Project report on

Development of a Web-Controlled Surveillance Robot Using Raspberry Pi for Real-Time Monitoring

Submitted by

Aditya Mallick(2341013087)

Kshitij Prasad(2341016424)

Sarvesh Prasad(2341016425)

B. Tech. (CSE(IOT) 4th Semester (Section–23412B1)

CENTRE FOR INTERNET OF THINGS
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Bhubaneswar, Odisha, India.

(May, 2025)

Abstract

The increasing need for efficient and remote surveillance solutions has led to the integration of robotics and web technologies in monitoring systems. This project presents the design and implementation of a web-controlled surveillance robot powered by a Raspberry Pi for real-time monitoring applications. The system is designed to navigate remotely controlled environments and transmit live video footage to a web-based interface accessible via any internet-enabled device.

The robot is equipped with a camera module connected to the Raspberry Pi, which captures and streams real-time video using web protocols. A lightweight web server hosted on the Raspberry Pi facilitates the user interface for controlling the robot’s movements and viewing the live feed. The robot’s mobility is achieved through motor drivers and chassis components, allowing directional control through a web interface.

Key features of the system include remote accessibility, low power consumption, ease of deployment, and scalability. The use of open-source tools and cost-effective components makes this solution suitable for applications in home security, industrial surveillance, and disaster-prone area monitoring. The project demonstrates the feasibility and effectiveness of using Raspberry Pi as a core controller for web-based robotic surveillance systems.

Contents

Table of Contents

Abstract i

Contents ii

Chapter 01: Introduction 1

1.1. Introduction 1

1.2. Background 1

1.3. Project Objectives 2

1.4. Scope 2

1.5. Project Management 3

1.6. Overview and Benefits 5

Chapter 02: Background Review & Survey 7

2.1. Related Works 7

Chapter 03: Theoretical Aspects 9

3.1. Internet of Things (IoT) 9

3.2. Features of IoT 9

3.3 Advantages of IoT 10

3.3. Disadvantages of IoT 11

3.4. Application areas of IoT 12

3.5. IOT Technologies and Protocols 13

3.6 Brief Description 15

Chapter 04: Hardware Requirements 16

4.1 Raspberry Pi Board 16

4.2 Camera Module 16

4.3 Power Supply 16

4.4 Motor Driver Module 16

4.5 DC Motors with Wheels 16

4.6 Chassis 17

4.7 Wi-Fi Module 17

4.8 MicroSD Card 17

4.9 Block diagram of the proposed system 17

4.10 Working of the system 18

4.11 Circuit Diagram 19

4.12 Components Required 20

Chapter 05: Software Requirements……………………………………………………………..21

5.1 Thonny IDE (Python) 21

5.2 Logic and Flowchart 21

Chapter 06: Project development & Testing Aspects 24

6.1 Project Development... 24

6.2 Testing Aspects... 24

6.3 Challenges Faced 25

Chapter 07: Conclusion & Future Scope 26

7.1 Result… 26

7.2 Conclusion 26

7.3 Limitations.. 26

7.4 Further Enhancement and Future Scope 26

References.. 27

Appendix 01 28

A01.1. Code Listing.. 28

A01.2. Main Code.. 31

A01.3. Libraries 35

Appendix 02 36

A02.1. Project Proposal Form 36

A02.2. Project Management... 36

A02.3. Bill of Material.. 36

Appendix 03 37

A03.1. Data Sheets.. 37

Chapter 01: Introduction

1.1. Introduction

In today’s rapidly evolving technological landscape, the need for enhanced security and surveillance systems has become increasingly critical. Traditional surveillance systems, while effective, often lack mobility and require constant human supervision, limiting their scope and adaptability in dynamic environments. The integration of robotics with web-based control systems offers a promising solution to these limitations by enabling real-time monitoring and remote operation.

This project explores the development of a mobile surveillance robot that can be controlled via a web interface and is powered by a Raspberry Pi microcomputer. The system is designed to provide a flexible and cost-effective surveillance solution, capable of navigating various terrains and transmitting live video feeds to a remote user. Through a web interface, users can control the robot's movements and observe the monitored environment in real time, using any device with internet access.

The Raspberry Pi serves as the core processing unit of the robot, managing video capture, wireless communication, and motor control. It hosts a lightweight web server to handle user inputs and stream live video from the attached camera module. This setup eliminates the need for dedicated surveillance infrastructure and allows the robot to be deployed in a wide range of applications, such as home security, industrial site monitoring, search and rescue operations, and hazardous environment inspections.

This introduction lays the foundation for the subsequent sections, which detail the design, components, implementation, and performance evaluation of the web-controlled surveillance robot.

1.2. Background

Security and surveillance are critical in both public and private domains. Traditional CCTV systems are limited by static camera placements and lack of interactive control. To overcome these limitations, mobile surveillance robots are gaining popularity for their flexibility, mobility, and ability to be controlled remotely.

Raspberry Pi, a compact and affordable single-board computer, has revolutionized DIY robotics and embedded system development. Its capability to run a full operating system and interface with various hardware modules makes it ideal for real-time surveillance applications.

This project builds upon existing technologies by integrating a live video camera with a Raspberry Pi-based robotic platform that can be controlled via a web interface. This approach removes the need for dedicated software or proximity to the robot, as users can monitor and navigate the robot from any device connected to the internet. The background work involves knowledge of web development, hardware interfacing, wireless communication (e.g., Wi-Fi), and basic robotics. It addresses a growing demand for smarter, more interactive security solutions in a wide range of environments.

1.3. Project Objectives

The primary objective of this project is to design and implement a mobile surveillance robot that can be controlled remotely via a web interface and provide real-time video streaming using a Raspberry Pi. To achieve this, the following specific objectives have been identified:

- **To design a mobile robotic platform** equipped with motors and sensors capable of navigating various environments for surveillance purposes.
- **To integrate a camera module** with the Raspberry Pi to capture and transmit live video feeds over the internet.
- **To develop a web-based user interface** that allows users to control the movement of the robot (forward, backward, left, right) and monitor live video streams remotely.
- **To implement wireless communication** using Wi-Fi for real-time data transmission between the robot and the remote user.
- **To utilize open-source tools and libraries** such as Python, OpenCV, and Flask to enable efficient development and reduce system costs.
- **To ensure system reliability and responsiveness** in real-time scenarios, including low-latency video streaming and smooth robot maneuvering.
- **To evaluate the performance** of the surveillance robot in different environmental conditions and distances to determine its operational range and effectiveness.

1.4. Scope

This project focuses on the development of a low-cost, web-controlled surveillance robot designed for real-time remote monitoring using a Raspberry Pi. The scope of the project includes both the hardware and software aspects necessary for building and operating the surveillance system. Key elements covered within the scope include:

1. **Design and Assembly of the Robot:**
    - Construction of a mobile robotic platform with wheels, motors, and a motor driver.
    - Integration of a Raspberry Pi as the central control unit.
    - Mounting of a camera module for live video streaming.
2. **Web Interface Development:**

- Creation of a simple, user-friendly web-based GUI for remote control.
- Inclusion of directional controls (forward, backward, left, right).
- Display of live video feed on the web interface.

3. **Real-Time Communication:**

   - Implementation of Wi-Fi-based communication between the robot and the web interface.
   - Live video streaming using Python, OpenCV, and Flask framework.

4. **Software Integration:**

   - Programming motor control using GPIO pins on the Raspberry Pi.
   - Deployment of a local web server on the Raspberry Pi to host the control interface.

5. **Testing and Evaluation:**

   ❖ Performance testing of the robot in indoor and controlled outdoor environments.
   ❖ Evaluation of video quality, latency, and control responsiveness over different distances.

