Panchal, T., Patel, H., & Panchal, A. (2016). License plate detection using Harris corner and character segmentation by integrated approach from an image. *Procedia Computer Science*, 79, 419-425.
- [8] Zhang, Z., & Wang, C. (2012). The research of vehicle plate recognition technical based on BP neural network. *Aasri Procedia*, 1, 74-81.

- [9] Gao, P., Zeng, Z., & Sun, S. (2018). Segmentation-Free Vehicle License Plate Recognition Using CNN. In International Conference On Signal and Information Processing, Networking And Computers (pp. 50-57). Springer, Singapore.
- [10] Masood, S. Z., Shu, G., Dehghan, A., & Ortiz, E. G. (2017). License plate detection and recognition using deeply learned convolutional neural networks. arXiv preprint arXiv:1703.07330.
- [11] Sundararaman, V., Vijayalakshmi, T. G., Swathi, G. V., & Mohapatra, S. (2016). Automatic License Plate Recognition System Using Raspberry Pi. In Proceedings of the International Conference on Recent Cognizance in Wireless Communication & Image Processing (pp. 217-222). Springer, New Delhi.
- [12] Thangam, E. C., Mohan, M., Ganesh, J., & Sukesh, C. V. (2018). Internet of Things (IoT) based Smart Parking Reservation System using Raspberry-pi. International Journal of Applied Engineering Research, 13(8), 5759-5765.
- [13] Kochlá, M., Hodo, M., echovi, L., Kapitulík, J., & Jureka, M. (2014, September). WSN for traffic monitoring using Raspberry Pi board. In 2014 Federated Conference on Computer Science and Information Systems (pp. 1023-1026). IEEE.
- [14] Iszaidy, I., Ngadiran, R., Ahmad, R. B., Jais, M. I., & Shuhaizar, D. (2016). Implementation of raspberry Pi for vehicle tracking and travel time information system: A survey. In 2016 International Conference on Robotics, Automation and Sciences (ICORAS) (pp. 1-4). IEEE.
- [15] Chang, S. L., Chen, L. S., Chung, Y. C., & Chen, S. W. (2004). Automatic license plate recognition. IEEE transactions on intelligent transportation systems, 5(1), 42-53.
- [16] Anagnostopoulos, C. N. E., Anagnostopoulos, I. E., Loumos, V., & Kayafas, E. (2006). A license plate-recognition algorithm for intelligent transportation system applications. IEEE Transactions on Intelligent transportation systems, 7(3), 377-

392.

- [17] Babu, C. N. K., Subramanian, T. S., & Kumar, P. (2010). A feature based approach for license plate-recognition of Indian number plates. In 2010 IEEE International Conference on Computational Intelligence and Computing Research (pp. 1-4). IEEE.
- [18] Kocer, H. E., & Cevik, K. K. (2011). Artificial neural networks based vehicle license plate recognition. *Procedia Computer Science*, 3, 1033-1037.
- [19] Sathya, K. B., Vaidehi, V., & Kavitha, G. (2017). Vehicle License Plate Recognition (VLPR). 2017 Trends in Industrial Measurement and Automation (TIMA). doi:10.1109/tima.2017.8064786
- [20] Jagtap, J., & Holambe, S. (2018). Multi-Style License Plate Recognition using Artificial Neural Network for Indian Vehicles. 2018 International Conference on Information , Communication, Engineering and Technology (ICICET). doi:10.1109/icicet.2018.8533707

APPENDIX

APPENDIX

Code For Vehicle Number Plate Recognition System for Authorised access:

