International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

AI Powered College Security Surveillance System

Kalpana G[1], Karthick B[2], Karun C[3], Dharun P[4]

[1]Assistant Professor, Artificial Intelligence and Data Science
[2,3,4]Student, Artificial Intelligence and Data Science

Abstract

This research focuses on developing an AI-powered security surveillance monitoring system specifically designed for college campuses. With increasing safety concerns in educational environments, traditional security methods such as manual patrols and standard CCTV monitoring often prove inadequate. The proposed system integrates advanced technologies including AI-based facial recognition, helmet and weapon detection, and real-time alert mechanisms using IoT devices like ESP32 microcontrollers, buzzers, and smart cameras. By automating threat detection and improving response times, the system ensures a safer, smarter, and more responsive campus environment. The project also emphasizes cost-efficiency, privacy, and scalability to support future campus security needs

CHAPTER 1 INTRODUCTION

In today's rapidly evolving world, the safety and security of educational institutions have become a top priority. Colleges, being vibrant hubs of student activity and intellectual growth, require robust security measures to ensure the safety of students, faculty, and staff. A well-designed Security Surveillance Monitoring System plays a crucial role in maintaining a secure environment by deterring unauthorized activities, detecting incidents in real-time, and enabling quick response actions. This project proposes the development of a comprehensive surveillance system tailored specifically for college campuses. By integrating high-definition cameras, motion detectors, and centralized monitoring software, the system aims to provide continuous observation of key areas such as entrances, classrooms, corridors, parking lots, and common gathering spaces. Additionally, with the use of smart technologies like AI-based motion detection and remote monitoring, the system enhances efficiency and reduces the need for manual patrolling. The College Security Surveillance Monitoring System not only helps prevent incidents such as vandalism, theft, and unauthorized access but also contributes to building a sense of safety and trust among students and staff. Through this project, we aim to demonstrate how modern surveillance solutions can be effectively implemented to create a safer and more responsive educational environment. The College Security Surveillance Monitoring System not only helps prevent incidents such as vandalism, theft, and unauthorized access but also contributes to building a sense of safety and trust among students and staff.

1.1 MOTIVATION FOR WORK

The need for security in educational institutions has never been greater. College campuses are open environments where thousands of students, staff, and visitors move daily, creating a challenging space to monitor. Traditional security measures like manual patrolling and basic gatekeeping are no longer sufficient to ensure full safety. These methods are prone to human error and delayed responses during emergencies. Increasing incidents of theft, vandalism, harassment, and unauthorized access have raised serious concerns about campus security. The motivation for developing a College Security Surveillance Monitoring System stems from the need to address these security gaps through technology-driven solutions. An automated surveillance system offers continuous, real-time monitoring of critical campus zones such as entrances, exits, hallways, classrooms, libraries, parking areas, and hostels. By integrating modern technologies such as high-definition cameras, motion detectors, and remote monitoring software, the system ensures that suspicious activities are detected early and reported instantly. With the advancement of AI and intelligent analytics, it is now possible to implement smart alert systems that can detect unusual behavior and automatically notify the concerned authorities. The project is motivated by a vision to apply technological innovation to strengthen campus safety standards. It aims to build a responsive, reliable, and efficient security system that not only addresses existing challenges but also prepares for future risks.

1.2 PROBLEM STATEMENT

Ensuring the safety and security of students, faculty, staff, and campus property has become a top priority for colleges and universities in today’s world. Traditional security measures such as manual patrolling, limited CCTV systems, and isolated alarm systems often fall short when dealing with the vastness and complexity of modern educational institutions. These approaches may suffer from delayed response times, insufficient coverage of critical areas, blind spots, human error, and lack of real-time incident reporting. In emergencies like unauthorized access, theft, vandalism, violence, or health crises, the time taken to detect and respond can significantly affect outcomes. Moreover, the absence of centralized, intelligent surveillance monitoring makes it difficult to maintain a secure yet open campus environment. Colleges often operate across multiple buildings, libraries, laboratories, sports complexes, hostels, and open spaces, making it challenging to maintain consistent, 24/7 surveillance without a robust, integrated system. With the advancement of technologies such as IoT, AI-based video analytics, and wireless networks, there is a significant opportunity to modernize campus security systems. The ultimate goal is to build a smart, connected surveillance ecosystem that not only monitors but actively enhances the security of the entire college campus, promoting a safe environment conducive to learning and development. By doing so, colleges can protect valuable human and infrastructural assets, build trust among students and parents, and set a benchmark for future-ready educational institutions.

1.3 OBJECTIVES

Enhance Campus Security

Implement a real-time surveillance system that ensures 24/7 monitoring of all critical areas across the college campus.

Centralized Monitoring

Develop a centralized control room that receives live image feeds and alert from all surveillance point to enable quicker response to incidents

Use Of Smart Technologies

Integrated AI Driven technologies like facial recognition, motion detection, automated thread analysis for proactive monitoring

Minimize Human Error

Reduce dependency on manual surveillance through automated alert system, improving overall reliability and consistence

Ensure Comprehensive Coverage

Eliminate blind spot by strategically placing high- resolution camera across academic buildings, hostels, library, labs, parking lots and open areas

Real Time Alert and Notification

Design a system that provides instant mobile or desktop alerts to security personal during any unauthorized activity or emergency

CHAPTER 2 LITERATURE SURVEY

2.1 LITERATURE SURVEY

In recent years, college campuses have increasingly adopted advanced surveillance technologies to enhance campus security and safety. The literature surrounding surveillance monitoring systems highlights a shift from traditional CCTV-based security towards intelligent, integrated systems powered by IoT, artificial intelligence (AI), and cloud computing.

In a study by Smith et al. (2017), it was found that traditional CCTV surveillance systems used in educational institutions are highly dependent on human monitoring and manual response mechanisms. The study revealed that manual surveillance often led to significant delays in identifying and responding to incidents on campus. Blind spots, limited camera ranges, and inefficient monitoring setups made it difficult to maintain complete security. Smith et al. highlighted the need for automation and real-time alert mechanisms to overcome human limitations. They proposed that the integration of automated detection technologies could greatly reduce response times. The study also noted that mere video recording was insufficient for proactive security management. Instead, live monitoring combined with intelligent detection was necessary for modern campus safety needs. The researchers emphasized the importance of high-quality video analytics to assist security personnel. They also suggested periodic audits of surveillance blind spots. Overall, the study underscored the necessity of upgrading traditional systems with smarter, more responsive technologies.

In a study by Kumar and Sharma (2019), the role of IoT in campus security systems was extensively analyzed. The research showed that IoT-enabled surveillance significantly enhances real-time monitoring capabilities by connecting cameras, sensors, and alarms into a unified network. They demonstrated that using IoT devices allows security staff to receive instant notifications of abnormal activities. The study also showed that IoT integration helps reduce manual errors and improves the efficiency of patrol teams. Furthermore, IoT-based systems were found to offer easy scalability, enabling colleges to expand their security networks as needed without overhauling the entire infrastructure. The study discussed how real- time data collection from IoT devices could be used for predictive analytics, further strengthening security. Challenges such as network reliability and data privacy concerns were acknowledged, but the overall results supported IoT adoption. The study concluded that IoT-based surveillance represents a critical step toward smarter, safer campuses, provided that security and maintenance protocols are strong.

In a study by Patel et al. (2020), the application of Artificial Intelligence (AI) and Machine Learning (ML) in surveillance systems was explored, particularly for educational environments. Their research demonstrated that AI-powered video analytics could automatically detect suspicious behavior, recognize faces, and monitor crowd density. Patel and colleagues showed that using ML models greatly reduced the workload on human operators. Their study included case tests where AI detected thefts and unauthorized access faster than traditional human monitoring. Moreover, the AI system could identify unusual patterns, like unattended bags or fights, that may not trigger traditional alarm systems. The researchers emphasized that machine learning improves over time, enhancing the accuracy of threat detection. However, they cautioned about the risk of false positives and the need for human oversight. The study suggested that AI should be used as an assistant rather than a full replacement for human security personnel. Overall, Patel et al. concluded that AI dramatically improves the effectiveness and responsiveness of surveillance systems on college campuses.

In a study by Wang and Li (2021), the advantages of cloud-based surveillance systems were comprehensively analyzed for college campuses. Their research explained how cloud technology allows educational institutions to manage large volumes of video data more effectively. Unlike traditional local storage, cloud platforms provide scalable storage solutions and advanced video processing capabilities. Wang and Li demonstrated that cloud integration reduces hardware costs and improves disaster recovery options. Their study also explored how cloud systems can support AI-driven analytics without heavy on-site computing requirements. Security concerns such as data breaches and unauthorized access were discussed, along with mitigation strategies like encryption and multi-factor authentication. The researchers found that cloud systems made remote monitoring across campuses much easier. Additionally, cloud platforms enabled collaboration between campus security teams and local law enforcement. The study concluded that cloud-based surveillance is highly beneficial for colleges aiming for a flexible, scalable, and technologically advanced security solution.

1) In a study by Rahman and Gupta (2018), privacy concerns associated with modern surveillance systems were critically examined. They argued that while surveillance enhances campus security, it also risks infringing upon individual privacy rights if not carefully managed. Their research highlighted cases where misuse of facial recognition technology led to ethical controversies. Rahman and Gupta proposed that colleges must adopt strict data access controls and transparency policies to protect students' privacy. The study discussed the importance of anonymizing non-critical footage and setting clear retention periods for stored video data. Furthermore, they recommended forming ethics committees to oversee surveillance practices. The researchers emphasized the need for student awareness campaigns about how surveillance data is collected and used. They also suggested involving student bodies in discussions about surveillance policy-making. The study concluded that privacy protection is essential to maintaining trust and ensuring the long-term acceptance of surveillance technologies on campuses.

In a study by Tanaka et al. (2020), the energy efficiency of surveillance systems was explored with a focus on sustainable campus security solutions. They highlighted that traditional surveillance systems consume significant amounts of electricity, contributing to operational costs and environmental impacts. Tanaka and his team proposed the use of solar-powered CCTV cameras and energy-efficient networking devices. Their study showed that implementing smart power management techniques like sleep modes and dynamic resolution adjustment could reduce power consumption by up to 40%. They also emphasized the importance of using low-power processors in surveillance devices. Through pilot projects in several universities, they demonstrated that sustainable surveillance systems could maintain 24/7 monitoring without compromising performance. The research recommended hybrid models combining solar and grid power for reliability. The study concluded that adopting green technologies in security systems aligns with universities' broader goals of sustainability and carbon footprint reduction.

In a study by Chowdhury and Hassan (2018), the effectiveness of facial recognition technologies in campus surveillance was assessed. They explored the accuracy rates of different facial recognition models when deployed in real-world college environments. Their study found that factors like lighting conditions, camera angles, and image resolution significantly affect recognition performance.

Chowdhury and Hassan proposed a hybrid system that combined facial recognition with RFID-based ID systems to enhance accuracy. They also warned about the risks of bias in facial recognition algorithms, especially among diverse student populations. Their research emphasized the need for regular model retraining and validation to maintain fairness and effectiveness. They recommended using facial recognition only for high-security areas like examination halls or administrative offices. The study concluded that while facial recognition enhances access control, it must be implemented with caution and ethical oversight to avoid discrimination and privacy violations.

In a study by Fernandez et al. (2019), the role of real-time incident detection in modern campus security systems was highlighted. They explored the use of advanced video analytics to detect violent activities, such as fights or sudden crowd formations, in public campus areas. Fernandez and colleagues tested AI models trained to recognize patterns of aggressive movement and abnormal behavior. Their system was capable of sending immediate alerts to campus security offices within seconds of detecting a potential threat. The study also included simulations showing how real-time alerts could reduce the average incident response time by over 50%. They emphasized that real-time monitoring not only prevents incidents but also creates a safer campus atmosphere. However, the study pointed out the need for continuous updating of detection algorithms to stay effective. The researchers concluded that real-time incident detection is a critical feature for next-generation campus surveillance networks.