Limitations (Out of Scope):

- Advanced features such as obstacle avoidance, autonomous navigation, or facial recognition are not included.
- The system is designed primarily for small-scale use and may not support large-scale or industrial surveillance out of the box.
- Remote control is limited to Wi-Fi range unless extended through network configurations or additional hardware.

1.5. Project Management

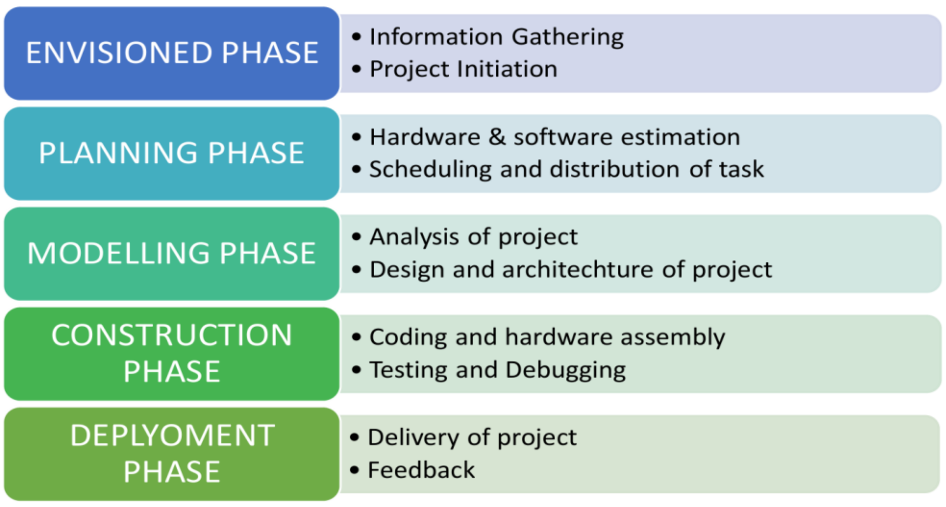


Figure 1. Model of phases in project management.

The project management strategy for the development of the Web-Controlled Surveillance Robot using Raspberry Pi is divided into five structured phases, ensuring a systematic and organized approach to planning, development, and delivery. These phases help monitor progress, allocate resources efficiently, and minimize risks throughout the project lifecycle.

1.5.1 **Envisioned Phase**

- Information Gathering: Research on existing surveillance systems, Raspberry Pi capabilities, and relevant technologies such as OpenCV, Flask, and wireless control systems.
- Project Initiation: Defining the project goals, scope, deliverables, and forming the project team with clear roles and responsibilities.

1.5.2 **Planning Phase**

- Hardware & Software Estimation: Identifying required hardware components (Raspberry Pi, motors, camera, motor driver, etc.) and software tools (Python, OpenCV, Flask, etc.).
- Task Scheduling and Distribution: Creating a timeline (e.g., Gantt chart), assigning tasks among team members, and setting milestones for each phase.

1.5.3 **Modelling Phase**

- Analysis of Project Requirements: Evaluating both functional (e.g., remote control, video streaming) and non-functional requirements (e.g., latency, scalability).
- Design and Architecture: Creating system architecture diagrams, circuit schematics, and web interface wireframes to guide implementation.

1.5.4 **Construction Phase**

- Coding and Hardware Assembly: Programming control logic for motors and camera; assembling hardware components onto the robot chassis.
- Testing and Debugging: Iterative testing of robot movement, web interface controls, and video streaming to identify and resolve bugs or connectivity issues.

1.5.5 **Deployment Phase**

- Delivery of Project: Final deployment of the robot in a real environment for demonstration and evaluation.
- Feedback: Collecting feedback from users and supervisors to assess system performance and identify areas for future improvement or expansion.

1.6. Overview and Benefits

Overview

This project involves the design and implementation of a web-controlled surveillance robot using Raspberry Pi for real-time remote monitoring. The system combines robotics, embedded systems, wireless communication, and web technologies to create a mobile platform capable of transmitting live video feeds and responding to user commands over a web interface.

The robot is equipped with a camera module connected to a Raspberry Pi, which serves as the central control unit. The system operates over a Wi-Fi network, allowing users to control the robot's movements (forward, backward, left, right) and view a live video stream using a standard web browser. The integration of Python, OpenCV, and Flask enables real-time control and streaming with minimal latency.

The project aims to offer a scalable, cost-effective solution for basic surveillance tasks in environments where mobility and remote access are critical.

Benefits

- **Real-Time Remote Monitoring**

  Enables users to stream live video and control the robot from a distance using any web browser, enhancing situational awareness and security.

- **Low-Cost and Open-Source**

  Utilizes affordable hardware (Raspberry Pi, basic motors, webcam) and free software tools (Python, OpenCV, Flask), making it budget-friendly and accessible.

- **Safe Operation in Risky Environments**

  Can be deployed in dangerous or hard-to-reach areas, reducing the need for human presence in unsafe locations (e.g., fire zones, tight spaces).

- **Educational and Skill-Building Tool**

Offers hands-on experience in programming, electronics, robotics, and networking—ideal for students and hobbyists.

- **Customizable and Expandable Design**

  Built with modular components, allowing for easy upgrades such as obstacle avoidance, sensors, or autonomous navigation.

Chapter 02: Background Review & Survey

2.1. Related Works

Over the years, various surveillance systems have been developed using embedded platforms, wireless communication, and robotics to enhance security and monitoring capabilities. This section explores related works and projects that share similarities in concept or technology with the proposed system.

(1) Mobile Surveillance Robots Using Arduino and Wi-Fi

Several projects have employed Arduino-based robots controlled over Wi-Fi networks. These systems often include basic directional movement and camera integration. However, due to Arduino's limited processing power, such systems typically require external modules for video streaming, making them less efficient for real-time applications.

(2) Raspberry Pi-Based Surveillance Systems

Raspberry Pi has been widely adopted in modern surveillance systems due to its computing capabilities, support for Python, and ability to handle video processing. For example, some implementations use Raspberry Pi with OpenCV for motion detection, video capture, and remote access through web servers. These systems form the foundation for creating more advanced and scalable surveillance solutions.

(3) IoT-Enabled Remote Monitoring Systems

Recent advancements in the Internet of Things (IoT) have enabled the integration of remote monitoring systems using cloud platforms, mobile apps, and embedded devices. These systems can transmit data in real time and offer features like alerts, image capture, and remote control. However, they often require more complex network configurations and cloud infrastructure.

(4) Smartphone-Controlled Robot Surveillance

Some related projects focus on controlling robots via smartphone apps over Bluetooth or Wi-Fi. While these systems offer convenience, they are often limited by range (in Bluetooth-based systems) and may not provide live video feeds or web-based control interfaces.

(5) Surveillance Drones and Autonomous Monitoring Robots

Higher-end solutions include drones or ground robots equipped with AI and autonomous navigation features. These systems are expensive and often used in military or industrial applications. They are beyond the scope of this project but represent the high end of surveillance technology evolution.

Summary

Most existing systems are either too basic (lacking real-time streaming and user-friendly interfaces) or too complex and expensive for personal or educational use. The proposed project aims to bridge this gap by developing a cost-effective, web-controlled surveillance robot using Raspberry Pi that supports real-time video and responsive control through a browser interface.

Chapter 03: Theoretical Aspects

3.1. Internet of Things (IoT)

The **Internet of Things (IoT)** refers to the network of physical objects—“things”—that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet or local networks. These objects can range from household devices and wearable technology to industrial machinery and smart vehicles.

In the context of this project, the surveillance robot functions as an **IoT-enabled device**. It collects real-time video data through its camera, processes user commands via the internet, and responds by executing physical movements. The integration of sensors, actuators, and connectivity allows the system to operate remotely and interactively.

