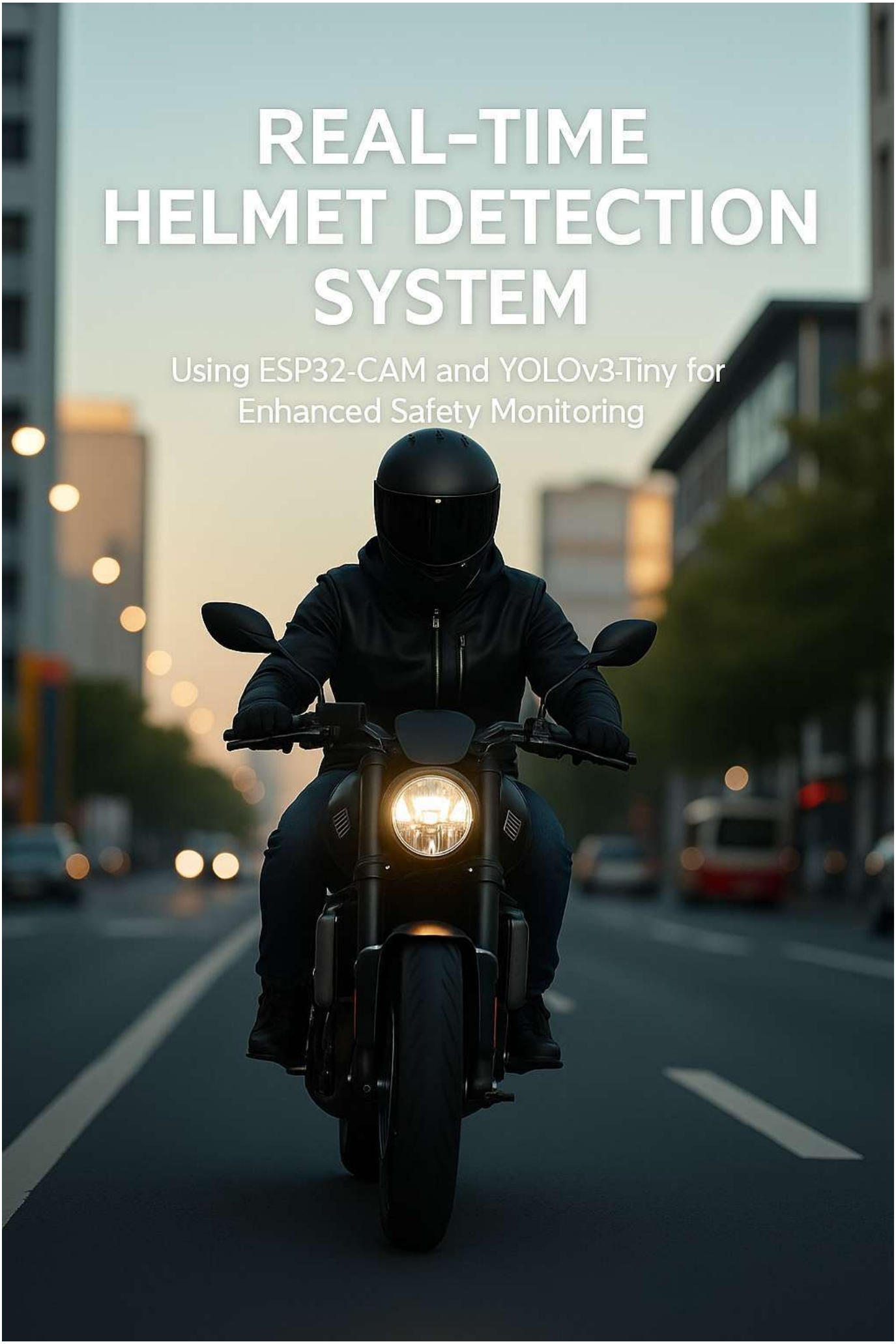
REAL-TIME
HELMET DETECTION
SYSTEM
Using ESP32-CAM and YOLOv3-Tiny for
Enhanced Safety Monitoring