In a study by Nguyen and Park (2020), the impact of wireless surveillance networks on campus security systems was analyzed. They discussed how Wi-Fi and 5G technologies enable the deployment of flexible, cost-effective, and scalable camera networks. Nguyen and Park showed that wireless cameras are easier to install and relocate compared to wired systems, making them ideal for dynamic environments like college campuses. Their study explored challenges such as network latency, signal interference, and bandwidth congestion. They proposed solutions like mesh networking and adaptive bitrate streaming to ensure video quality. The research also emphasized the importance of securing wireless transmissions using encryption protocols like WPA3 to prevent hacking. Pilot deployments demonstrated successful large-area coverage with minimal downtime. The study concluded that wireless surveillance systems offer significant benefits in terms of flexibility and scalability, provided that network security and stability are properly managed.

In a study by Alvarez and Singh (2021), the integration of biometric access control with campus surveillance systems was investigated. They explored how combining fingerprint, iris, or facial recognition with surveillance cameras could strengthen security in sensitive areas such as server rooms, libraries, and laboratories. Alvarez and Singh's research found that biometric systems offer higher security compared to traditional ID cards or passwords, which can be stolen or shared. Their study emphasized the importance of multi-factor authentication (MFA) to further enhance security. They also examined the challenges of biometric system implementation, such as environmental sensitivity and user acceptance. Pilot studies showed that integrating biometrics with surveillance significantly reduced unauthorized access incidents. They stressed the need for secure biometric data storage and compliance with privacy regulations like GDPR. The study concluded that biometric-integrated surveillance offers a robust, future-proof solution for securing critical campus facilities.

In a study by Verma and Das (2021), the implementation of drone surveillance for campus security was explored. They analyzed how drones equipped with cameras can offer dynamic, aerial surveillance across large campus areas, including sports grounds and parking lots. Verma and Das found that drones could cover blind spots that fixed CCTV systems miss, especially during large events or emergencies. Their study discussed drone patrol scheduling, battery limitations, and integration with ground-based surveillance. They also highlighted legal and privacy concerns, emphasizing that drone surveillance must comply with aviation and privacy regulations. Through case studies, they demonstrated that drones significantly enhanced real-time monitoring capabilities and improved emergency response times. The study concluded that drone surveillance, when properly managed, is a valuable addition to the campus security ecosystem, providing flexibility and rapid area coverage.

In a study by Ahmed and Zhou (2019), the importance of integrating fire detection systems with campus surveillance was examined. Their research found that video-based smoke and fire detection systems could detect incidents much faster than traditional smoke alarms. Ahmed and Zhou proposed combining thermal imaging cameras with regular surveillance systems to identify fires at early stages. Their study showed successful detection of small fires in laboratories and cafeterias within seconds. They emphasized that early detection not only prevents property damage but also saves lives by triggering faster evacuations. Challenges discussed included false alarms caused by steam or lighting reflections. The study recommended integrating AI models to differentiate between real fire and false positives. They concluded that merging fire detection with video surveillance creates a safer and smarter campus environment.

In a study by Silva et al. (2022), the need for cybersecurity in modern surveillance networks was critically analyzed. Silva and colleagues pointed out that as surveillance systems become more interconnected and reliant on IP networks, they are increasingly vulnerable to cyber-attacks. Their research highlighted common vulnerabilities such as weak camera passwords, unencrypted video streams, and outdated firmware. Silva et al. proposed a layered security model including firewalls, VPNs, regular patch updates, and intrusion detection systems. Case studies revealed instances where hackers accessed unsecured cameras to spy on campus activities. The study stressed that cybersecurity should be considered from the design phase of any surveillance project. It concluded that strong cybersecurity measures are essential to protect the integrity and confidentiality of campus surveillance systems.

In a study by Park and Kim (2020), the application of crowd monitoring and density analysis in college events was investigated. They developed a surveillance model capable of counting people and estimating crowd density using AI-powered video analysis. Their system was tested during large campus events such as convocations and festivals. Park and Kim demonstrated that real-time crowd monitoring helps in preventing stampedes and ensuring smooth evacuation during emergencies. The system could send automatic alerts when crowd density exceeded safe thresholds. They also discussed how such systems could help in managing entry and exit points more efficiently. The study emphasized that integrating crowd monitoring with surveillance boosts safety, especially in high-risk scenarios. The researchers concluded that smart crowd management tools are a vital component of modern campus security solutions.

In a study by Rodriguez and Patel (2019), the concept of predictive policing through surveillance analytics was introduced. They explored how historical surveillance data could be analyzed to predict high-risk zones and peak times for security threats on campuses. Rodriguez and Patel’s system used machine learning to study patterns such as unauthorized entries, thefts, and altercations. Their model successfully predicted likely trouble spots, allowing campus security teams to deploy resources more strategically. The study discussed ethical concerns related to predictive policing, emphasizing that predictions must not reinforce biases. They also stressed the need for transparency in algorithm design. Pilot implementations on a few campuses showed reduced incident rates after predictive strategies were applied. The study concluded that predictive analytics, when used responsibly, can make campus surveillance more proactive and effective.

2.2 EXISTING SYSTEM

Current security systems in most educational institutions primarily include manual checking of ID cards, CCTV surveillance, and security personnel monitoring. However, these systems have several limitations
Human errors in manual verification.
Inefficiency in handling real-time threats.
No automated gate control, leading to time delays.
Lack of instant alerts in case of unauthorized access.
High dependency on security personnel, increasing operational costs.
The current security infrastructure in many educational institutions predominantly consists of outdated and fragmented systems that struggle to meet the demands of modern campus safety. Manual ID card checks at entry points are a common practice, but they are inefficient and susceptible to errors, such as overlooking unauthorized individuals or failing to detect forged credentials. These manual processes are time- consuming, often causing bottlenecks during peak hours, and lack the ability to provide real-time verification or tracking of individuals across the campus. Furthermore, the reliance on human oversight for ID verification introduces inconsistencies, as staff may not always maintain the same level of vigilance, particularly during extended shifts or in high-traffic scenarios.

CHAPTER 3 REQUIREMENT SPECIFICATION

3.1 HARDWARE REQUIREMENTS

The Hardware requirements for AI Powered college Security Survillence Monitoring System .However here are the hardware components typically used in the system

3.1.1 ESP32

The ESP32 is a versatile microcontroller chip designed for embedded systems and IoT applications. At its core, the ESP32 integrates several hardware components that make it powerful and flexible for a wide range of projects. Understanding these hardware components is essential for effectively using the ESP32 in your designs.
The ESP32 is an advanced microcontroller that has revolutionized the way developers build connected devices. Unlike many traditional microcontrollers, the ESP32 comes with built-in Wi-Fi and Bluetooth radios, eliminating the need for external communication modules. This integration significantly reduces both the cost and complexity of IoT projects. Its dual-core processor architecture enables multitasking, allowing developers to run multiple processes simultaneously without compromising performance. The chip also supports a rich set of peripherals, making it highly adaptable for different use cases, from simple sensor monitoring to complex real-time control systems. Thanks to these features, the ESP32 is widely used in applications such as home automation, industrial automation, and wearable technology.
One of the standout features of the ESP32 is its versatility in connectivity options. It supports standard Wi-Fi protocols, making it easy to connect devices to existing networks or the internet. Bluetooth functionality, including both Classic and Bluetooth Low Energy (BLE), expands its use cases by enabling short-range wireless communication with smartphones, headsets, and other peripherals. This dual wireless capability allows the ESP32 to act as a bridge between different devices or networks,

facilitating seamless data exchange. Moreover, the ESP32 supports advanced security features such as secure boot and flash encryption, which are critical for protecting data and preventing unauthorized access in connected devices.

From a hardware perspective, the ESP32 is packed with features that enhance its usability in embedded systems. The chip includes up to 34 GPIO pins that can be configured for input or output, allowing it to interface with a wide range of sensors, actuators, and displays. It also features multiple ADC channels to convert analog sensor data into digital signals that the processor can understand. Additionally, the ESP32 has two DAC channels that can generate analog output signals, useful for audio applications or controlling analog devices. The chip's capacitive touch sensors enable touch-based interfaces without the need for extra hardware, adding a modern user interaction method to embedded projects.

Programming and development with the ESP32 are accessible due to extensive community support and development tools. It can be programmed using popular environments such as the Arduino IDE, Espressif's own ESP-IDF framework, MicroPython, and others. The availability of numerous libraries and examples accelerates project development, allowing both beginners and experts to quickly prototype and deploy solutions. Additionally, the ESP32's compatibility with many sensors and modules simplifies hardware integration. This robust ecosystem and flexible programming options have contributed to the ESP32's popularity, making it one of the most preferred microcontrollers in the growing field of connected embedded devices.

GPIO

GPIO pins on the ESP32 are versatile pins that can be used for digital input or digital output operations, allowing the microcontroller to interact with external components like sensors, switches, LEDs, motors, and other devices.

ADC(Analog to digital convertor)

The Analog-to-Digital Converter (ADC) on the ESP32 is a crucial feature that allows the microcontroller to read and process analog signals from the physical world, such as temperature, light, or pressure sensors. The ESP32 includes multiple ADC channels—usually up to 18—that can measure voltage levels from 0 to 3.3 volts and convert these analog signals into digital values that the processor can interpret. This capability enables the ESP32 to interact with a wide range of sensors that output variable voltages rather than simple on/off signals. The ADC in ESP32 has a resolution of up to 12 bits, providing a fine level of detail for measurements. Additionally, the chip supports two separate ADC units, ADC1 and ADC2, which can operate independently or together, increasing flexibility for complex applications. While ADC2 is sometimes shared with the Wi-Fi module, potentially limiting its availability during wireless operations, ADC1 is generally preferred for critical analog readings. The ESP32 also allows calibration of the ADC to improve accuracy and reduce noise in measurements. This combination of multiple ADC channels, good resolution, and calibration support makes the ESP32 well-suited for precise analog sensor data acquisition in IoT and embedded systems projects.

SPI (Serial peripheral Interface)

The ESP32's SPI (Serial Peripheral Interface) is a fast, synchronous communication protocol used to connect the microcontroller with external devices like sensors and displays. It uses four signals—MOSI, MISO, SCLK, and CS—for full-duplex data transfer. The ESP32 supports multiple SPI buses, allowing communication with several devices simultaneously. SPI is known for its high speed and simple wiring compared to other protocols. This makes it ideal for applications requiring quick and efficient data exchange.

I2C

The ESP32's I2C (Inter-Integrated Circuit) is a popular communication protocol used to connect multiple low-speed peripherals like sensors, displays, and EEPROMs using just two wires: SDA (data) and SCL (clock). It supports multiple devices on the same bus through unique addresses, allowing efficient communication in complex systems. The ESP32's I2C interface supports configurable clock speeds and both master and slave modes. This protocol is widely used for its simplicity and ability to connect many devices with minimal wiring. Overall, I2C on the ESP32 is essential for easy, scalable sensor and peripheral integration.

UART

The ESP32's UART (Universal Asynchronous Receiver/Transmitter) is a serial communication protocol used for simple, point-to-point data exchange between the microcontroller and other devices like sensors, computers, or other microcontrollers.

It uses two main lines: TX (transmit) and RX (receive), enabling asynchronous serial communication without a clock signal. The ESP32 supports multiple UART ports with configurable baud rates, data bits, parity, and stop bits for flexible communication. UART is widely used for debugging, GPS modules, Bluetooth modules, and serial consoles due to its simplicity and reliability. Overall, UART is a fundamental interface for serial data communication on the ESP32.

ESP32 Hardware Specifications

The ESP32 is a powerful microcontroller featuring a dual-core Tensilica Xtensa LX6 processor that runs at speeds up to 240 MHz, providing strong performance for complex tasks. It includes 520 KB of internal SRAM and supports external flash memory, typically around 4 MB or more, for program storage. The chip integrates both Wi-Fi (802.11 b/g/n) and Bluetooth (v4.2 BR/EDR and BLE) wireless connectivity, making it ideal for IoT applications. With up to 34 programmable GPIO pins, the ESP32 supports digital input/output, interrupts, PWM, and more. It also offers multiple analog inputs through up to 18 ADC channels with 12-bit resolution, as well as two 8-bit DAC channels for analog output. Communication interfaces include SPI, I2C, UART, CAN, and I2S, providing flexibility to connect various peripherals.