Python Code:

```
#!/usr/bin/env python
from __future__ import absolute_import, division, print_function

import argparse
import collections
import csv

import RPi.GPIO as GPIO
import datetime
import serial
import numpy as np
import subprocess
import os

import functools
import pandas as pd

setgpio=0

GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
GPIO.output(11,0)

def parse_arguments(args_hook=lambda _ : _):
    parser = argparse.ArgumentParser(
        description=
        'Read license plates from images and output the result as JSON.',
        epilog='Examples:\n'
        'To send images to our cloud service: '
        'python plate_recognition.py --api-key MY_API_KEY /path/to/vehicle-*.jpg\n',
        formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument('-a', '--api-key', help='Your API key.', required=False)
    parser.add_argument(
        '-r',
        '--regions',
        help='Match the license plate pattern of a specific region',
```

```
        required=False,
        action="append")
    parser.add_argument(
        '-s',
        '--sdk-url',
        help="Url to selfhosted sdk example, http://localhost:8080",
        required=False)
    parser.add_argument('--camera-id',
                        help="Name of the source camera.",
                        required=False)
    parser.add_argument('files', nargs='+', help='Path to vehicle images')
    args_hook(parser)
    args = parser.parse_args()
    if not args.sdk_url and not args.api_key:
        raise Exception ('api-key is required')
    return args

def recognition_api(fp,
                  regions= [],
                  api_key=None,
                  sdk_url=None,
                  config= {},
                  camera_id=None,
                  timestamp=None,
                  mmc=None,
                  exit_on_error=True):
    data = dict(regions=regions, config=json.dumps(config))
    if camera_id:
        data['camera_id'] = camera_id
    if mmc:
        data['mmc'] = mmc
    if timestamp:
        data['timestamp'] = timestamp
    response = None
    if sdk_url:
        fp.seek(0)
        response = requests.post(sdk_url + '/v1/plate-reader/',
                                files=dict(upload=fp),
                                data=data)
    else:
        for _ in range (3):
            fp.seek(0)
            response = requests.post(
                'https://api.platerecognizer.com/v1/plate-reader/',
```

```
        files=dict(upload=fp),
        data=data,
        headers= {'Authorization': 'Token ' + api_key})
    if response.status_code == 429: # Max calls per second reached
        time.sleep(1)
    else:
        break
if not response:
    return {}
if response.status_code < 200 or response.status_code > 300:
    #print(response.text)
    if exit_on_error:
        exit(1)
return response.json(object_pairs_hook=OrderedDict)

def blur(im, blur_amount, api_res, ignore_no_bb=False, ignore_list=None):
    for res in api_res.get('results', []):
        if ignore_no_bb and res['vehicle']['score'] == 0.0:
            continue

        if ignore_list:
            skip_blur = False
            for ignore_regex in ignore_list:
                if re.search(ignore_regex, res['plate']):
                    skip_blur = True
                    break
            if skip_blur:
                continue

        b = res['box']
        width, height = b['xmax'] - b['xmin'], b['ymax'] - b['ymin']
        crop_box = (b['xmin'], b['ymin'], b['xmax'], b['ymax'])
        ic = im.crop(crop_box)

        # Increase amount of blur with size of bounding box
        blur_image = ic.filter(
            ImageFilter.GaussianBlur(radius=math.sqrt(width * height) * .3 *
                                     blur_amount / 10))
        im.paste(blur_image, crop_box)
    return im

def draw_bb(im, data, new_size=(1920, 1050), text_func=None):
    draw = ImageDraw.Draw(im)
    font_path = Path('assets/DejaVuSansMono.ttf')
    if font_path.exists():
        font = ImageFont.truetype(str(font_path), 10)
```

```
else:
    font = ImageFont.load_default()
    rect_color = (0, 255, 0)
    for result in data:
        b = result['box']
        coord = [(b['xmin'], b['ymin']), (b['xmax'], b['ymax'])]
        draw.rectangle(coord, outline=rect_color)
        draw.rectangle(((coord[0][0] - 1, coord[0][1] - 1),
                        (coord[1][0] - 1, coord[1][1] - 1)),
                        outline=rect_color)
        draw.rectangle(((coord[0][0] - 2, coord[0][1] - 2),
                        (coord[1][0] - 2, coord[1][1] - 2)),
                        outline=rect_color)
        if text_func:
            text = text_func(result)
            text_width, text_height = font.getsize(text)
            margin = math.ceil(0.05 * text_height)
            draw.rectangle(
                [(b['xmin'] - margin, b['ymin'] - text_height - 2 * margin),
                 (b['xmin'] + text_width + 2 * margin, b['ymin'])],
                fill='white')
            draw.text((b['xmin'] + margin, b['ymin'] - text_height - margin),
                      text,
                      fill='black',