ABSTRACT

The Real-Time Helmet Detection System using YOLOv3-Tiny is an advanced intelligent surveillance solution engineered to automate the monitoring of helmet use among two-wheeler riders. Designed to address the persistent concern of road safety, especially in busy urban environments where traffic violations can easily go unnoticed, this system harnesses the power of the lightweight YOLOv3-Tiny object detection model. By processing real-time video feeds, it efficiently identifies riders who are not wearing helmets and enables immediate alert generation or systematic logging of violations for later analysis. The development strategy behind this system involves fine-tuning a deep learning model on a comprehensive custom dataset featuring diverse helmeted and non-helmeted scenarios. This rigorous training process is enhanced by techniques such as data augmentation, anchor box optimization, and transfer learning, which collectively contribute to an impressive level of detection accuracy and system robustness.

Engineered with edge deployment in mind, the system is optimized for low-latency operations, making it an ideal solution for real-world applications like smart traffic monitoring and automated safety enforcement. The robust design not only promotes timely identification of helmet violations but also plays a significant role in minimizing road accidents and enhancing overall safety compliance among riders. By seamlessly integrating deep learning methodologies with real-time object detection and edge computing capabilities, the solution delivers reliable performance in dynamic traffic environments while reducing the need for manual oversight. This innovative approach fosters safer driving habits and supports wider initiatives aimed at transforming urban traffic surveillance and control systems.

Keywords: Helmet Detection, YOLOv3-Tiny, Real-Time Surveillance, Road Safety, Deep Learning, Object Detection, Smart Traffic Monitoring, Embedded Visio

Contents

List of Figures

LIST OF ABBREVIATIONS

Abbreviation	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
GPIO	General Purpose Input Output
GSM	Global System for Mobile Communication
HOG	Histogram of Oriented Gradients
IDE	Integrated Development Environment
IoT	Internet of Things
LCD	Liquid Crystal Display
LED	Light Emitting Diode
OTP	One-Time Password
PCB	Printed Circuit Board
PPE	Personal Protective Equipment
RAM	Random Access Memory
RFID	Radio Frequency Identification
RX	Receive (UART Communication)
SD	Secure Digital
SVM	Support Vector Machine
TX	Transmit (UART Communication)

CHAPTER 1

INTRODUCTION

1.1 Introduction

In recent years, safety regulations and enforcement have become increasingly vital in various industries, particularly in construction, manufacturing, and transportation sectors. Helmet usage is a crucial aspect of personal protective equipment (PPE), significantly reducing the risk of head injuries. However, ensuring compliance with helmet-wearing policies remains a challenge due to manual monitoring limitations.

This project presents a Real-Time Helmet Detection System utilizing the ESP32-CAM and YOLOv3-Tiny for efficient and automated safety monitoring. The system aims to detect individuals wearing or not wearing helmets in real-time, providing instant alerts to ensure compliance with safety protocols.

The ESP32-CAM is a low-cost, compact module equipped with a camera and Wi-Fi capabilities, making it suitable for edge-based AI applications. YOLOv3-Tiny, a lightweight object detection algorithm, enables the system to process image frames quickly while maintaining accurate helmet detection. By leveraging these technologies, the system can be deployed in workplaces, construction sites, and industrial zones where safety monitoring is essential.

The proposed system offers several advantages, including real-time monitoring, cost-effectiveness, and ease of deployment. Additionally, the integration of cloud-based reporting and alert mechanisms enhances safety enforcement. This document outlines the development, implementation, and benefits of the Real-Time Helmet Detection System using ESP32-CAM and YOLOv3-Tiny, demonstrating its potential impact on improving workplace safety standards.

1.2 Introduction to Embedded System

Physically, embedded systems range from portable devices such as digital watches and MP3 players to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low,

with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure. In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, handheld computers share some elements with embedded systems such as the operating systems and microprocessors which power them but are not truly embedded systems because they allow different applications to be loaded and peripherals to be connected. Embedded systems have a span in all aspects of modern life and there are many examples of their use. Telecommunications systems employ numerous embedded systems from telephone switches for the network and network bridges to route data.As IoT continues to evolve, it is poised to revolutionize how we interact with the world by enabling smarter environments and more efficient systems.