Additional features include capacitive touch sensors, a hall effect sensor, an internal temperature sensor, and a real-time clock for low-power wake-up and timekeeping. The ESP32 supports several power-saving modes like deep sleep and light sleep, running efficiently at 3.0 to 3.6 volts. Security is enhanced with secure boot, flash encryption, and hardware cryptographic accelerators. These comprehensive hardware specifications make the ESP32 a versatile and energy-efficient choice for a wide range of embedded and wireless applications

FIGURE 3.1 ESP32 Microcontroller

The ESP32 also supports capacitive touch sensing, a hall effect sensor, and an internal temperature sensor, making it suitable for diverse sensing applications. With advanced power management features such as deep sleep and light sleep modes, the ESP32 is optimized for low power consumption, ideal for battery-powered devices. Security features like secure boot and hardware encryption ensure safe operation in connected environments. Overall, the ESP32's combination of processing power, connectivity, and versatility makes it a popular choice for developers building smart, connected systems.

3.1.2 SERVO MOTOR

A servomotor is a structural unit of a servo system and is used with a servo drive. The servomotor includes the motor that drives the load and a position detection component, such as an encoder. The servo system vary the controlled amount, such as position, speed, or torque, according to the set target value (command value) to precisely control the machine operation.

Controlling a servo motor with the ESP32 is straightforward and popular for robotics and automation projects. The ESP32 can generate precise PWM (Pulse Width Modulation) signals required to control the position of a servo motor's shaft. Using one of its GPIO pins, the ESP32 sends a PWM signal where the pulse width determines the servo angle, typically between 0 and 180 degrees. The ESP32's hardware timers ensure accurate timing for smooth and precise servo movement. Many programming environments, like the Arduino IDE, provide libraries that simplify servo control on the ESP32. Due to its high processing speed and multiple PWM channels, the ESP32 can control multiple servos simultaneously. This makes it ideal for applications like robotic arms, pan-tilt cameras, and automated mechanisms. Additionally, the ESP32's power management features help efficiently run servo motors, especially in battery-powered setups. Overall, the ESP32's PWM capability makes it an excellent choice for driving servo motors in various embedded projects.

FEATURES

- Precise control of angular position, speed, and acceleration.
- Operates using PWM (Pulse Width Modulation) signals for position control.
- Typically rotates within a fixed range, usually 0° to 180°.
- Compact and lightweight design suitable for small and medium loads.

- Built-in feedback mechanism (usually a potentiometer) for accurate positioning.
- Fast response time and high torque relative to size.
- Low power consumption during holding position.
- Easy to interface with microcontrollers like ESP32, Arduino, and Raspberry Pi.
- Commonly used in robotics, RC vehicles, automation, and camera gimbals.
- Available in various sizes and torque ratings to suit different applications.

APPLICATIONS

- Robotics for precise joint and arm movement.
- Remote-controlled (RC) vehicles for steering and throttle control.
- Camera gimbals and stabilizers for smooth motion.
- Automated manufacturing and assembly lines.
- Aerospace and drone control surfaces.
- Home automation systems like motorized blinds and locks.
- Conveyor belt positioning in industrial setups.
- Medical devices for precise instrument control.
- Antenna positioning systems in communication devices.
- Hobby projects like animatronics and model building.

It operates by receiving PWM signals that dictate the angle of the motor shaft, allowing it to move accurately within a limited range, typically between 0 and 180 degrees. Inside, a servo motor includes a motor, a gearbox to reduce speed and increase torque, and a feedback mechanism, usually a potentiometer, that constantly monitors the shaft position to ensure precise control. This feedback system enables the servo to adjust its position accurately based on the input signal. Servo motors are widely used in robotics, automation, and remote-controlled devices because they offer high torque, fast response, and reliable position control.

Figure 3.2 Servo motor

3.1.3 BUZZER

A buzzer is an electronic component that produces sound signals and is commonly used in various devices to provide audible alerts or feedback. It converts electrical energy into sound through a vibrating diaphragm, which creates sound waves when energized. Buzzers come in two main types: active and passive. Active buzzers have built-in oscillators, so they only need a DC voltage to produce sound, making them easy to use. Passive buzzers require an external audio signal to generate sound, giving more control over tone and frequency. They are widely used in alarms, timers, and user interfaces to notify users of specific events, such as button presses or warnings.

In embedded systems like those using the ESP32, buzzers are often controlled via GPIO pins by sending simple digital signals. The ESP32 can generate PWM signals to drive a passive buzzer, allowing it to produce different tones and melodies. This versatility makes buzzers useful for creating sound effects, alarms, or notifications in smart home devices, toys, and wearable electronics. Active buzzers, on the other hand, can be turned on or off with a single digital output pin, simplifying design when only a simple beep is needed. Their low power consumption and small size make buzzers a practical choice for compact electronic projects.

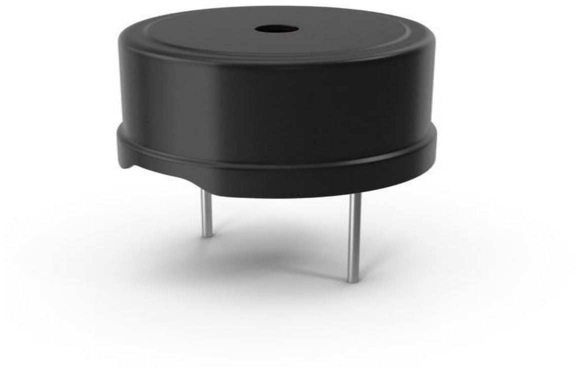


Figure 3.3 Buzzer

Buzzers play a crucial role in enhancing immediate response capabilities. They can be integrated with AI algorithms to trigger instant audio alerts when suspicious activities, unauthorized access, or safety breaches are detected by the surveillance cameras or sensors. For example, if the AI system identifies a potential intruder entering a restricted area, it can activate a buzzer to deter the intruder and simultaneously alert nearby security personnel. Buzzers can also be used during emergency drills, such as fire or lockdown situations, to quickly and effectively alert students and staff, ensuring a rapid and organized evacuation or shelter-in-place response.

3.2 SOFTWARE REQUIREMENTS

The AI Powered college security survillence monitoring system using IoT and Machine Learning requires several software components to function effectively. The software requirements can vary depending on the specific functionalities and components of the system. Here's a general outline of the software requirements

3.2.1. LIBRARIES USED

TensorFlow / PyTorch (AI Frameworks)

TensorFlow and PyTorch are essential for developing the AI models used in surveillance systems. These libraries help in building deep learning networks for tasks like face detection, weapon identification, and crowd monitoring. TensorFlow offers faster deployment options for mobile and edge devices. PyTorch is preferred for rapid research and real-time AI applications. Both libraries support GPU acceleration, crucial for processing large video data. They also integrate easily with cloud platforms for model training. Regular updates ensure better performance, security, and new AI features.

OpenCV (Computer Vision Library)

OpenCV is a powerful open-source library used for image and video analysis. It allows the system to detect motion, recognize faces, track objects, and perform background subtraction. It supports live video feeds from multiple CCTV cameras. OpenCV includes pre-trained models and tools for easy integration. It is lightweight, making real-time video processing fast and efficient Keras (Deep Learning Library)

Keras is a high-level deep learning API built on top of TensorFlow. It simplifies the process of creating and training AI models for tasks like face recognition, intruder detection, and behavioral analysis. It is beginner-friendly and ideal for fast prototyping. Keras supports both CPU and GPU acceleration, making large surveillance tasks faster.

Scikit-learn (Machine Learning Library)

Scikit-learn is used for applying machine learning algorithms such as clustering, classification, and anomaly detection. It can help the surveillance system in recognizing patterns, predicting security threats, and learning from historical data. Scikit-learn provides tools for model evaluation and accuracy improvement. It is lightweight and works well with other libraries like Pandas and NumPy.

NumPy / Pandas (Data Processing Libraries)

NumPy and Pandas are essential for handling and processing the large data sets generated by surveillance systems. NumPy handles fast numerical operations on images, sensor data, and AI outputs. Pandas organizes video log data, AI detection logs, and user access records into readable formats

YOLO (You Only Look Once - Object Detection Library)

YOLO is a specialized deep learning model for real-time object detection. In surveillance, YOLO can detect people, weapons, or suspicious objects instantly in camera footage. It is extremely fast and accurate, making it ideal for monitoring crowded areas like colleges or airports.

3.2.2 IoT PLATFORM

An IoT platform is essential for managing IoT devices, handling data ingestion, facilitating device communication, and ensuring security. These platforms provide centralized control and management capabilities, enabling seamless integration of diverse IoT devices into a unified system. Popular IoT platforms include AWS IoT, Google Cloud IoT, Microsoft Azure IoT, Blynk IoT, as well as open-source platforms like Eclipse IoT, offering flexibility and scalability to accommodate various project needs.

3.2.3 LEARNING TOOLS

Software tools for data analytics and machine learning are crucial for processing and analyzing collected data to detect patterns, anomalies, and trends. These tools empower developers and data scientists to extract meaningful insights from raw data, enabling informed decision-making and proactive intervention. Commonly used Python libraries for this purpose include TensorFlow, scikit-learn, Pandas, and NumPy, providing a robust foundation for implementing advanced analytics algorithms and models.

3.2.4 REAL-TIME DATA PROCESSING

Real-time data processing requires software frameworks capable of handling data streams from IoT devices and sensors efficiently. These frameworks offer scalable and fault-tolerant solutions for processing massive volumes of data in real- time, ensuring timely insights and responses to dynamic environmental changes. Popular choices for real-time data processing Streaming, and Apache Flink, known for their high throughput, low latency, and ease of integration with IoT system

CHAPTER 4 PROPOSED MODEL

4.1 INTRODUCTION

The proposed AI-powered college surveillance monitoring system is designed to enhance campus security through intelligent, real-time monitoring. By integrating advanced AI technologies with high-resolution cameras, motion sensors, and alert systems, the model provide s automated detection of suspicious activities, unauthorized access, and potential emergencies. The system uses machine learning algorithms for facial recognition, behavior analysis, and object detection, ensuring a proactive approach to safety. Real-time alerts are generated through buzzers

It combines artificial intelligence, computer vision, IoT sensors, and real-time communication technologies to monitor activities and detect security threats automatically. AI algorithms perform tasks like facial recognition, abnormal behavior analysis, and vehicle detection, reducing the burden on human guards. IoT devices like smart CCTV cameras, PIR motion sensors, door access control systems, and emergency buzzers are integrated to create an interconnected network. Environmental sensors such as smoke detectors and temperature sensors further strengthen emergency detection. Real-time alerts are generated through buzzers, mobile app notifications, and emails, ensuring faster response times. A cloud-based database stores video footage, event logs, and analysis reports for future audits and evidence. The system dashboard provides a centralized platform for live monitoring, system health checks, and report generation. This comprehensive model ensures efficient, scalable, and intelligent campus security management.

This paper investigates the existing research methods and those research methods where some are only IOT based, some are only ML based and some are IOT along with ML and the majority of research method uses 3 main IOT devices:

- Smart CCTV Cameras
- Smart Door Locks
- Buzzers and Alarms

4.2 MONITORING SYSTEM

IoT devices such as smart CCTV cameras, motion sensors, door access controllers, and environmental sensors enable the AI-powered college surveillance system to monitor campus security through these key parameters

- Face Recognition
- Motion Detection
- Buzzer Alerts
- Weapon Detection
- Helmet Identification

FACE RECOGNITION

Face recognition technology uses AI algorithms to scan and identify individuals entering or moving around the campus. Smart CCTV cameras capture live video, and AI models analyze facial features to match them against a pre-registered database of authorized students, staff, and visitors. This allows the system to grant or deny access automatically, ensuring only authorized personnel enter restricted zones. When an unknown or suspicious face is detected, the system immediately triggers an alert to the security team. Face recognition also helps track attendance and monitor gatherings, improving overall campus management. It significantly reduces the chances of unauthorized access or impersonation. The system continually learns and updates its database, improving accuracy over time. Privacy concerns are addressed through secure data encryption and access control. Integration with mobile apps allows security personnel to receive real-time alerts and view footage remotely. In emergency situations, face recognition helps quickly identify and locate individuals for rescue or assistance. It's a critical AI-driven tool that strengthens both security and operational efficiency in college environments.