3.2. Features of IoT

a) Intelligence

IoT devices are embedded with software and algorithms that allow them to perform smart functions such as data analysis, decision-making, and autonomous control. In the context of this project, the Raspberry Pi processes user commands and video data intelligently, allowing the robot to operate with minimal delay and high responsiveness.

b) Connectivity

Connectivity is central to IoT, enabling devices to communicate with each other and with external systems over networks. The surveillance robot uses Wi-Fi to connect the Raspberry Pi to a web interface, allowing remote control and real-time video streaming. This wireless connection transforms a basic robot into an IoT-enabled smart device.

c) Dynamic Nature

IoT systems are capable of adapting to changing environments and inputs. Devices can join or leave the network, update their status, and respond to new commands dynamically. The robot in this project responds in real-time to directional commands and continuously updates the video feed, illustrating this dynamic nature.

d) Enormous Scale

IoT supports the integration of a vast number of devices and systems across different environments. While this project involves a single robot, the architecture is scalable, allowing multiple robots or sensors to be added for more complex surveillance networks, such as in smart buildings or campuses.

e) Sensing

Sensing is a fundamental feature of IoT, enabling devices to detect changes in the environment and collect data. The surveillance robot uses a camera as a primary sensor to capture real-time video. Additional sensors (e.g., for temperature, motion, or distance) can be incorporated to enhance monitoring capabilities.

f) Heterogeneity

IoT systems support a wide range of devices with different hardware, software, and communication protocols. The use of Raspberry Pi, web browsers, Python, and Flask in this project demonstrates heterogeneity. Each component interacts seamlessly, enabling efficient system integration and performance.

g) Security

Security is critical in IoT systems to protect data, devices, and communications from unauthorized access or cyberattacks. Although the current project is for small-scale use, implementing password protection, network encryption, and secure coding practices is essential to ensure safe operation and prevent misuse.

3.3 Advantages of IoT

a) Communication

IoT enables seamless machine-to-machine (M2M) and machine-to-human (M2H) communication through wireless networks. In this project, commands from the user are transmitted to the robot via a web interface, and the robot responds with live video feedback. This continuous and real-time exchange of data improves coordination and responsiveness.

b) Automation and Control

IoT allows devices to be operated automatically or remotely with minimal human intervention. The surveillance robot responds to directional commands (forward, backward, left, right) and streams live video without requiring direct physical interaction, showcasing how automation simplifies surveillance tasks and increases user convenience.

c) Information

IoT systems gather, transmit, and display valuable data from connected devices. In this project, the camera mounted on the robot provides live visual information from the monitored area. This visual data helps users make informed decisions and enhances situational awareness, especially in inaccessible or hazardous locations.

d) Monitoring

One of the core strengths of IoT is remote monitoring. The system allows users to monitor specific environments in real time, ensuring security and safety without the need for physical presence. This is especially useful for home security, remote facilities, or restricted zones, where continuous human observation may not be practical.

e) Efficiency

IoT systems optimize time, resources, and effort by enabling faster communication and reducing the need for manual processes. The web-controlled surveillance robot increases efficiency by allowing remote control from any device with internet access, minimizing the need for onsite personnel and reducing response time in critical situations.

3.3. Disadvantages of IoT

a) Compatibility

IoT systems often involve a wide range of devices, platforms, and communication protocols. Ensuring that all components work seamlessly can be difficult. In this project, integrating the Raspberry Pi with various software libraries (Flask, OpenCV, GPIO, etc.) and ensuring compatibility with different browsers or devices can be challenging, especially during updates or scaling.

b) Complexity

Although IoT aims to simplify automation, the underlying system architecture can be quite complex. The combination of hardware, networking, and software components requires careful design and troubleshooting. Building a real-time, remotely accessible surveillance robot involves managing video streams, motor control, server deployment, and error handling—tasks that require technical expertise.

c) Privacy/Security

Security is one of the most critical concerns in IoT. Devices connected to networks can be vulnerable to hacking, unauthorized access, or data breaches. In this project, if not properly secured, the video feed or robot controls could be exploited by attackers. Protecting the system with authentication, encrypted communication, and secure coding practices is essential but often overlooked in small-scale implementations.

d) Safety

Malfunctions or unauthorized access to IoT systems can pose safety risks. For instance, if the surveillance robot is used in a sensitive environment and is accessed remotely without permission, it could lead to privacy violations or physical disruptions. Additionally, improper motor control could cause the robot to move unexpectedly or damage objects.

3.4. Application areas of IoT

The Internet of Things (IoT) has revolutionized numerous sectors by enabling smart connectivity, automation, and real-time data exchange. Its vast range of applications spans domestic, commercial, industrial, and governmental domains. Below are key application areas where IoT is making a significant impact:

Home Automation (Smart Homes)

IoT is widely used in creating intelligent home environments where devices such as lights, thermostats, security systems, and appliances can be controlled remotely. Surveillance robots like the one in this project can serve as mobile security units, monitoring different parts of a home and providing live video feeds to homeowners.

Healthcare and Remote Monitoring

In healthcare, IoT enables remote patient monitoring, smart diagnostic tools, and real-time data collection from wearable devices. Surveillance robots can be adapted for use in hospitals or care homes to observe patients without physical intrusion, reducing the risk of infection and improving patient safety.

Industrial Automation

Industries use IoT for predictive maintenance, real-time monitoring of machinery, and automation of production lines. Surveillance robots can patrol factory floors or hazardous zones to monitor equipment status, detect anomalies, and ensure operational safety without endangering human workers.

Smart Cities and Public Safety

IoT contributes to smart city development through traffic monitoring, environmental sensing, and public surveillance. Mobile surveillance robots can be deployed in public spaces to provide security coverage, crowd monitoring, and real-time incident alerts, supporting law enforcement and emergency response.

Agriculture and Environmental Monitoring

IoT is employed in precision agriculture to monitor soil conditions, crop health, and irrigation systems. Surveillance robots can assist in monitoring large farms or remote areas, capturing visual data to detect pest activity or monitor livestock movement.

Transportation and Logistics

IoT helps in fleet management, vehicle tracking, and cargo condition monitoring. Surveillance robots can be utilized in warehouses and logistics centers for security patrols, intruder detection, and inventory observation.

Military and Defense

In defense, IoT is used in surveillance, reconnaissance, and remote weapon systems. Robotic surveillance units can operate in potentially dangerous areas, gathering intelligence and enhancing situational awareness without risking human lives.

3.5. IOT Technologies and Protocols

IoT systems rely on various wireless technologies and communication protocols to connect devices, transfer data, and enable remote control. Each technology serves different needs in terms of range, power consumption, data rate, and application context. Below is an overview of key IoT communication technologies and protocols relevant to modern IoT deployments:

a) Bluetooth

Bluetooth is a short-range wireless technology designed for low-power, low-bandwidth communication between devices. It is commonly used in wearables, smart home devices, and health monitoring systems. While useful for local device pairing, its limited range and bandwidth make it less suitable for real-time video surveillance applications like this project.

b) Zigbee

Zigbee is a low-power, low-data-rate wireless communication protocol used in sensor networks and smart home systems. It supports mesh networking, making it scalable and reliable for device-to-device communication. However, Zigbee is not ideal for high-bandwidth applications like live video streaming due to its limited data capacity.

c) Z-Wave

Z-Wave is another wireless communication protocol used primarily in smart home automation. It is optimized for low latency and power consumption, supporting a mesh network topology. Like Zigbee, Z-Wave is well-suited for simple control applications but not for high-speed data transmission like video.

d) Wi-Fi

Wi-Fi is a widely used wireless technology that provides high-speed internet connectivity. It supports larger data transmission and is ideal for real-time applications such as video streaming and remote control. In this project, Wi-Fi is used to connect the Raspberry Pi to a web interface, enabling live video feed and robot control over a local network or the internet.

e) Cellular (3G/4G/5G)

Cellular networks offer wide-area coverage and are used for IoT applications requiring mobility or long-distance communication. Though more expensive and power-consuming than Wi-Fi, cellular technology is essential for remote monitoring systems deployed in areas without Wi-Fi access. It could be a future upgrade for extending the surveillance robot’s range.

f) NFC (Near Field Communication)

NFC is a short-range communication protocol used in access control, mobile payments, and identification systems. It enables quick, secure communication between devices in very close proximity (a few centimeters). NFC is not applicable for real-time surveillance or remote control but may be useful in user authentication scenarios.

g) LoRaWAN (Long Range Wide Area Network)

LoRaWAN is designed for long-range, low-power communication and is ideal for remote sensing and monitoring in rural or large-scale environments. It has low data rates and is unsuitable for transmitting video or real-time data but excels in transmitting small amounts of sensor data over kilometers.