1.2.1 Examples of Embedded System

Automated teller machines (ATMs) Integrated systems in aircraft and missiles Cellular telephones and telephonic switches Computer network equipment, including routers, time servers, and firewalls Computer printers, copiers Disk drives (floppy disk drive and hard disk drive) Medical equipment Programmable logic controllers (PLCs) for industrial automation and monitoring Stationary IDE of game controllers

1.3 Introduction to IoT

The Internet of Things (IoT) plays a crucial role in enhancing helmet detection systems by enabling real-time monitoring, data transmission, and automated safety enforcement. IoT integrates smart sensors, cameras, and AI-powered image processing units to detect whether individuals are wearing helmets in environments such as traffic surveillance, construction sites, and industrial workplaces. These IoT-enabled devices capture images or video feeds, process them using deep learning models like YOLO or Faster R-CNN, and transmit the results to cloud-based platforms or local servers for further analysis and action.

With IoT connectivity, helmet detection systems can be integrated with edge computing devices such as Raspberry Pi, NVIDIA Jetson, or Arduino, enabling faster

1.4 Software Tools

- Arduino IDE – Used to write, compile, and upload code to the ESP32-CAM ESP32 Board Manager and Libraries – Necessary drivers and libraries for ESP32 programming.
- Python (For AI Processing) – Used for advanced AI image classification if external processing is required OpenCV – Helps in pre-processing images for helmet detection.
- TensorFlow Lite – Lightweight AI models that can run on the ESP32-CAM for local inference

1.5 Additional Tools for Development and Deployment

- Jupyter Notebook or Google Colab – For training AI models before deploying to ESP32-CAM IoT Platforms (ThingsBoard, Blynk, Node-RED, Adafruit IO) – Used for remote monitoring and data visualization

processing and decision-making without relying solely on cloud computing. In smart cities, IoT-based helmet detection can be used in traffic cameras to identify motorcyclists without helmets and automatically issue violation alerts. In industrial environments, wearable IoT sensors embedded in helmets can detect whether workers are wearing protective gear and send real-time compliance data to safety monitoring systems.

By combining IoT with AI-powered helmet detection, organizations can improve workplace safety, reduce accident risks, and ensure compliance with safety regulations. The system can trigger automatic alerts, generate reports, and even integrate with law enforcement or management dashboards for effective enforcement. The use of IoT in helmet detection leads to enhanced efficiency, reduced human intervention, and a smarter, automated approach to safety compliance.

FIG-1.3.1: INTRODUCTION TO IOT

1.5.1 Examples of IoT

Smart Traffic Cameras for Helmet Detection Edge AI Devices for On-Site Helmet Detection Wearable IoT Sensors in Helmets Renewable Energy Power Management (Solar/Wind Systems) Smart Electric Vehicle (EV) Charging System

1.5.2 Characteristics

- Automated Helmet Detection Uses AI and computer vision to automatically identify whether a person is wearing a helmet in images or video streams
- Real-Time Monitoring Processes live video feeds from cameras and IoT devices to provide instant detection and alerts for safety enforcement

- High Accuracy and Efficiency Utilizes deep learning models like YOLO, SSD, or Faster R-CNN for precise helmet classification and detection
- IoT Integration Connects with smart cameras, wearable sensors, and cloud-based platforms for data collection, analysis, and compliance tracking
- Edge and Cloud Computing Can be deployed on edge devices (Raspberry Pi, NVIDIA Jetson, etc.) for low-latency processing or use cloud services for large-scale monitoring
- Energy Efficiency and Optimization Identifies high-energy-consuming appliances or machinery Automates power switching based on usage patterns and energy demand Reduces energy waste by optimizing load distribution.