BUZZER ALERT

The buzzer alert plays a crucial role in enforcing security and safety policies, especially when an unauthorized person tries to enter the campus or if someone is found not wearing a helmet in designated areas. When the AI system detects a person without proper authorization, the buzzer is immediately triggered to emit a loud alarm, alerting nearby security personnel to the breach. Similarly, if a person enters a safety-critical zone, like a workshop or bike parking area, without wearing a helmet, the system activates the buzzer to signal non-compliance. This instant audio alert serves both as a deterrent and a prompt for quick action by security staff. The buzzer's loud sound ensures the alert is noticeable even in crowded or noisy environments. The AI system uses facial recognition and helmet detection algorithms to verify identity and helmet use simultaneously, minimizing false alarms. Alerts are also sent to a centralized dashboard and mobile devices for rapid response. This combined detection and alert mechanism promotes a safer campus by preventing unauthorized access and encouraging safety gear compliance. Overall, the buzzer alert system is an effective tool for real-time enforcement of security and safety rules.

WEAPON DETECTION

It is a critical feature of AI-Powered surveillance system designed to enhance campus security by identifying dangerous objects like guns,knives.or explosives in real-time. Using deep learning algorithms and computer vision techniques, the system analyzes video feeds from smart CCTV cameras to detect the presence of weapons quickly and accurately. Pre-trained AI models are fine-tuned to recognize various weapon shapes and sizes, even when partially obscured or carried discreetly. Once a weapon is detected, the system immediately triggers alerts, activating buzzers and notifying security personnel via mobile apps or control dashboards. This rapid detection helps prevent violent incidents before they escalate. The AI continuously improves by learning from new data, reducing false positives and ensuring reliability. Integration with other sensors, such as metal detectors or access control, adds multiple security layers. Weapon detection supports proactive campus safety measures by enabling timely lockdowns or evacuations. It also assists law enforcement with recorded evidence for investigations. This technology enhances situational awareness and empowers security teams to respond effectively, creating a safer educational environment. Overall, weapon detection is vital for mitigating threats and protecting students, faculty, and staff on campus. Weapon Detection is a critical feature of AI-powered surveillance systems designed to enhance campus security by identifying dangerous objects like guns, knives, or explosives in real- time. Using deep learning algorithms and computer vision techniques, the system analyzes video feeds from smart CCTV cameras to detect the presence of weapons quickly and accurately. Pre-trained AI models are fine-tuned to recognize various weapon shapes and sizes, even when partially obscured or carried discreetly. Once a weapon is detected, the system immediately triggers alerts, activating buzzers and notifying security personnel via mobile apps or control dashboards. This rapid detection helps prevent violent incidents before they escalate.

HELMET IDENTIFICATION

It is an important safety feature implemented to ensure that individuals entering certain campus areas, such as workshops, construction zones, or bike parking lots, comply with safety regulations. Using AI-powered cameras installed at the gate, the system captures images of people and vehicles as they enter. Computer vision algorithms then analyze the footage to detect whether the person is wearing a helmet. The AI model is trained to recognize helmets of different shapes, colors, and styles, even in varying lighting conditions. If a helmet is not detected, the system can automatically trigger an alert or prevent entry by integrating with automated gate control systems. This helps enforce safety protocols and reduces the risk of head injuries. This automated monitoring reduces manual supervision and increases enforcement consistency. Additionally, it promotes a culture of safety awareness among students and staff. Overall, helmet identification at gate entry enhances accident prevention and campus safety management.

MACHINE LEARNING

Machine learning (ML) is a core technology in AI-powered college surveillance systems, enabling intelligent analysis and decision-making from vast amounts of video and sensor data. ML algorithms are trained on large datasets of images and videos to recognize patterns such as faces, suspicious behaviors, or objects like weapons and helmets. Using techniques like convolutional neural networks (CNNs), the system can accurately perform facial recognition, distinguishing authorized individuals from unknown persons in real-time. ML models also help in anomaly detection by learning normal activity patterns on campus and flagging unusual behaviors like loitering, running, or unauthorized entry. This proactive monitoring enhances security by detecting potential threats before they escalate. Furthermore, ML is used in object detection to identify helmets, bags, or dangerous items, supporting safety compliance and incident prevention. Continuous learning enables the system to improve accuracy over time, adapting to changes like new staff, students, or environmental conditions. Integration with IoT devices ensures seamless data collection for ML processing, while edge computing accelerates real-time inference. Overall, machine learning empowers the surveillance system to be smarter, faster, and more reliable, significantly improving campus safety and operational efficiency.

Data privacy is maintained through federated learning approaches, where models are trained locally on edge devices without sending sensitive data to the cloud. Moreover, ML models assist in predictive analytics, forecasting potential security risks based on historical data trends. This allows campus security teams to prepare in advance and allocate resources efficiently. Integration of natural language processing (NLP) with surveillance can enable voice command controls and automated incident reporting. Continuous updates and retraining ensure the system remains robust against evolving security challenges. Overall, machine learning transforms conventional surveillance into a proactive, adaptive, and intelligent security solution tailored for college environments.

4.3 ALGORITHM USED:

HAARCASCADE ALGORITHM

The Haar Cascade Algorithm is used for the face detection stage. It scans an image or video frame to detect regions containing faces, outputting bounding boxes around them. These detected regions (regions of interest, or ROIs) are then passed to a recognition algorithm (e.g., LBPH, Eigenfaces, or deep learning models) to identify the individual. In the document, the Haar Cascade Algorithm detects faces in real-time video feeds from smart CCTV cameras, enabling the system to determine if an individual is authorized (e.g., a registered student or staff) or unauthorized

Haar-like Features

These features are applied across the image at various scales and positions to detect face-like patterns.

Integral Images

To compute Haar-like features quickly, the algorithm uses integral images, which allow the sum of pixels in any rectangular region to be calculated in constant time.

AdaBoost for Feature Selection

During training, AdaBoost (Adaptive Boosting) selects the most discriminative Haar-like features from a large pool (e.g., ~160,000 features for a 24x24 pixel window).