3.6 Brief Description

This project involves the development of a **web-controlled surveillance robot** designed for real-time remote monitoring using **Raspberry Pi** as the core controller. The system combines the principles of **IoT (Internet of Things)**, **embedded systems**, and **wireless communication** to create a mobile robotic platform that can stream live video and respond to directional control via a web interface.

The robot consists of a chassis mounted with **DC motors**, a **Raspberry Pi board**, a **camera module**, and a **motor driver circuit**. The Raspberry Pi serves as the brain of the robot, handling both motion control and video processing. The live video captured by the camera is streamed to a web browser using Python and the Flask framework, while user inputs from the browser are transmitted back to the Raspberry Pi to control movement.

Communication between the robot and the control interface occurs over a **Wi-Fi network**, allowing the user to operate the robot remotely from a smartphone, tablet, or PC. This makes the system highly useful for surveillance tasks in indoor environments or within the Wi-Fi range in outdoor areas.

The goal of this system is to provide a **cost-effective**, **user-friendly**, and **flexible** surveillance solution that demonstrates the practical applications of IoT in real-world monitoring and security contexts.

Chapter 04: Hardware Requirements

4.1 Raspberry Pi Board

- **Model:** Raspberry Pi 4 Model B
- **Specifications:**
  - Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
  - RAM: 2GB / 4GB / 8GB LPDDR4-3200
  - Wireless: 802.11ac Wi-Fi and Bluetooth 5.0
  - USB ports for peripherals
  - GPIO pins for hardware interfacing
- **Function:** Acts as the central processing unit, managing data collection, processing, control commands, and communication between components.

4.2 Camera Module

- **Model:** Raspberry Pi Camera Module v2 or compatible USB webcam
- **Specifications:**
  - 8MP Sony IMX219 sensor
  - Supports 1080p video at 30fps
- **Function:** Captures high-resolution images and streams live video for surveillance purposes.

4.3 Power Supply

- **Options:**
  - 5V 3A USB-C Power Adapter
  - Lithium-Ion Battery Pack (10,000mAh or higher) for portability
- **Function:** Provides power to the Raspberry Pi and other components to ensure uninterrupted operation.

4.4 Motor Driver Module

- **Model:** L298N Dual H-Bridge Motor Driver.
- **Function:** Controls the rotation direction and speed of DC motors used in the robot's mobility system. It interfaces with the Raspberry Pi via GPIO pins.

4.5 DC Motors with Wheels

- **Specification:** 6V geared DC motors with rubber wheels
- **Function:** Facilitates movement of the robot, allowing it to patrol predefined areas or reposition for better surveillance.

4.6 Chassis

- **Material:** Acrylic / Aluminium
- **Function:** Serves as the structural framework, enabling secure mounting of all hardware components.

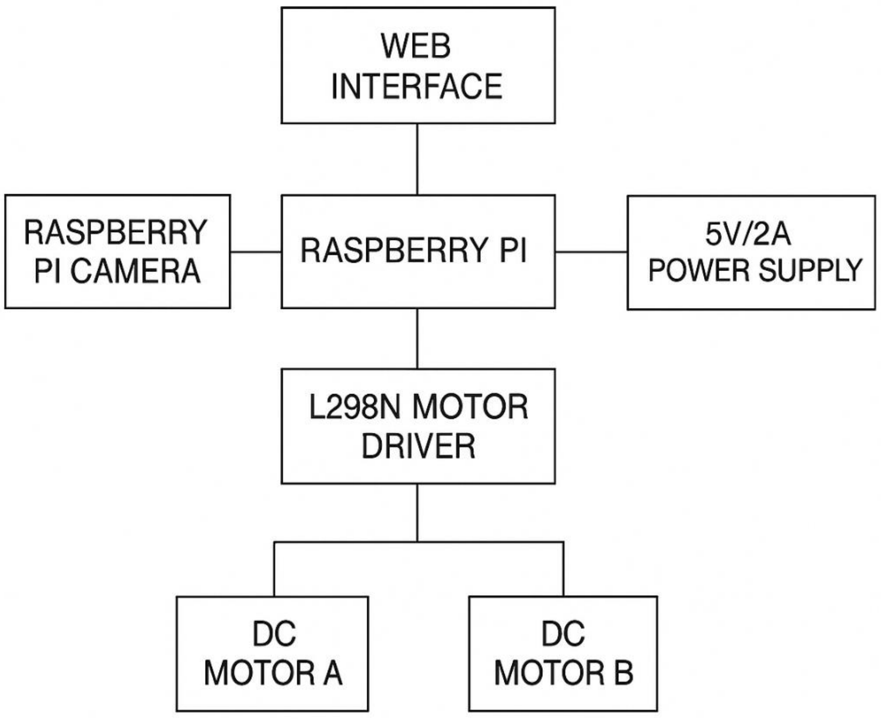
4.7 Wi-Fi Module

- **Type:** Built-in (Raspberry Pi 4) or external Wi-Fi dongle
- **Function:** Provides wireless connectivity for remote control and video streaming via the internet or local network.

4.8 MicroSD Card

- **Specification:** Class 10 MicroSD Card (16GB or higher)
- **Function:** Stores the Raspberry Pi OS and surveillance application scripts necessary for operation.