                      font=font)

    if new_size:
        im = im.resize(new_size)
    return im
```

```
def flatten_dict(d, parent_key="", sep='_'):
    items = []
    for k, v in d.items():
        new_key = parent_key + sep + k if parent_key else k
        if isinstance(v, collections.MutableMapping):
            items.extend(flatten_dict(v, new_key, sep=sep).items())
        else:
            if isinstance(v, list):
                items.append((new_key, json.dumps(v)))
            else:
                items.append((new_key, v))
    return dict(items)

def flatten(result):
    plates = result['results']
    #print("=====PLATES", plates)
```



```
del result['results']
del result['usage']
if not plates:
    return result
for plate in plates:
    data = result.copy()
    data.update(flatten_dict(plate))
return data
def save_results(results, args):
    path = Path(args.output_file)
    if not path.parent.exists():
        print('%s does not exist' % path)
        return
    if not results:
        return
    if args.format == 'json':
        with open (path, 'w') as fp:
            json.dump(results, fp)
    elif args.format == 'csv':
        fieldnames = []
        for result in results [:10]:
            candidate = flatten(result.copy()).keys()
            if len(fieldnames) < len(candidate):
                fieldnames = candidate
        with open (path, 'w') as fp:
            writer = csv.DictWriter(fp, fieldnames=fieldnames)
            writer.writeheader()
            for result in results:
                writer.writerow(flatten(result))
def custom_args(parser):
    parser.epilog += 'To blur images: python plate_recognition.py --sdk-url http://localhost:8080 --blur-amount 4 --blur-plates /path/to/vehicle-*.jpg\n'
    parser.epilog += 'To save results: python plate_recognition.py --sdk-url http://localhost:8080 -o data.csv --format csv /path/to/vehicle-*.jpg\n'
    parser.add_argument('--engine-config', help='Engine configuration.')
    parser.add_argument('-o', '--output-file', help='Save result to file.')
    parser.add_argument('--format',
                        help='Format of the result.',
                        default='json',
                        choices='json csv'.split())
    parser.add_argument(
        '--mmc',
        action='store_true',
        help='Predict vehicle make and model. Only available to paying users.')
    parser.add_argument(
        '--blur-amount',
        help=
```

```
'Amount of blurring to apply on the license plates. Goes from 0 (no blur) to 10. Defaults to 5.
',
    default=5,
    type=float,
    required=False)
parser.add_argument(
    '--blur-plates',
    action='store_true',
    help='Blur license plates and save image in filename_blurred.jpg.',
    required=False)
def json_extract(obj, key):
    """Recursively fetch values from nested JSON."""
    arr = []
    def extract (obj, arr, key):
        """Recursively search for values of key in JSON tree."""
        if isinstance(obj, dict):
            for k, v in obj.items():
                if isinstance(v, (dict, list)):
                    extract (v, arr, key)
                elif k == key:
                    arr.append(v)
        elif isinstance(obj, list):
            for item in obj:
                extract (item, arr, key)
    return arr

values = extract (obj, arr, key)
return values
def main ():
    args = parse_arguments(custom_args)
    paths = args.files
    results = []
    engine_config = {}
    if args.engine_config:
        try:
            json.loads(args.engine_config)
        except json.JSONDecodeError as e:
            print(e)
        return
    for path in paths:
        with open (path, 'rb') as fp:
            api_res = recognition_api(fp,
                                     args.regions,
                                     args.api_key,
                                     args.sdk_url,
                                     config=engine_config,
                                     camera_id=args.camera_id,
                                     mmc=args.mmc)
    if args.blur_plates:
        im = blur (Image.open(path), args.blur_amount, api_res)
```

```
        filename = Path(path)
        im.save(filename.parent / ('%s_blurred%s' %
                                   (filename.stem, filename.suffix)))    results.append(api_res)
if args.output_file:
    save_results(results, args)
else:
    #print(results)
    #print(type(results))
    #print("-----\n")
    #print (json.dumps(results, indent=2))
    jtopy=json.loads(json.dumps(results))
    onev=jtopy[0]
    #print ("==1",onev)
    #print(type(onev))
    #print("-----\n")
    twov=onev['results']
    #print("==2"twov) # type is list
    #print(len(twov))
    numberofelements=len(twov)
    print ("----- NUMBER OF OBJECTS ",numberofelements)
    for i in range(0,numberofelement):
        #print(twov[i]['plate'])
        #print("-----\n")
        platenumber=twov[i]['plate']
        detectedplated=platenumber.isupper()
        print("DETECTED NUMBER PLATE:", platenumber)
        GPIO.output(11,1)
        time.sleep(2)
        GPIO.output(11,0)
if __name__ == '__main__':
    main()
```