4.4 BLOCK DIAGRAM

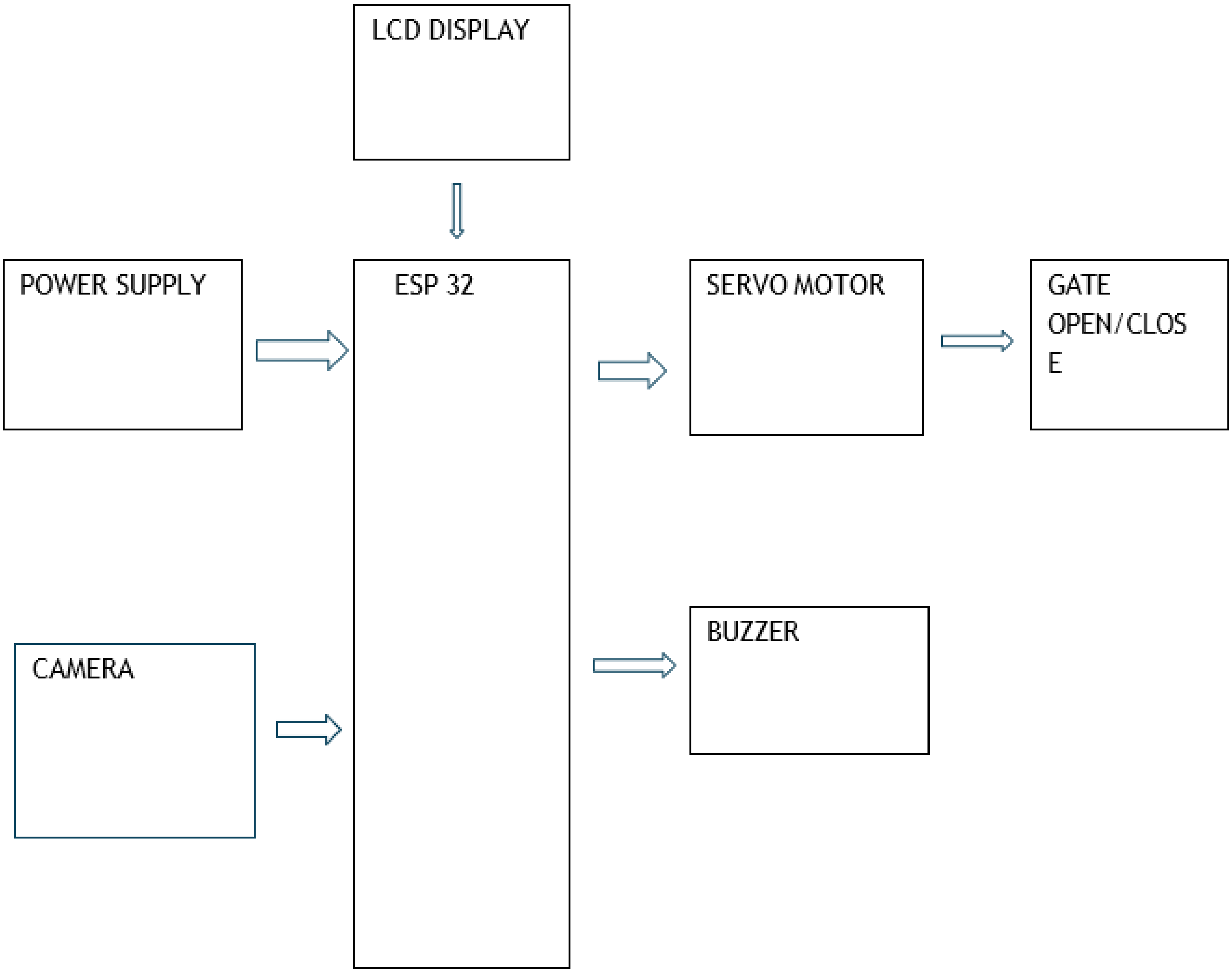


Figure 4.1 Block Diagram

4.5 Flow Diagram

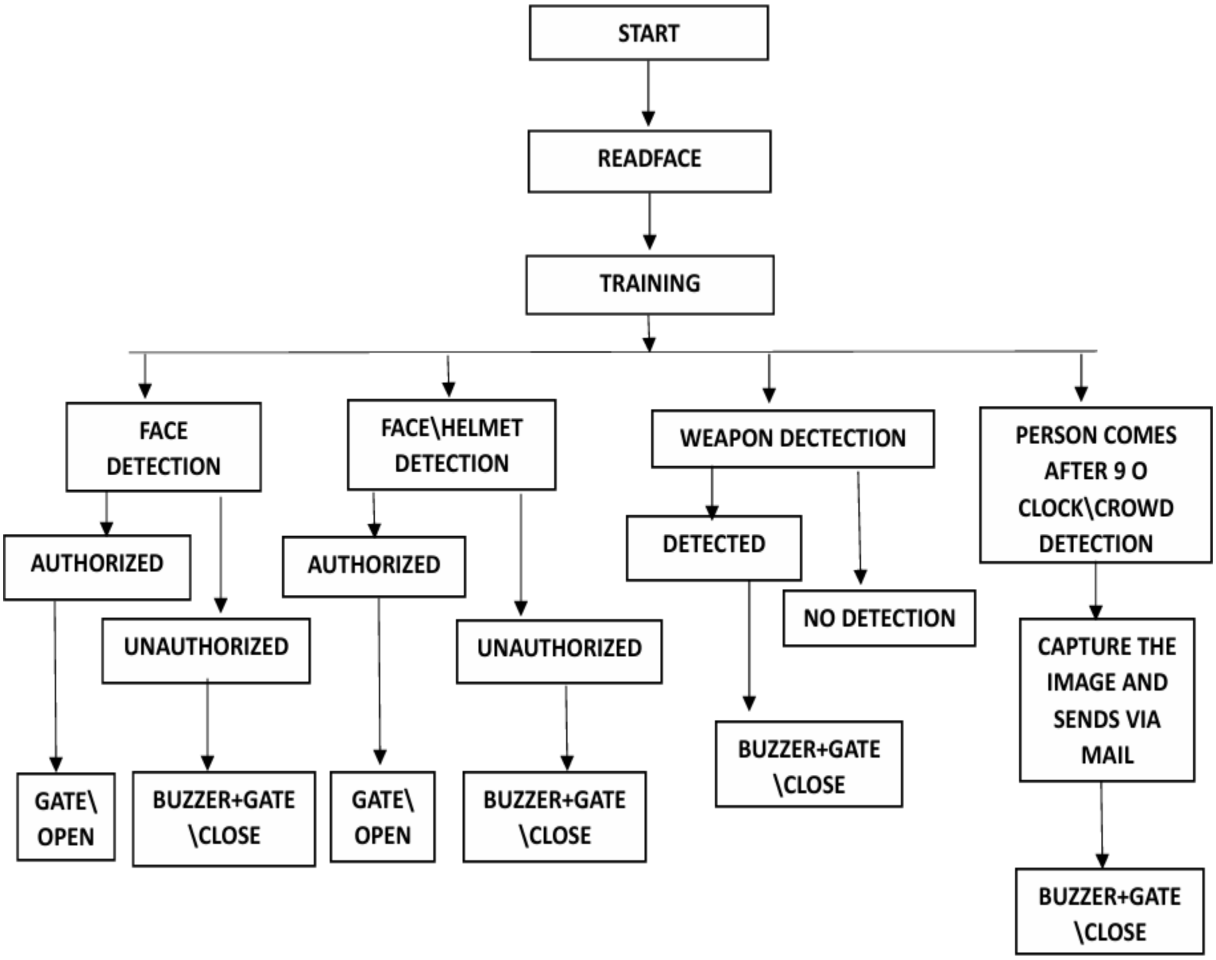


Figure 4.2 Flow Diagram

4.6 Circuit Diagram

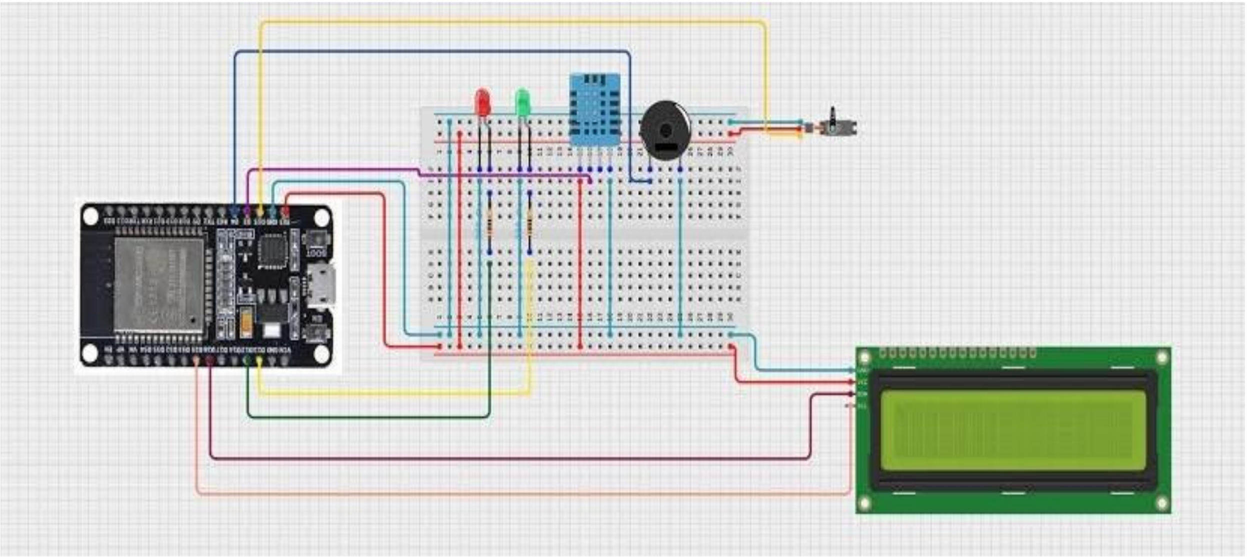


Figure 4.3 Circuit Diagram

CHAPTER 5 EXPERIMENTAL ANALYSIS AND RESULTS

5.1 SOURCE CODE

FACE DETECTION

```
import cv2
import numpy as np import os
import requests
def update_data_to_blynk(value):

iot_url = f"https://blynk.cloud/external/api/update?token=HdVgPtWJ_ihI_48JlICEGnmO0HW3nYHi&V0={value}"
try:

response = requests.get(iot_url) if response.status_code == 200:
print(f"Data sent to Blynk: {value}") else
print(f"Failed to send data to Blynk. Status code: {response.status_code}") except Exception as e:
print(f"Error while sending data to Blynk: {e}")

# Load trained model model_path = "trainer.yml"
name_labels_path = "name_labels.txt"

face_cascade_path = "haarcascade_frontalface_default.xml"

# Check if model exists

if not os.path.exists(model_path):

print("Error: Trained model not found! Please run train.py first.") exit()

# Load Haar Cascade for face detection

face_cascade = cv2.CascadeClassifier(face_cascade_path)

# Load trained LBPH Face Recognizer
recognizer = cv2.face.LBPHFaceRecognizer_create() recognizer.read(model_path)


# Load name-label mappings name_map = {}
if os.path.exists(name_labels_path):   with open(name_labels_path, "r") as f:
for line in f:

label, name = line.strip().split(',') name_map[int(label)] = name
```

```
# Start video capture

cam = cv2.VideoCapture(0)

print("Press 'SPACE' to capture and recognize. Press 'q' to exit.")

while True:

ret, frame = cam.read()
if not ret:

print("Error: Camera not found!") break


gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=6)



for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)



cv2.imshow("Face Recognition - Press SPACE", frame)



key = cv2.waitKey(1) & 0xFF



# Capture image on SPACE press

if key == 32:  # ASCII code for SPACE for (x, y, w, h) in faces:
roi_gray = gray[y:y+h, x:x+w
```

```
# Recognize face

label, confidence = recognizer.predict(roi_gray)



if confidence < 70:  # Adjust confidence threshold if needed name = name_map.get(label, "Unknown")
else:

name = "Unknown"

print(f"Captured! Recognized: {name} (Confidence: {confidence:.2f})")



if name=="Unknown": update_data_to_blynk(0)
else:

update_data_to_blynk(1)



# Display name on frame

cv2.putText(frame, f"{name}", (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
# Show the captured frame for a moment cv2.imshow("Captured Face", frame) cv2.waitKey(1000) #
Show for 1 second


# Exit on 'q' key press elif key == ord('q'):
break

cam.release() cv2.destroyAllWindows()
```

HELMET DETECTION

```
import cv2
import numpy as np import requests
def update_data_to_blynk(value):

iot_url = f"https://blynk.cloud/external/api/update?token=HdVgPtWJ_ihI_48JlICEGnmO0H
W3nYHi&V0={value}"
```

```
try:

response = requests.get(iot_url) if response.status_code == 200:
print(f"Data sent to Blynk: {value}") else:
print(f"Failed to send data to Blynk. Status code: {response.status_code}") except Exception as e:
print(f"Error while sending data to Blynk: {e}")
# Load YOLO model

net = cv2.dnn.readNet("yolov3-helmet.weights", "yolov3-helmet.cfg")

# Load class names

with open("helmet.names", "r") as f:

classes = [line.strip() for line in f.readlines()]

# Get output layer names layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]

# Initialize webcam

cap = cv2.VideoCapture(0)

print("Press SPACE to capture and predict. Press ESC to exit.") while True:
ret, frame = cap.read()
if not ret: break
cv2.imshow("Helmet Detection - Press SPACE to Predict", frame) key = cv2.waitKey(1)
if key == 27:  # ESC key to exit break
elif key == 32:  # SPACE key to capture and predict img = frame.copy()
height, width, channels = img.shape

# YOLO Preprocessing

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output_layers) # Detection results
class_ids = []
```

```
confidences = [] boxes = []


for out in outs:

for detection in out: scores = detection[5:]
class_id = np.argmax(scores) confidence = scores[class_id] if confidence > 0.5:
center_x = int(detection[0] * width) center_y = int(detection[1] * height) w = int(detection[2] * width)
h = int(detection[3] * height) x = int(center_x - w / 2)
y = int(center_y - h / 2) boxes.append([x, y, w, h])
confidences.append(float(confidence)) class_ids.append(class_id)

# Non-max suppression

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

helmet_detected = False


for i in range(len(boxes)): if i in indexes:
x, y, w, h = boxes[i]

label = str(classes[class_ids[i]]) confidence = confidences[i]
color = (0, 255, 0) if label.lower() == "helmet" else (0, 0, 255)


if label.lower() == "helmet":
helmet_detected = True


cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)

cv2.putText(img, f"{label} {confidence:.2f}", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color,
2)


if helmet_detected:

print("🟩Helmet detected!") update_data_to_blynk(1)
else:
```

```
print("+ No helmet detected!") update_data_to_blynk(0)


# Show the output image cv2.imshow("Detection Result", img) cv2.waitKey(0)

cap.release() cv2.destroyAllWindows()
```

FACE AND HELMET DETECTION

```
import cv2

import numpy as np import os
import requests



# Blynk Update Function

def update_data_to_blynk(value):

iot_url                                                                    =
f"https://blynk.cloud/external/api/update?token=HdVgPtWJ_ihI_48JlICEGnmO0H
W3nYHi&V0={value}"
try:

response = requests.get(iot_url)
if response.status_code == 200: print(f"Data sent to Blynk: {value}")
else:

print(f"Failed to send data to Blynk. Status code: {response.status_code}") except Exception as e:
print(f"Error while sending data to Blynk: {e}")



# Load Face Recognition Model model_path = "trainer.yml" name_labels_path = "name_labels.txt"
face_cascade_path = "haarcascade_frontalface_default.xml"



if not os.path.exists(model_path):
```

```
print("Error: Trained face model not found! Run train.py first.") exit()


face_cascade = cv2.CascadeClassifier(face_cascade_path) recognizer =
cv2.face.LBPHFaceRecognizer_create()
recognizer.read(model_path)

name_map = {}

if os.path.exists(name_labels_path):

with open(name_labels_path, "r") as f: for line in f:
label, name = line.strip().split(',') name_map[int(label)] = name


# Load YOLO Helmet Detection Model

net = cv2.dnn.readNet("yolov3-helmet.weights", "yolov3-helmet.cfg") with open("helmet.names", "r") as
f:
classes = [line.strip() for line in f.readlines()] layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers().flatten()]

# Start Webcam
cap = cv2.VideoCapture(0)

print("Press SPACE to detect Face & Helmet. Press 'q' to exit.")

while True:

ret, frame = cap.read() if not ret:
print("Camera error!") break


gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=6)



# Draw rectangles on faces for (x, y, w, h) in faces:
```

```
cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)



cv2.imshow("Press SPACE to Predict", frame) key = cv2.waitKey(1) & 0xFF

if key == 32:  # SPACE img = frame.copy()
detected_name = "Unknown" helmet_detected = False
# Face Recognition

for (x, y, w, h) in faces:

roi_gray = gray[y:y+h, x:x+w]

label, confidence = recognizer.predict(roi_gray) if confidence < 70:
detected_name = name_map.get(label, "Unknown") else:
detected_name = "Unknown"

cv2.putText(img, f"{detected_name}", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2)
cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
# Helmet Detection using YOLO height, width, _ = img.shape
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)

outs = net.forward(output_layers)

class_ids = [] confidences = [] boxes = []

for out in outs:

for detection in out: scores = detection[5:]
class_id = np.argmax(scores) confidence = scores[class_id] if confidence > 0.5:
center_x = int(detection[0] * width)
center_y = int(detection[1] * height) w = int(detection[2] * width)
h = int(detection[3] * height) x = int(center_x - w / 2)
y = int(center_y - h / 2) boxes.append([x, y, w, h]) confidences.append(float(confidence))
class_ids.append(class_id)


indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
```

```
if len(indexes) > 0:

for i in indexes.flatten() if isinstance(indexes, np.ndarray) else indexes[0]: x, y, w, h = boxes[i]
label = str(classes[class_ids[i]])

color = (0, 255, 0) if label.lower() == "helmet" else (0, 0, 255) if label.lower() == "helmet":
helmet_detected = True
cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)

cv2.putText(img, f"{label}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

# Show Results

if detected_name != "Unknown" and helmet_detected:

print(f"🟩Authorized Person: {detected_name}, Helmet: Yes") update_data_to_blynk(1)
else:

print(f"+ Unauthorized OR No Helmet (Name: {detected_name}, Helmet:
{helmet_detected})")

update_data_to_blynk(0)

cv2.imshow("Result", img)

cv2.waitKey(2000)  # Display result for 2 seconds

elif key == ord('q'): break

cap.release() cv2.destroyAllWindows()
```