4.9 Block diagram of the proposed system

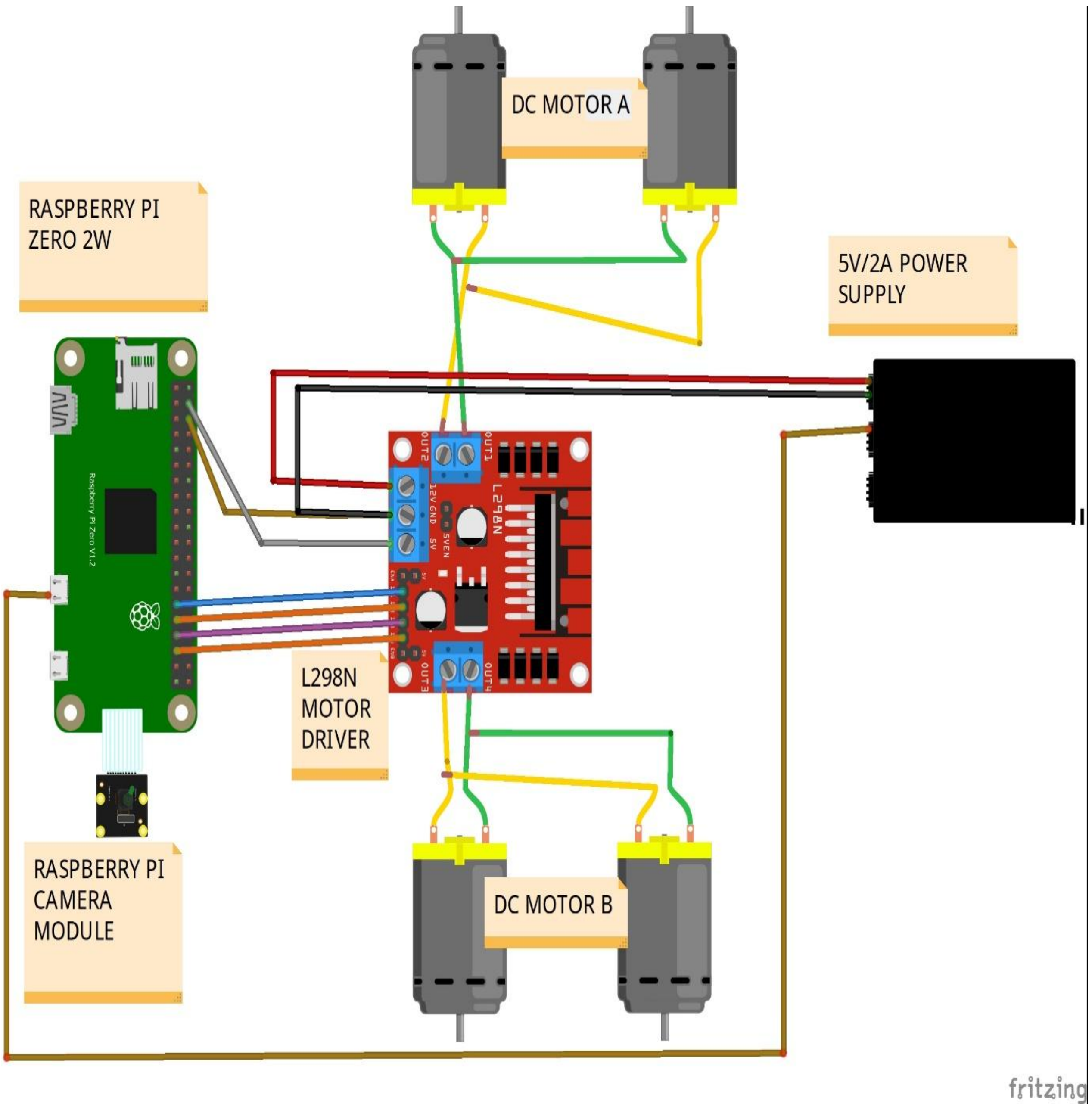


4.10 Working of the system

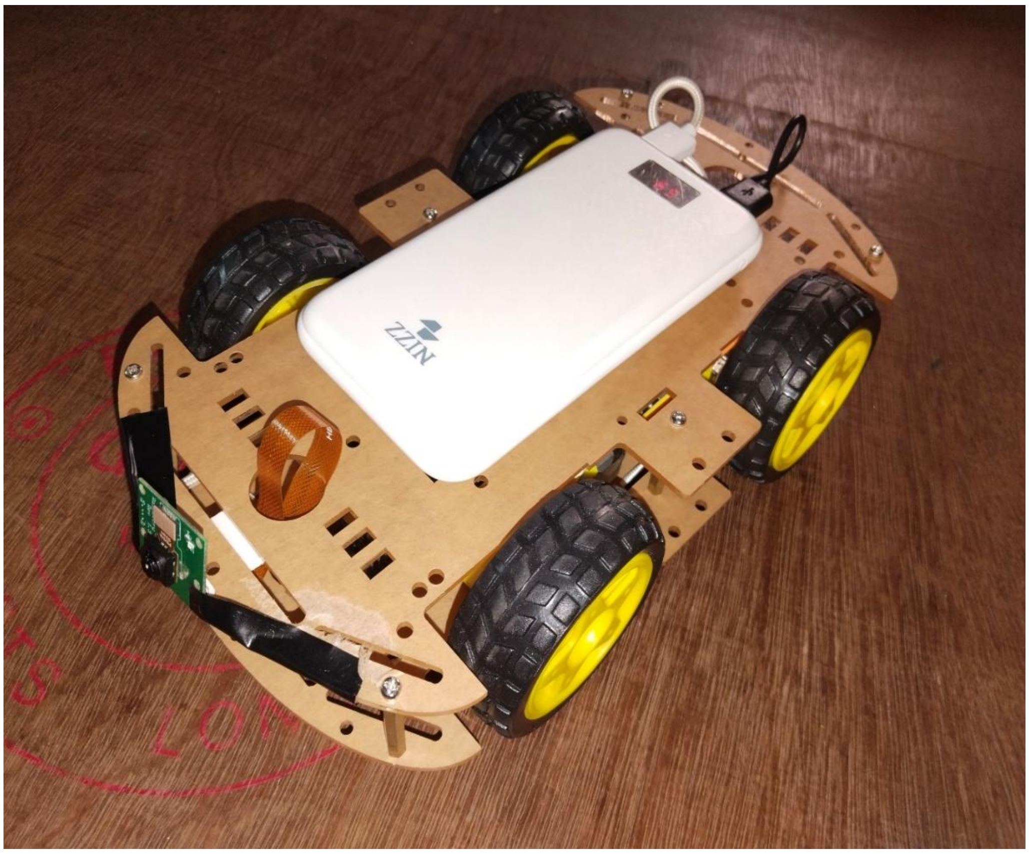
The surveillance robot is designed to monitor a specific area and provide real-time information to a remote user. Its working involves the coordination of several modules, including sensors, a camera, a microcontroller, and a communication system.

- **Power Supply:** The robot is powered using a battery or external supply, which energizes all the components.
- **Initialization:** Once powered on, the microcontroller (such as Arduino or Raspberry Pi) initializes all connected devices like motors, sensors, and the camera module.
- **Environment Sensing:** The robot uses obstacle detection sensors (e.g., ultrasonic or IR sensors) to detect objects or barriers in its path. This allows the robot to avoid collisions while navigating.
- **Surveillance and Monitoring:**
    - A camera module (such as an ESP32-CAM or USB webcam) captures live video footage of the surroundings.
    - This footage is transmitted wirelessly (via Wi-Fi, Bluetooth, or cellular network) to a remote device like a smartphone, PC, or cloud server for monitoring.
- **Mobility:** Based on sensor inputs and remote commands, the microcontroller sends signals to the motor driver, which controls the motors to move the robot in the desired direction.
- **Remote Control (optional):** The robot can be manually controlled through a mobile app or computer interface, or operate autonomously depending on the design.
- **Data Transmission:** All collected data (video, sensor readings) are transmitted in real-time, enabling the user to monitor the environment remotely and take necessary action if any unusual activity is detected.

4.11 Circuit Diagram



4.12 WORKING PROJECT MODEL :-



4.13 Components Required

Table 1. Component listing.

Sl. No.	Component and Specification	Quantity
1.	Microcontroller (e.g., Raspberry pi zero 2w/ ESP32)	1
2.	Camera Module (e.g., ESP32-CAM / USB Webcam)	1
3.	Motor Driver Module (e.g., L298N)	1
4.	DC Geared Motors (12V, 100 RPM)	2
5.	Power Supply (Li-ion Battery Pack, 12V)	1
6.	Chassis with Wheels	1 set
7.	Wi-Fi or Bluetooth Module (if not inbuilt)	1

Chapter 05: Software Requirements

5.1 Thonny IDE (Python)

- **Thonny IDE:**

Thonny is a lightweight and beginner-friendly Integrated Development Environment (IDE) primarily used for programming in Python. It is compatible with microcontrollers like the ESP32 and Raspberry Pi Pico, making it suitable for developing the control code.

- **Python:**

The main programming language used within Thonny IDE to write and upload scripts that control the robot’s sensors, motors, and communication modules.