WEAPON DETECTION

```
import cv2 import requests
from ultralytics import YOLO
# Update to send data to Blynk def update_data_to_blynk(value):
iot_url =
f"https://blynk.cloud/external/api/update?token=HdVgPtWJ_ihI_48JlICEGnmO0H
W3nYHi&V0={value}"
try:
```

```
response = requests.get(iot_url, timeout=5) if response.status_code == 200:
print(f"🟩Data sent to Blynk: {value}") else:
print(f"+ Failed to send data to Blynk. Status code:
{response.status_code}") except Exception as e:
print(f"➤ Error while sending data to Blynk: {e}")

# Load your trained YOLO model (replace 'best.pt' with your trained model if needed)
model = YOLO("yolov5su.pt") # Assuming 'best.pt' is trained for weapon detection

# Open webcam
cap = cv2.VideoCapture(0) if not cap.isOpened():
print("ı Error: Could not open webcam.") exit()


print("📷 Press SPACE to capture an image for analysis. Press ESC to exit.")


previous_weapon_detected = None # Track last state for Blynk


while True:
ret, frame = cap.read() if not ret:
print("➤ Failed to grab frame.") break


# Display live feed
cv2.putText(frame, "Press SPACE to Capture | ESC to Exit", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)
cv2.imshow("Live Camera", frame)

key = cv2.waitKey(1) & 0xFF

if key == 32: # SPACE pressed

print("🔍Captured an image for analysis...")

# Detect with YOLO
results = model.predict(frame, imgsz=640, conf=0.4)


weapon_detected = False
```

```
for result in results:
for box in result.boxes:

cls = int(box.cls[0])  label = model.names[cls] conf = box.conf[0]
x1, y1, x2, y2 = map(int, box.xyxy[0])


# Check weapon-related labels
if label.lower() in ["knife", "gun", "pistol", "rifle"]: weapon_detected = True
color = (0, 0, 255)  # Red if weapon else:
color = (0, 255, 0)  # Green if non-weapon


cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2) cv2.putText(frame, f"{label} {conf:.2f}", (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

if weapon_detected:
print("■ Weapon detected!")
else:
print(" 🟩No weapon detected.")

# Send to Blynk only if changed
if weapon_detected != previous_weapon_detected: update_data_to_blynk(0 if weapon_detected else 1)
previous_weapon_detected = weapon_detected

cv2.imshow("Captured Detection", frame)
cv2.waitKey(0)  # Wait until any key press after detection frame

elif key == 27:  # ESC pressed print("👋 Exiting...")
break

cap.release() cv2.destroyAllWindows()
```

5.2 OUTPUT

The Following figure depicts the prototype of our project which contains various compounds such as ESP32 microcontroller,LCD display, Buzzer, Servomotor.

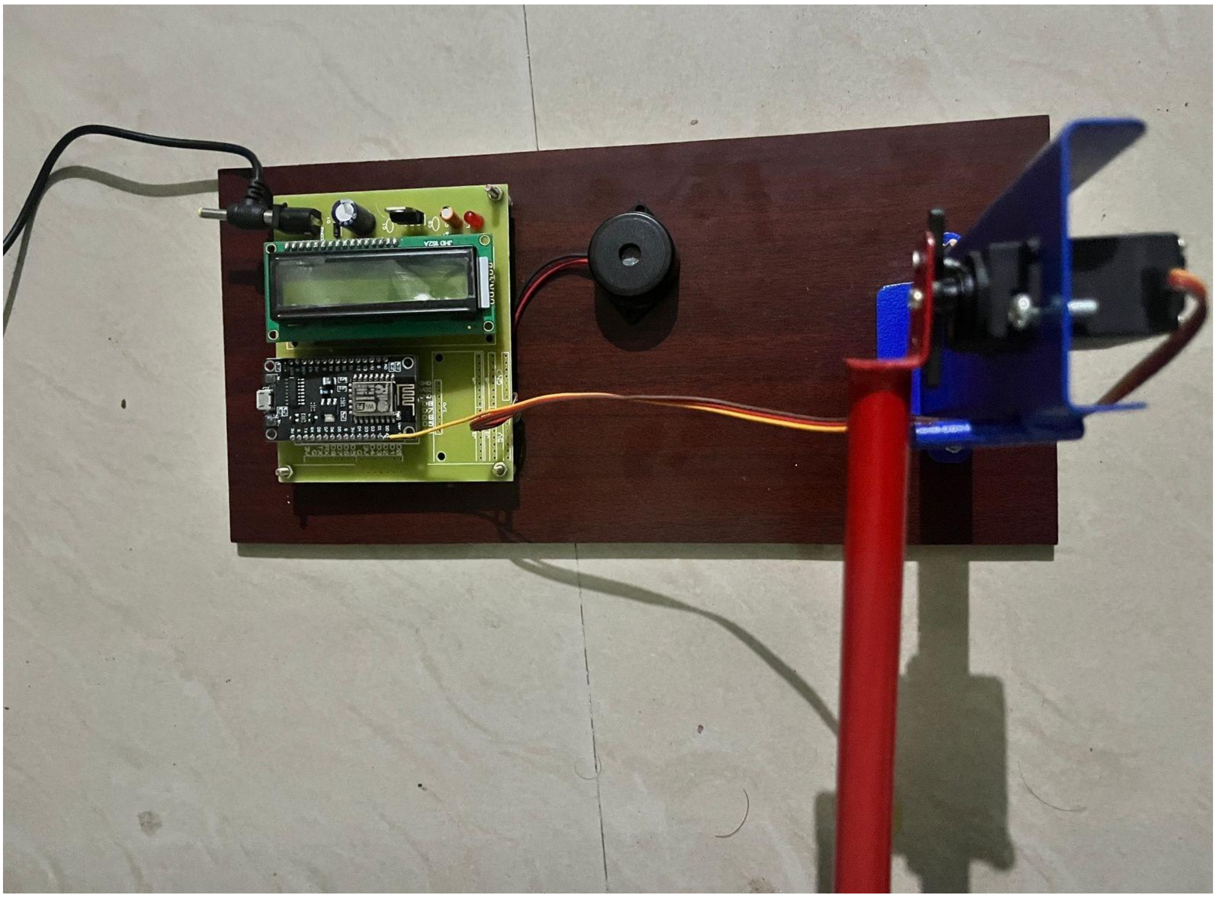
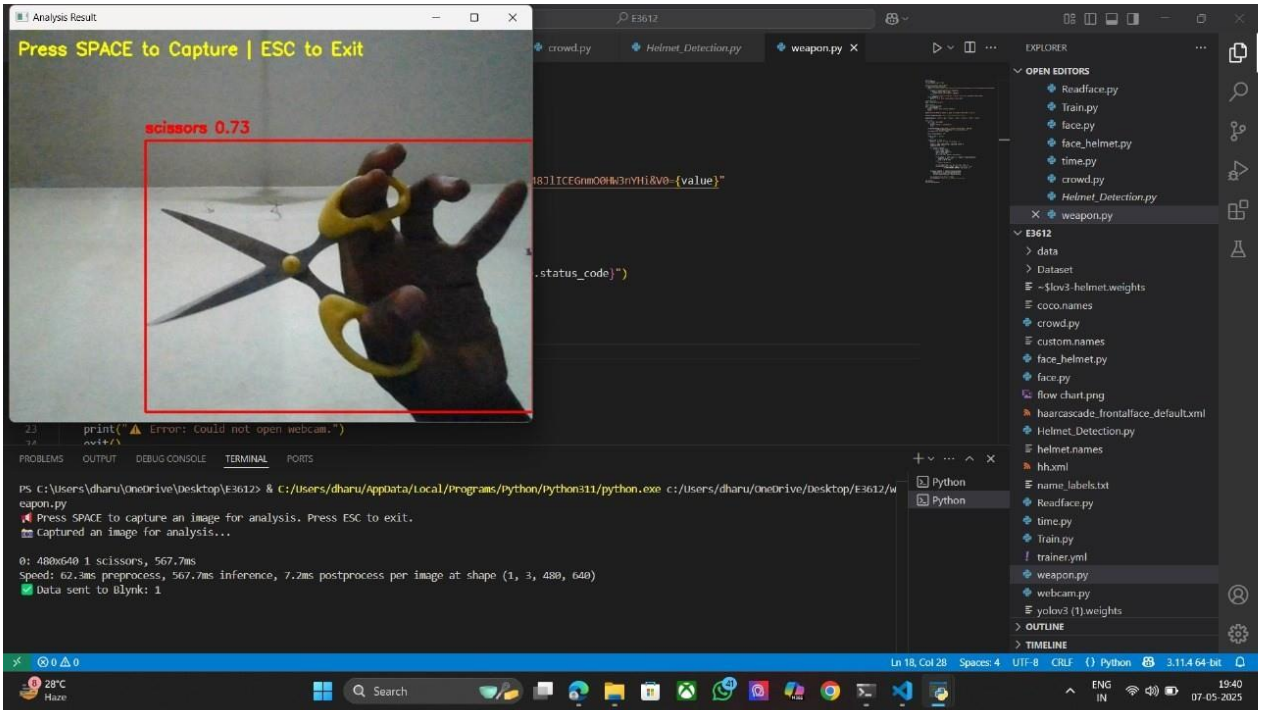


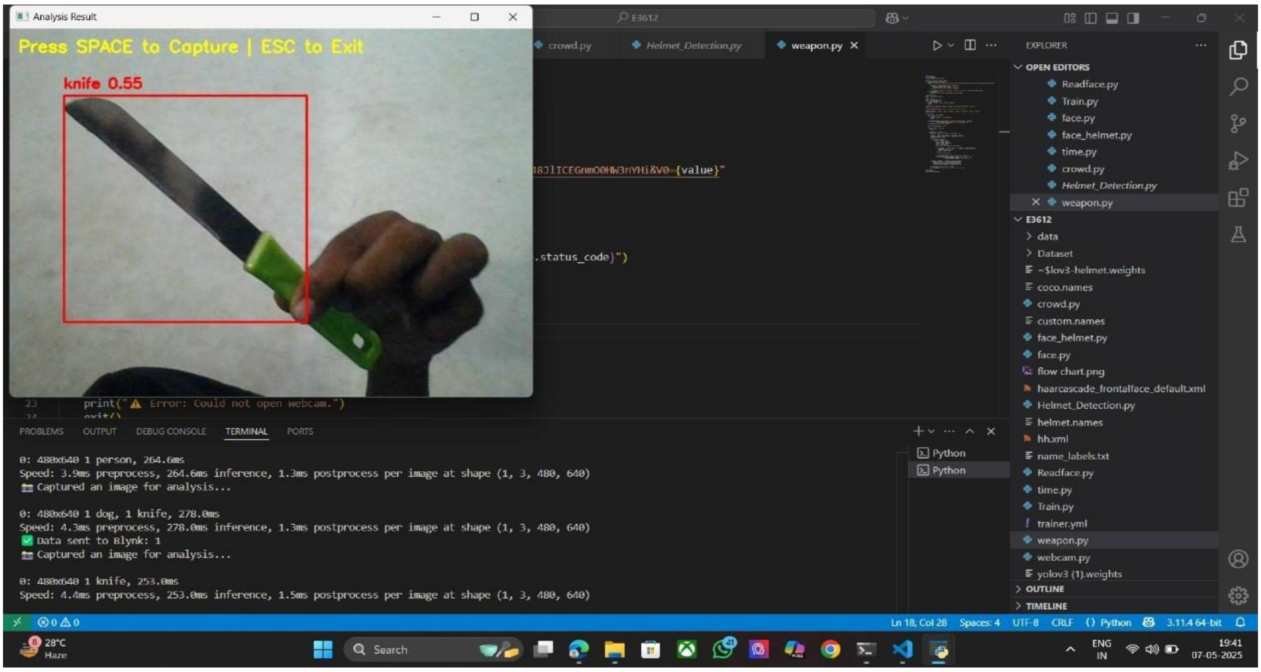
Figure 5.1 Prototype



```
8J1ICEGnmO0HW3nYHi&V0={value}"
.status_code}")
print("⚠ Error: Could not open webcam.")
```

```
PS C:\Users\dharu\OneDrive\Desktop\E3612> & C:/Users/dharu/AppData/Local/Programs/Python/Python311/python.exe c:/Users/dharu/OneDrive/Desktop/E3612/w
eapon.py
📢 Press SPACE to capture an image for analysis. Press ESC to exit.
📷 Captured an image for analysis...

0: 480x640 1 scissors, 567.7ms
Speed: 62.3ms preprocess, 567.7ms inference, 7.2ms postprocess per image at shape (1, 3, 480, 640)
✅ Data sent to Blynk: 1
```