- **Microcontroller Firmware:**

Appropriate firmware (e.g., MicroPython or CircuitPython) installed on the microcontroller to enable Python scripting.

- **Additional Libraries:**

Required Python libraries for handling camera, motor control, and wireless communication, such as machine, network, or custom motor driver libraries.

5.2 Logic and Flowchart

Logic:

- **Start**

Initialize all hardware components — microcontroller, motors, camera, and communication modules.

- **Check Sensors**

Continuously monitor sensor inputs (if any) or check for remote commands.

- **Capture Video**

Activate the camera module to capture live video.

- **Send Data**

Transmit the video feed and sensor data wirelessly to the remote monitoring device.

- **Movement Control**

Receive commands from the remote device and control motors accordingly to navigate the robot.

- **Obstacle Avoidance (optional)**

If sensors detect an obstacle, stop or change direction to avoid collision.

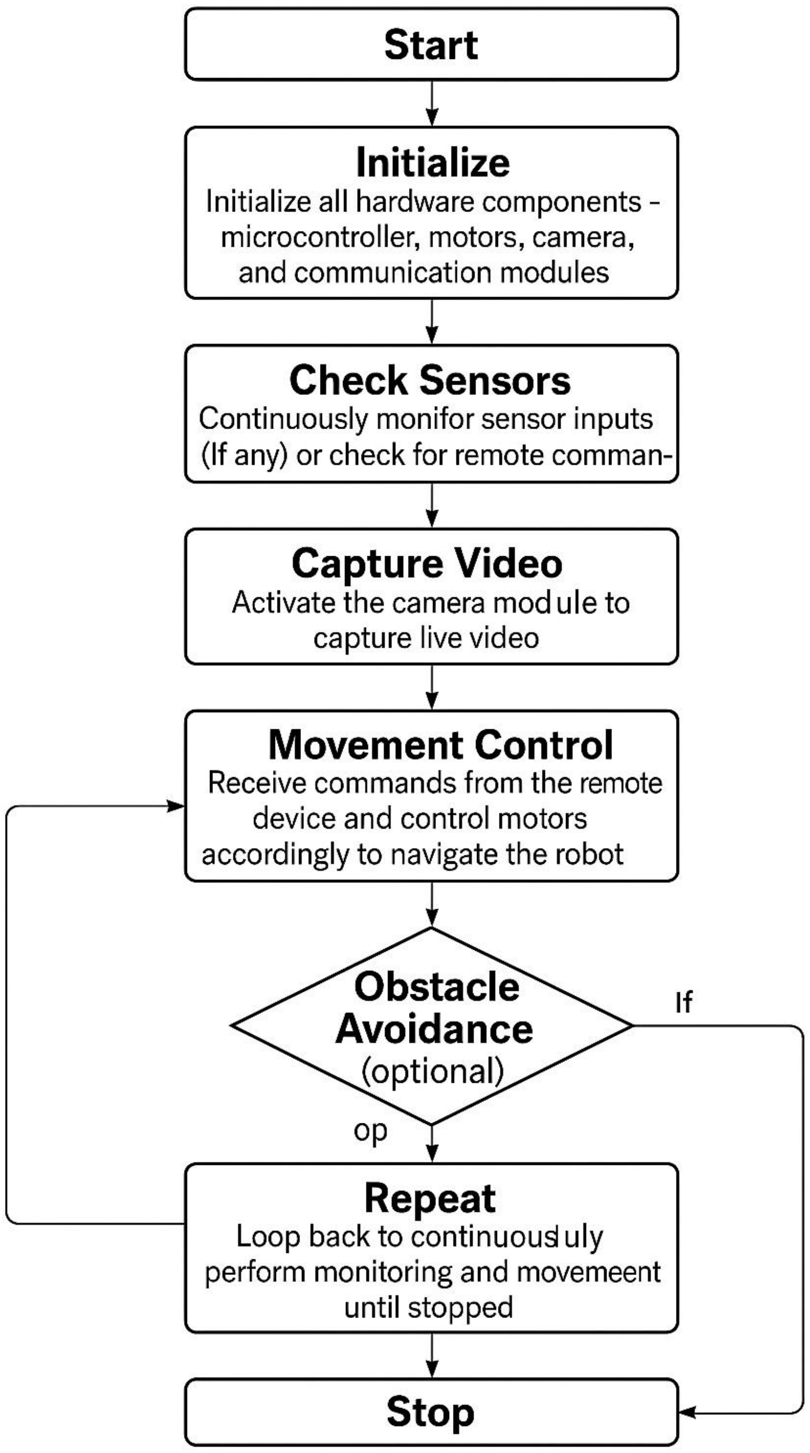
- **Repeat**

Loop back to continuously perform monitoring and movement until stopped.

- **Stop**

Safely shut down all modules.

Flowchart:


Start
Initialize
Initialize all hardware components - microcontroller, motors, camera, and communication modules
Check Sensors
Continuously monifor sensor inputs (If any) or check for remote comman-
Capture Video
Activate the camera mod ule to capture live video
Movement Control
Receive commands from the remote device and control motors accordingly to navigate the robot
Obstacle Avoidance (optional)
If
op
Repeat
Loop back to continuousl uly perform monitoring and movemeent until stopped
Stop

Chapter 06: Project development & Testing Aspects

6.1 Project Development

The development of the surveillance robot involved several stages:

- **Requirement Analysis:**

Identified the key features such as live video streaming, remote control, obstacle avoidance, and autonomous navigation.

- **Component Selection:**

  Chose suitable hardware like microcontroller (Arduino/ESP32), camera module, motor drivers, motors, power supply, and communication modules.

- **Circuit Design & Assembly:**

  Designed the circuit schematic and physically assembled the components on the robot chassis, ensuring proper wiring and connections.

- **Software Development:**

  Developed the control program using Python (with Thonny IDE) or Arduino IDE. The code includes motor control, video capture, wireless communication, and sensor data processing.

- **Integration:**
  Integrated hardware and software components, tested communication between modules, and ensured smooth operation.

6.2 Testing Aspects

- **Unit Testing:**

  Each component (motors, sensors, camera, communication) was tested independently to verify functionality.

  - **Integration Testing:**

Combined the components and tested them together for seamless operation and data flow.

- **Functional Testing:**

  Verified if the robot could move, avoid obstacles (if sensors are used), stream live video, and respond to remote commands correctly.

- **Field Testing:**

  Conducted real-world testing in the intended environment to check the robot's performance under actual conditions.
- **Performance Metrics:**

  Evaluated battery life, communication range, video quality, response time to commands, and obstacle detection accuracy.

- **Troubleshooting & Debugging:**

  Identified and fixed issues like connectivity drops, motor stalls, or camera lag.

6.3 Challenges Faced

- Managing real-time video streaming without lag.
- Ensuring stable wireless communication in different environments.
- Power management for prolonged operation.

Chapter 07: Conclusion & Future Scope

7.1 Result

The surveillance robot was successfully developed and tested. It was able to capture live video footage and transmit it wirelessly to a remote monitoring device. The robot demonstrated basic mobility, allowing remote control navigation. The system showed reliable performance during field tests with stable communication and effective real-time surveillance capabilities.

7.2 Conclusion

The project successfully achieved its objective of designing and implementing a functional surveillance robot. The integration of hardware components like the microcontroller, camera, motor driver, and communication modules was seamless. The robot can perform surveillance tasks effectively, making it suitable for applications such as security monitoring in homes, offices, or restricted areas. The use of user-friendly software tools like Thonny IDE helped simplify development and testing processes.