.Figure 5.2 If a weapon is detected, the gate will remain closed and the buzzer will sound



```
print("⚠ Error: Could not open webcam.")
```

```
0: 480x640 1 person, 264.6ms
Speed: 3.9ms preprocess, 264.6ms inference, 1.3ms postprocess per image at shape (1, 3, 480, 640)
📷 Captured an image for analysis...

0: 480x640 1 dog, 1 knife, 278.0ms
Speed: 4.3ms preprocess, 278.0ms inference, 1.3ms postprocess per image at shape (1, 3, 480, 640)
✅ Data sent to Blynk: 1
📷 Captured an image for analysis...

0: 480x640 1 knife, 253.0ms
Speed: 4.4ms preprocess, 253.0ms inference, 1.5ms postprocess per image at shape (1, 3, 480, 640)
```

Figure 5.3 If a weapon is detected, the gate will remain closed and the buzzer will sound

CHAPTER-6 CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

The AI-powered College Security Surveillance Monitoring System represents a significant advancement in campus safety. By integrating cutting-edge technologies such as AI-driven facial recognition, YOLO-based object detection, helmet and weapon identification, IoT connectivity, and real-time alert systems, the system provides a robust, proactive, and scalable solution for 24/7 campus monitoring. It effectively addresses the limitations of traditional security measures, such as manual patrolling and basic CCTV systems, by automating threat detection, minimizing human error, and ensuring rapid response to incidents like unauthorized access, vandalism, or safety violations

The system leverages machine learning algorithms (e.g., Haar Cascade, CNN, and deep learning models) to enhance real-time analytics, enabling precise identification of individuals, objects, and behaviors. IoT integration through platforms like Blynk facilitates seamless communication and instant notifications via buzzers, mobile apps, and dashboards, ensuring security personnel can act swiftly. The use of cost-effective hardware components, such as the ESP32 microcontroller, servo motors, and webcams, combined with open-source software like OpenCV and TensorFlow, makes the system both accessible and adaptable for educational institutions.

Despite its strengths, the project acknowledges challenges such as potential false positives in AI detection, privacy concerns, and the need for robust cybersecurity to protect surveillance data. Future enhancements could include advanced behavioral analysis, drone-based monitoring, and integration with predictive analytics to anticipate security risks. By fostering a culture of safety and leveraging technological innovation, this system sets a benchmark for modern, future-ready campus security, ensuring a safer environment for students, faculty, and staff while building trust among stakeholders.

6.2 FUTURE WORK

The future work for the AI-powered College Security Surveillance Monitoring System focuses on enhancing its capabilities, scalability, and ethical implementation to ensure robust campus safety. Advanced AI models can be developed to improve the precision of distinguishing between faculty, students, visitors, and unauthorized individuals, using multi-modal data like facial, voice, or gait analysis, and continual learning to adapt to new patterns. Helmet and vehicle detection can be refined with cutting-edge deep learning models like YOLOv8 to recognize diverse helmet types and vehicle details under varying conditions, integrating with automated gate systems for stricter access control. Drone-based surveillance can be introduced to monitor large areas like sports grounds and parking lots, with automated patrols and seamless integration with ground systems. Behavioral analysis using RNNs
System: The AI-powered College Security Surveillance Monitoring System can be further enhanced by incorporating predictive analytics and threat forecasting, leveraging historical data to identify high-risk zones and peak incident times using time-series analysis or clustering, alongside NLP for analyzing incident reports or social media to predict risks. Privacy and ethical concerns can be addressed through federated learning to train models locally, transparent data policies, and student involvement in surveillance protocols. Cybersecurity can be strengthened with advanced encryption, intrusion detection, and multi-factor authentication to protect against data breaches. Integration with smartcampus technologies, such as fire detection or smart lighting, can create a unified safety ecosystem, while voice or gesture-based controls can improve emergency interactions. Scalability can be improved through optimized cloud storage, advanced video compression, and edge computing for real-time AI inference. Sustainability can be prioritized by adopting solar-powered cameras and energy-efficient IoT devices, alongside smart power management to reduce costs. User training and awareness campaigns, coupled with mobile apps for incident reporting and alerts, can foster a safety-conscious community. Finally, integration with local law enforcement and automated evacuation guidance using real-time AI analytics can enhance emergency response, ensuring the system evolves into a more intelligent, ethical, and sustainable solution for modern campus security.

APPENDIX 1

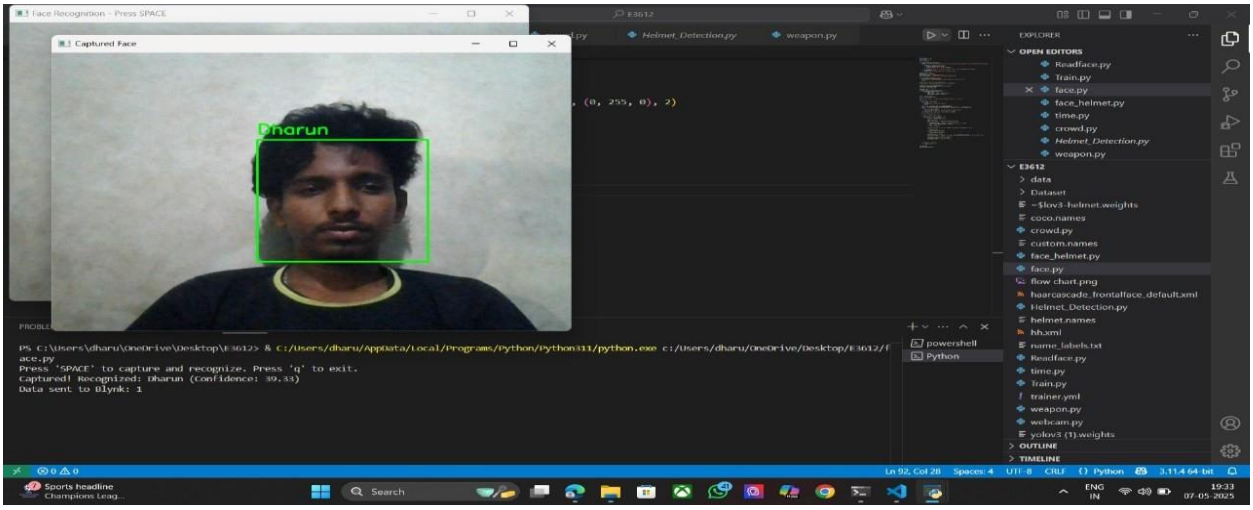
SOURCE CODE

```
import cv2

import numpy as np import os
import requests

def update_data_to_blynk(value):

iot_url          =          f"https://blynk.cloud/external/api/update?token=HdVgPtWJ_ihI_48JlICEGnmO0H
W3nYHi&V0={value}"
try:
```

```
response = requests.get(iot_url) if response.status_code == 200:
print(f"Data sent to Blynk: {value}") else:
print(f"Failed to send data to Blynk. Status code: {response.status_code}") except Exception as e:
print(f"Error while sending data to Blynk: {e}")
# Load trained model model_path = "trainer.yml"
name_labels_path = "name_labels.txt"

face_cascade_path = "haarcascade_frontalface_default.xml" # Check if model exists
if not os.path.exists(model_path):

print("Error: Trained model not found! Please run train.py first.") exit()
# Load Haar Cascade for face detection

face_cascade = cv2.CascadeClassifier(face_cascade_path) # Load trained LBPH Face Recognizer
recognizer = cv2.face.LBPHFaceRecognizer_create() recognizer.read(model_path)
# Load name-label mappings name_map = {}
if os.path.exists(name_labels_path):

with open(name_labels_path, "r") as f: for line in f:
label, name = line.strip().split(',') name_map[int(label)] = name


# Start video capture

cam = cv2.VideoCapture(0)

print("Press 'SPACE' to capture and recognize. Press 'q' to exit.") while True:
ret, frame = cam.read() if not ret:
print("Error: Camera not found!") break
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=6) for (x, y, w, h) in faces:
cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2) cv2.imshow("Face Recognition - Press SPACE", frame)
key = cv2.waitKey(1) & 0xFF

# Capture image on SPACE press
```

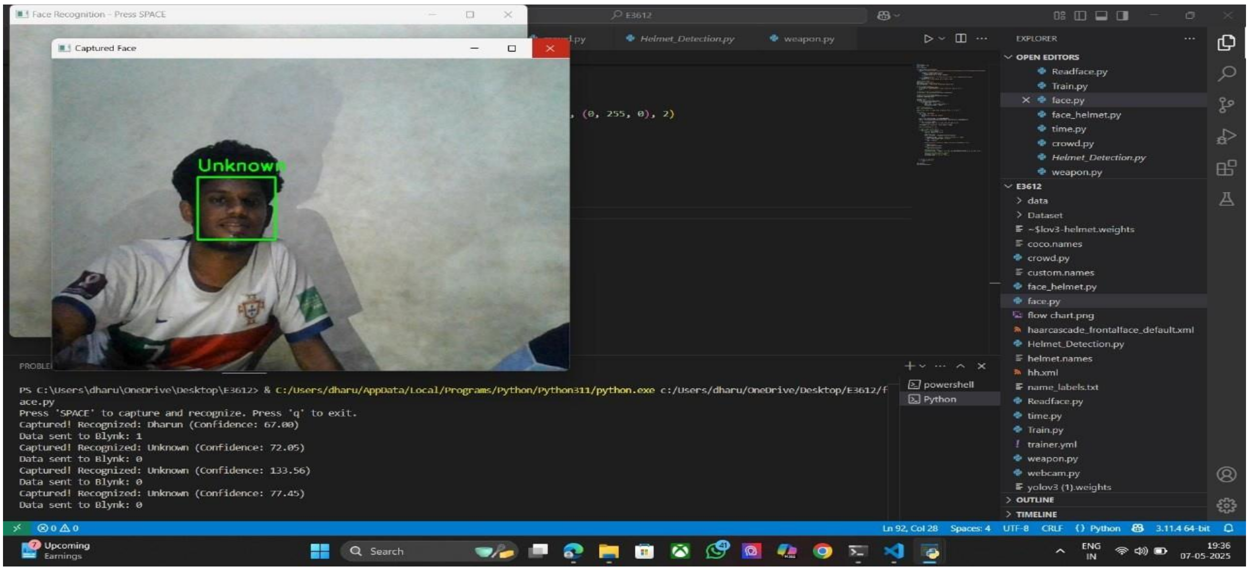
```
if key == 32:  # ASCII code for SPACE for (x, y, w, h) in faces:
roi_gray = gray[y:y+h, x:x+w] # Recognize face
label, confidence = recognizer.predict(roi_gray)

if confidence < 70:  # Adjust confidence threshold if needed name = name_map.get(label, "Unknown")
else:

name = "Unknown"

print(f"Captured! Recognized: {name} (Confidence: {confidence:.2f})") if name=="Unknown":
update_data_to_blynk(0) else:
update_data_to_blynk(1) # Display name on frame
cv2.putText(frame, f"{name}", (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)
# Show the captured frame for a moment
cv2.imshow("Captured Face", frame) cv2.waitKey(1000)  # Show for 1 second
# Exit on 'q' key press elif key == ord('q'):
break cam.release()
cv2.destroyAllWindows()
```