7.3 Limitations

- The robot lacks advanced obstacle detection and avoidance features due to the absence of sensors.
- Limited battery life restricts the duration of continuous operation.
- Video streaming quality and range depend on the wireless communication module's capability and environmental conditions.
- The robot requires manual control; it lacks autonomous navigation and decision-making.

7.4 Further Enhancement and Future Scope

- Integration of advanced sensors (e.g., ultrasonic, infrared, LIDAR) for improved obstacle avoidance and autonomous navigation.
- Implementation of AI-based algorithms for object detection and automatic threat recognition.
- Enhancement of power management with higher capacity batteries or solar charging.
- Development of a mobile app or web interface for more intuitive remote control and monitoring.
- Incorporation of night vision cameras for 24/7 surveillance capabilities.
- Adding cloud storage and data analytics features for storing and analyzing surveillance footage.

References

YouTube Tutorials:

- RaspberryPi-based surveillance robot tutorials: https://www.youtube.com/watch?v=akKH__TmEeY&list=P L-ZbNfIz7zVFArBDOCODU30EA4nvZ2-jf&index=8

Documentation & Research Papers:

- OpenCV official documentation:

  https://docs.opencv.org/
- Flask web development guide:

  https://flask.palletsprojects.com/
- MQTT protocol for IoT:

  https://mqtt.org/
- IEEE papers on IoT surveillance systems:

  https://ieeexplore.ieee.org/

Blogs & Articles:

- Raspberry Pi security camera setup:

  https://www.raspberrypi.org/blog/
- IoT-based surveillance solutions:

  https://www.iotforall.com/

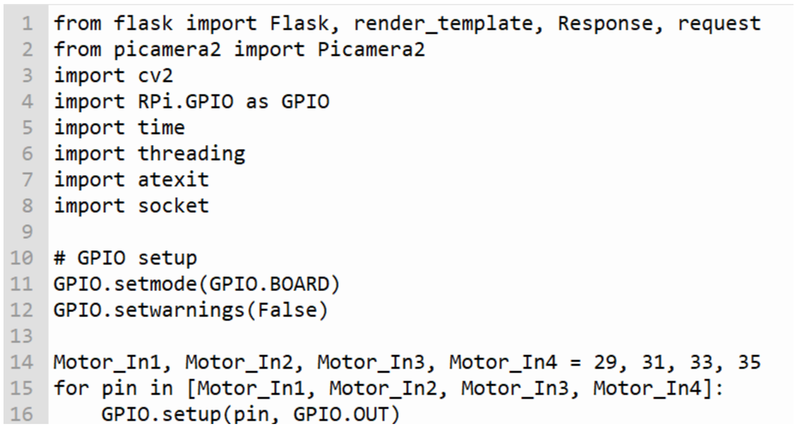
Appendix 01

A01.1. Code Listing

This appendix provides the complete source code used for developing the **Web-Controlled Surveillance Robot Using Raspberry Pi**, with minimal descriptions for each section.

Listing A01.1: Library Imports and GPIO Setup

This section imports required libraries and sets up GPIO pins for motor control.



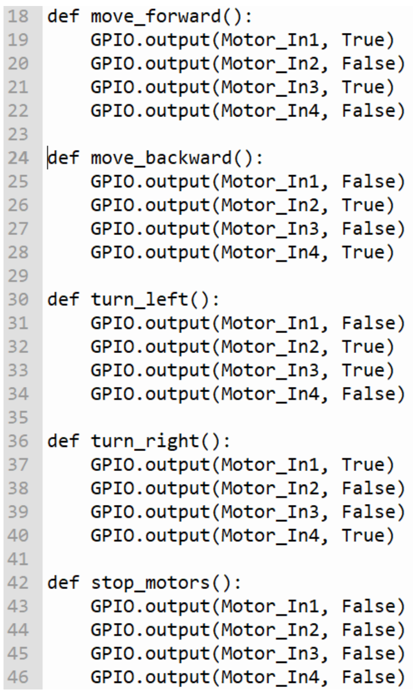
```python
from flask import Flask, render_template, Response, request
from picamera2 import Picamera2
import cv2
import RPi.GPIO as GPIO
import time
import threading
import atexit
import socket

# GPIO setup
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)

Motor_In1, Motor_In2, Motor_In3, Motor_In4 = 29, 31, 33, 35
for pin in [Motor_In1, Motor_In2, Motor_In3, Motor_In4]:
    GPIO.setup(pin, GPIO.OUT)
```

Listing A01.2: Motor Control Functions

Defines movement functions to control the robot's motors (forward, backward, left, right, stop).



```python
def move_forward():
    GPIO.output(Motor_In1, True)
    GPIO.output(Motor_In2, False)
    GPIO.output(Motor_In3, True)
    GPIO.output(Motor_In4, False)

def move_backward():
    GPIO.output(Motor_In1, False)
    GPIO.output(Motor_In2, True)
    GPIO.output(Motor_In3, False)
    GPIO.output(Motor_In4, True)

def turn_left():
    GPIO.output(Motor_In1, False)
    GPIO.output(Motor_In2, True)
    GPIO.output(Motor_In3, True)
    GPIO.output(Motor_In4, False)

def turn_right():
    GPIO.output(Motor_In1, True)
    GPIO.output(Motor_In2, False)
    GPIO.output(Motor_In3, False)
    GPIO.output(Motor_In4, True)

def stop_motors():
    GPIO.output(Motor_In1, False)
    GPIO.output(Motor_In2, False)
    GPIO.output(Motor_In3, False)
    GPIO.output(Motor_In4, False)
```

Listing A01.3: Cleanup and IP Retrieval

Performs safe shutdown of GPIO and retrieves the local IP address of the Raspberry Pi.

```python
@atexit.register
def cleanup():
    stop_motors()
    GPIO.cleanup()

# Get local IP address for display
def get_ip():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        ip = s.getsockname()[0]
        s.close()
        return ip
    except:
        return "127.0.0.1"
```

Listing A01.4: Camera Initialization and Frame Capture

Initializes the PiCamera2 and captures video frames continuously using a background thread.

```python
# Camera & threading
picam2 = Picamera2()
picam2.configure(picam2.create_video_configuration(
    main={"size": (320, 240)},
    buffer_count=3,
    controls={"FrameDurationLimits": (33333, 33333)}  # ~30 FPS
))
picam2.start()

frame_lock = threading.Lock()
latest_frame = None

def update_frames():
    global latest_frame
    while True:
        try:
            frame = picam2.capture_array()
            frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
            ret, buffer = cv2.imencode('.jpg', frame_bgr)
            with frame_lock:
                latest_frame = buffer.tobytes()
        except Exception as e:
            print("Frame capture error:", e)
        time.sleep(0.03)  # ~30 FPS

# Start the thread
frame_thread = threading.Thread(target=update_frames, daemon=True)
frame_thread.start()
```

Listing A01.5: Flask Web Server Setup

Initializes the Flask web application and defines the live video stream endpoint.

```python
# Flask app
Url_Address = get_ip()
app = Flask(__name__)

@app.route('/video_feed')
def video_feed():
    def generate():
        while True:
            with frame_lock:
                if latest_frame:
                    yield (b'--frame\r\n'
                           b'Content-Type: image/jpeg\r\n\r\n' + latest_frame + b'\r\n')
            time.sleep(0.03)
    return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/')
def index():
    return render_template("temp.html", HTML_address=Url_Address)
```

Listing A01.6: Motor Control Endpoints

Defines web routes to control the robot using HTML buttons (forward, backward, left, right, stop).

```python
@app.route('/Forward', methods=['POST'])
def forward():
    move_forward()
    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/Backward', methods=['POST'])
def backward():
    move_backward()
    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/left', methods=['POST'])
def left():
    turn_left()
    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/right', methods=['POST'])
def right():
    turn_right()
    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/stop', methods=['POST'])
def stop():
    stop_motors()
    return render_template("temp.html", HTML_address=Url_Address)
```