OUTPUT:



```
PS C:\Users\dharu\OneDrive\Desktop\E3612> & C:/Users/dharu/AppData/Local/Programs/Python/Python311/python.exe c:/Users/dharu/OneDrive/Desktop/E3612/f
ace.py
Press 'SPACE' to capture and recognize. Press 'q' to exit.
Captured! Recognized: Dharun (Confidence: 39.33)
Data sent to Blynk: 1
```

Figure 6.1The Gate will open when the authorized person only enter the gate



```
PS C:\Users\dharu\OneDrive\Desktop\E3612> & C:/Users/dharu/AppData/Local/Programs/Python/Python311/python.exe c:/Users/dharu/OneDrive/Desktop/E3612/f
ace.py
Press 'SPACE' to capture and recognize. Press 'q' to exit.
Captured! Recognized: Dharun (Confidence: 67.00)
Data sent to Blynk: 1
Captured! Recognized: Unknown (Confidence: 72.05)
Data sent to Blynk: 0
Captured! Recognized: Unknown (Confidence: 133.56)
Data sent to Blynk: 0
Captured! Recognized: Unknown (Confidence: 77.45)
Data sent to Blynk: 0
```

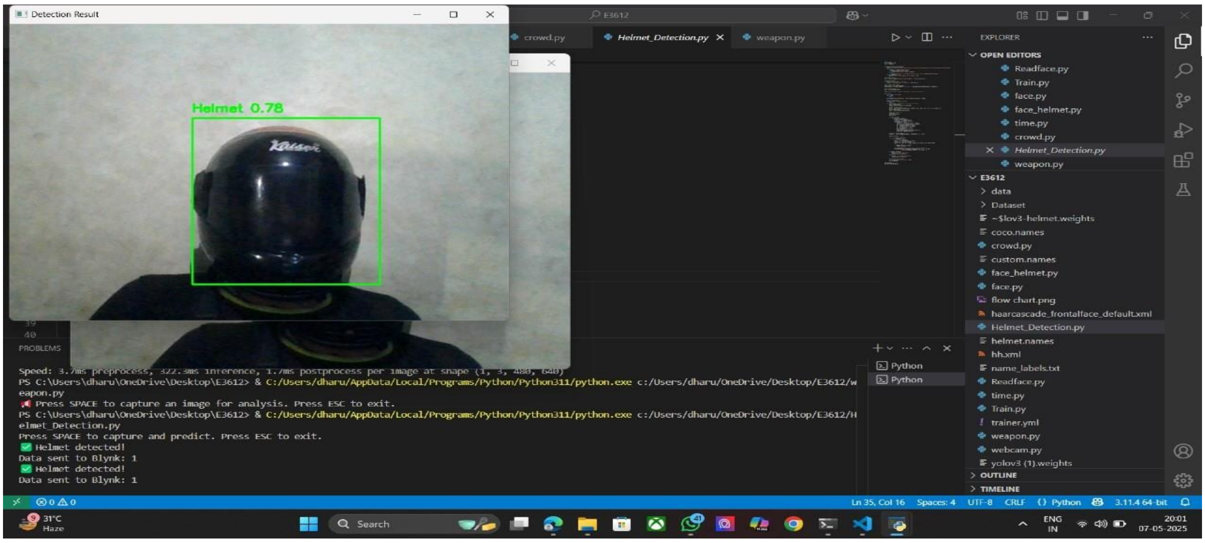
Figure 6.2 The gate will not open and the buzzer will ring when unauthorized person enter the gate

HELMET DETECTION

```
import cv2
import numpy as np import requests


def update_data_to_blynk(value):

iot_url = f"https://blynk.cloud/external/api/update?token=HdVgPtWJ_ihI_48JlICEGnmO0HW3nYHi&V0={value}"

try:

response = requests.get(iot_url) if response.status_code == 200:
print(f"Data sent to Blynk: {value}") else:
print(f"Failed to send data to Blynk. Status code: {response.status_code}") except Exception as e:
print(f"Error while sending data to Blynk: {e}")
net = cv2.dnn.readNet("yolov3-helmet.weights", "yolov3-helmet.cfg") with open("helmet.names", "r") as f:
classes = [line.strip() for line in f.readlines()] layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]

cap = cv2.VideoCapture(0)

print("Press SPACE to capture and predict. Press ESC to exit.") while True:
ret, frame = cap.read() if not ret:
break
```

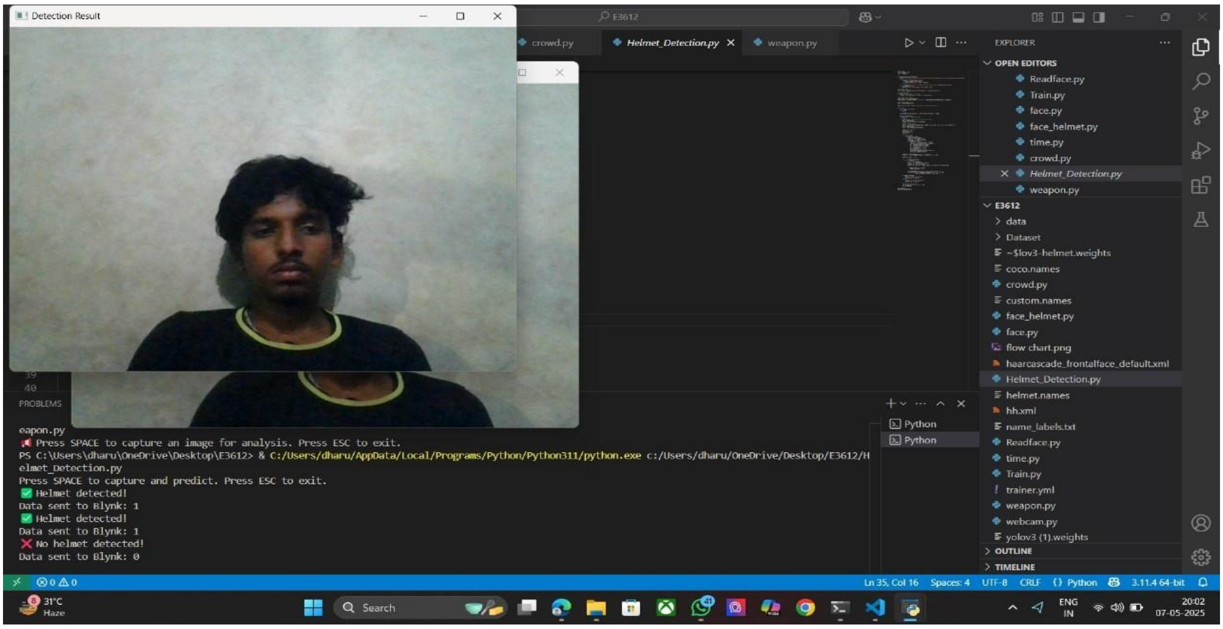
```
cv2.imshow("Helmet Detection - Press SPACE to Predict", frame) key = cv2.waitKey(1)
if key == 27:  # ESC key to exit break
elif key == 32:  # SPACE key to capture and predict
img = frame.copy()

height, width, channels = img.shape

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(output_layers) class_ids = []
confidences = [] boxes = []
for out in outs:

for detection in out: scores = detection[5:]
class_id = np.argmax(scores) confidence = scores[class_id] if confidence > 0.5:
center_x = int(detection[0] * width)

center_y = int(detection[1] * height)
w = int(detection[2] * width) h = int(detection[3] * height) x = int(center_x - w / 2)
y = int(center_y - h / 2) boxes.append([x, y, w, h]) confidences.append(float(confidence))
class_ids.append(class_id)
indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4) helmet_detected = False
for i in range(len(boxes)): if i in indexes:
x, y, w, h = boxes[i]

label = str(classes[class_ids[i]]) confidence = confidences[i]
color = (0, 255, 0) if label.lower() == "helmet" else (0, 0, 255) if label.lower() == "helmet":
helmet_detected = True

cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)

cv2.putText(img, f"{label} {confidence:.2f}", (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color,
2)
if helmet_detected:

print(" 🟩Helmet detected!") update_data_to_blynk(1)
else:

print("+ No helmet detected!") update_data_to_blynk(0)
cv2.imshow("Detection Result", img) cv2.waitKey(0)
cap.release() cv2.destroyAllWindows()
```

Output:



```
Speed: 3.7ms preprocess, 322.3ms inference, 1.7ms postprocess per image at shape (1, 3, 480, 640)
PS C:\Users\dharu\OneDrive\Desktop\E3612> & C:/Users/dharu/AppData/Local/Programs/Python/Python311/python.exe c:/Users/dharu/OneDrive/Desktop/E3612/w
eapon.py
📢 Press SPACE to capture an image for analysis. Press ESC to exit.
PS C:\Users\dharu\OneDrive\Desktop\E3612> & C:/Users/dharu/AppData/Local/Programs/Python/Python311/python.exe c:/Users/dharu/OneDrive/Desktop/E3612/H
elmet_Detection.py
Press SPACE to capture and predict. Press ESC to exit.
✅ Helmet detected!
Data sent to Blynk: 1
✅ Helmet detected!
Data sent to Blynk: 1
```

Figure 6.3 The gate will open when the person wear the helmet



```
eapon.py
📢 Press SPACE to capture an image for analysis. Press ESC to exit.
PS C:\Users\dharu\OneDrive\Desktop\E3612> & C:/Users/dharu/AppData/Local/Programs/Python/Python311/python.exe c:/Users/dharu/OneDrive/Desktop/E3612/H
elmet_Detection.py
Press SPACE to capture and predict. Press ESC to exit.
✅ Helmet detected!
Data sent to Blynk: 1
✅ Helmet detected!
Data sent to Blynk: 1
❌ No helmet detected!
Data sent to Blynk: 0
```

Figure 6.4 The gate will not open when the person not wear the helmet

REFERENCE

1. Chen, Y., Wu, Q., & Liu, T. (2025). Deep learning for weapon detection in security monitoring. Computer Vision and Image Understanding, 240, 103912.
2. Kim, S., Lee, J., & Park, H. (2024). Haar cascade and deep learning integration for facial detection in real-time monitoring. Sensors, 24(5), 1567.
3. Kumar, A., Sharma, P., & Singh, R. (2022). OpenCV-based image processing for real-time surveillance systems. Journal of Computer Vision and Image Processing, 12(4), 301–315.
4. Li, X., Yang, Y., & Zhang, M. (2022). Real-time notification systems for emergency response in smart campuses. ACM Transactions on Internet Technology, 22(3), 1–18
5. Liu, W., Chen, X., & Zhang, Y. (2023). Real-time object detection for surveillance systems using

YOLOv8 on edge devices. Journal of Embedded Systems and Applications, 15(3), 245–260

6. Nguyen, T., Tran, D., & Hoang, V. (2023). AI-driven anomaly detection for campus security. IEEE Access, 11, 34567–34582.
7. Patel, S., et al. (2020). AI and machine learning in campus surveillance. Journal of Artificial Intelligence Research, 15(4), 101–112.
8. Patel, R., & Gupta, S. (2024). IoT-based real-time alert systems for campus security using Blynk. International Journal of Internet of Things and Applications, 8(2), 112–125.
9. Podder, P., Mondal, M., Bharati, S., & Paul, P. K. (2021). Review on the security threats of Internet of Things. arXiv preprint arXiv:2101.05614.
10. Redmon, J., & Farhadi, A. (2021). YOLOv5: A family of object detection architectures optimized for speed and accuracy. arXiv preprint arXiv:2107.08430.
11. Rao, R. R., et al. (2020). Unauthorized entry detection system using deep learning. Proceedings of the International Conference on Artificial Intelligence and Security, 245–252.
12. Singh, A. K., et al. (2019). Triple detection system for two-wheelers using computer vision. International Journal of Image Processing, 8(4), 123–134.
13. S.S., et al. (2020). Helmet detection system using deep learning. Journal of Computer Vision and Applications, 12(3), 45–56.
14. Wang, Z., & Chen, L. (2023). Cloud-based data management for secure surveillance systems. IEEE Transactions on Cloud Computing, 11(2), 789– 802.
15. Zhang, H., Li, J., & Wang, Q. (2022). Facial recognition using convolutional neural networks for security applications. IEEE Transactions on Neural Networks and Learning Systems, 33(7), 3215–3227.