Listing A01.7: Run the Application

Starts the Flask app and makes it accessible on the network.

```python
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, threaded=True)

```

A01.2. Main Code

```
from flask import Flask, render_template, Response, request

from picamera2 import Picamera2

import cv2

import RPi.GPIO as GPIO

import time

import threading

import atexit

import socket

# GPIO setup

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

Motor_In1, Motor_In2, Motor_In3, Motor_In4 = 29, 31, 33, 35

for pin in [Motor_In1, Motor_In2, Motor_In3, Motor_In4]:

    GPIO.setup(pin, GPIO.OUT)

def move_forward():

    GPIO.output(Motor_In1, True)

    GPIO.output(Motor_In2, False)

    GPIO.output(Motor_In3, True)

    GPIO.output(Motor_In4, False)

def move_backward():

    GPIO.output(Motor_In1, False)

    GPIO.output(Motor_In2, True)

    GPIO.output(Motor_In3, False)

    GPIO.output(Motor_In4, True)
```

```
def turn_left():
    GPIO.output(Motor_In1, False)
    GPIO.output(Motor_In2, True)
    GPIO.output(Motor_In3, True)
    GPIO.output(Motor_In4, False)
def turn_right():
    GPIO.output(Motor_In1, True)
    GPIO.output(Motor_In2, False)
    GPIO.output(Motor_In3, False)
    GPIO.output(Motor_In4, True)
def stop_motors():
    GPIO.output(Motor_In1, False)
    GPIO.output(Motor_In2, False)
    GPIO.output(Motor_In3, False)
    GPIO.output(Motor_In4, False)
@atexit.register
def cleanup():
    stop_motors()
    GPIO.cleanup()
# Get local IP address for display
def get_ip():
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        ip = s.getsockname()[0]
        s.close()
        return ip
    except:
        return "127.0.0.1"
# Camera & threading
picam2 = Picamera2()
picam2.configure(picam2.create_video_configuration(
    main={"size": (320, 240)},
    buffer_count=3,
    controls={"FrameDurationLimits": (33333, 33333)}  # ~30 FPS))
picam2.start()
frame_lock = threading.Lock()
latest_frame = None
def update_frames():
    global latest_frame
    while True:
        try:
            frame = picam2.capture_array()
            frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
            ret, buffer = cv2.imencode('.jpg', frame_bgr)
            with frame_lock:
                latest_frame = buffer.tobytes()
        except Exception as e:
            print("Frame capture error:", e)
        time.sleep(0.03)  # ~30 FPS
# Start the thread
frame_thread = threading.Thread(target=update_frames, daemon=True)
```

```
frame_thread.start()

# Flask app

Url_Address = get_ip()

app = Flask(__name__)

@app.route('/video_feed')

def video_feed():

    def generate():

        while True:

            with frame_lock:

                if latest_frame:

                    yield (b'--frame\r\n'

                           b'Content-Type: image/jpeg\r\n\r\n' + latest_frame + b'\r\n')

            time.sleep(0.03)

    return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/')

def index():

    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/Forward', methods=['POST'])

def forward():

    move_forward()

    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/Backward', methods=['POST'])

def backward():

    move_backward()

    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/left', methods=['POST'])
```

```
def left():

    turn_left()

    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/right', methods=['POST'])

def right():

    turn_right()

    return render_template("temp.html", HTML_address=Url_Address)

@app.route('/stop', methods=['POST'])

def stop():

    stop_motors()

    return render_template("temp.html", HTML_address=Url_Address)

if __name__ == '__main__':

    app.run(host="0.0.0.0", port=8080, threaded=True)
```

A01.3. Libraries

Below are the Python libraries used in this project, each serving a specific role in enabling web control, video streaming, and motor operations:

- **flask**: A lightweight web framework used to create the web interface for controlling the robot and streaming live video.
- **render_template, Response, request** *(from Flask)*:
    - render_template is used to render HTML pages.
    - Response helps stream the video feed.
    - request is used to handle form/button submissions from the UI.
- **picamera2**: A camera control library that allows capturing frames from the Raspberry Pi Camera Module in real time.
- **cv2 (OpenCV)**: Used to convert camera frames from RGB to BGR and encode them as JPEG images for streaming over the web.
- **RPi.GPIO**: Provides functions to configure and control the GPIO pins on the Raspberry Pi, enabling motor movements like forward, backward, left, and right.
- **time**: Allows adding delays, such as regulating frame rates and motor control durations.
- **threading**: Enables concurrent execution—used here to run the video frame capture loop without blocking the web server.
- **atexit**: Ensures GPIO pins are properly turned off and cleaned up when the program is terminated.
- **socket**: Retrieves the local IP address of the Raspberry Pi, which is then displayed on the web interface for user access.

Appendix 02

A02.1. Project Proposal Form

The project proposal form was prepared and duly signed from our Faculty-in-Charge Dr. Biswaranjan Swain. The same is attached at the last of this report.

A02.2. Project Management

#	Component	Individual Contributions in %			Total
		Aditya Mallick	Kshitij Prasad	Sarvesh Prasad	In %
1.	Planning	40%	30%	30%	100%
2.	Background Research and Analysis	35%	30%	35%	100%
3.	Hardware design	25%	50%	25%	100%
4.	Software design	50%	25%	25%	100%
5.	Testing	35%	30%	35%	100%
6.	Final Assembling	35%	50%	35%	100%
7.	Project report writing	20%	20%	60%	100%
8.	Presentation	40%	30%	30%	100%
9.	Logistics	40%	40%	10%	100%

A02.3. Bill of Material

Table 1. Component listing.

#	Component	Specification	Unit Cost	Quantity	Total
1.	Raspberry Pi Zero 2 W with cable and adapter	Quad-core 1GHz, Wi-Fi onboard	2750	1	2750
2.	Pi Camera Module	Official Raspberry Pi Camera (v2.1 or mini)	300	1	300
3.	Motor Driver Module	L298N (for Pi compatibility)	150	1	150
4.	DC Gear Motors + Wheels	100 RPM geared motors with rubber wheels	50	4	200
5.	Chassis	4-wheel acrylic chassis with mounts	300	1	300
6.	Power Bank/Battery Pack	5V 2A USB Output (for Pi) + 12V for motors	600	1	600
7.	Jumper Wires & Connectors	Male-female, female-female + GPIO header pins	100	2	200

Grand Total	Rs4500

Appendix 03

A03.1. Data Sheets

#	Component	Key Specifications	Datasheet / Reference Link
1	**Raspberry Pi Zero 2 W**	Broadcom BCM2710A1, Quad-core Cortex-A53 (1 GHz), 512MB RAM, Wi-Fi 2.4GHz	Raspberry Pi Zero 2 W Datasheet (PDF)
2	**Pi Camera Module**	8MP Sony IMX219 sensor, 3280×2464 resolution, CSI interface	Raspberry Pi Camera Module v2 Datasheet
3	**Motor Driver Module (L298N)**	Dual H-Bridge, supports 2 DC motors, 5–35V input	L298N Datasheet (STMicroelectronics)
4	**DC Gear Motors + Wheels**	100 RPM, 12V geared motors, 6mm shaft, 0.5–1.5kg torque	Sample Product Page (Robu.in)
5	**Chassis**	4-wheel acrylic platform, motor mount brackets, screw kits	Example Kit Reference
6	**Power Bank/Battery Pack**	5V 2A USB output (for Pi), separate 12V line for motor driver	Generic Product Spec (Example)
7	**Jumper Wires & Connectors**	Male-to-male, female-to-female, 2.54mm pitch headers	Jumper Wires Details