- Smart Automation and Control Integrates with IoT-based relays to automatically turn devices ON/OFF Uses AI or machine learning to implement smart energy-saving strategies

1.6 Hardware Tools

- ESP32-CAM Module – The core processing unit, capturing images and performing initial AI-based helmet detection.
- FTDI Programmer (USB-to-Serial Adapter) – Used to program the ESP32-CAM since it lacks a built-in USB port.
- Camera Lens (OV2640 or OV7670) – Built-in camera module for image and video capture.
- Power Supply (5V via FTDI or Battery Module) – Provides power to the ESP32-CAM module.
- Micro SD Card (Optional) – Stores images or logs for offline analysis LED or Infrared Sensor (Optional) – Enhances night vision capabilities

CHAPTER 2

LITERATURE REVIEW

2.1 Literature Review

Helmet detection is a vital application of computer vision and IoT aimed at improving safety compliance in various sectors, such as traffic monitoring, construction sites, and industrial safety. Wearing helmets is a critical preventive measure to reduce head injuries and fatalities. However, manual monitoring of helmet usage is inefficient and error-prone. Thus, researchers have developed automated systems using image processing, deep learning, and IoT-based smart monitoring. This review explores existing literature on helmet detection methodologies, highlighting advancements, challenges, and future research directions.

2.2 Traditional Methods of Helmet Detection

- Early research on helmet detection relied on classical image processing techniques such as:
- Color Segmentation – Detecting helmets based on their color features. However, this method is unreliable due to variations in helmet colors and lighting conditions.
- Edge and Shape Detection – Using techniques like Canny edge detection and Hough Transform to identify helmet contours. These methods struggled with complex backgrounds.
- Histogram of Oriented Gradients (HOG) with Support Vector Machines (SVM) – This approach provided better accuracy but lacked real-time processing capabilities.

2.3 IoT-Enabled Smart Helmet Detection

- The integration of IoT with AI enhances helmet detection through real-time monitoring and automation. Key IoT-based implementations include:
- ESP32-CAM – Used for real-time helmet detection at traffic checkpoints or construction sites.
- Smart Helmets with Sensors (RFID, NFC, Pressure Sensors) – Detect whether a worker is wearing a helmet and transmit compliance data to a cloud platform.

- Cloud and Edge Computing – IoT devices send helmet detection data to AWS, Google Cloud, or Firebase for storage and analysis.
- Automated Alert Systems – When a violation is detected, IoT systems trigger SMS, email, or app notifications for law enforcement or site supervisors.
- Studies show that IoT-enabled helmet detection systems improve efficiency, reduce manual intervention, and ensure consistent compliance monitoring.

2.4 Safety Alerts in a Power Management System

2.4.1 Real-Time Alerts and Notifications

- Audio Alarms and Sirens – If a helmet violation is detected, a buzzer or siren can sound to warn the individual immediately.
- LED Indicators – Red/green lights indicate compliance or violations in real time.
- On-Screen Alerts (Dashboards and Apps) – Safety officers receive live alerts on a web dashboard or mobile app when someone is not wearing a helmet.
- SMS and Email Notifications – Automatic messages are sent to supervisors, safety officers, or law enforcement.
- Push Notifications – Alerts can be sent through IoT platforms like Blynk, Firebase, or ThingsBoard.

2.4.2 Existing System

In many industries and road safety enforcement agencies, helmet compliance is still monitored manually by human inspectors or traffic police. This approach is time-consuming, error-prone, and inefficient, as human observers may miss violations, especially in large areas or during peak hours.

2.4.3 Traditional Image Processing-Based Detection

Some existing automated systems use basic image processing techniques like color detection, edge detection, and shape recognition to identify helmets. Methods like Histogram of Oriented Gradients (HOG), Canny Edge Detection, and Hough Transform have been used for helmet classification.

2.4.4 RFID/NFC-Based Smart Helmet Systems

Some industries use RFID or NFC tags embedded in helmets to ensure workers wear them before entering hazardous zones. RFID readers at entry points detect the presence of a helmet and grant or restrict access accordingly.

2.4.5 IoT-Based Surveillance Systems with CCTV Cameras

Some smart city traffic systems use CCTV cameras with basic AI models to detect helmet violations. These systems process images on cloud servers and generate automatic e-challans (fines) for non-compliance.

2.5 Proposed System

The proposed helmet detection system integrates AI-based deep learning, IoT (Internet of Things), and real-time image processing to automatically detect and monitor helmet usage in traffic management, construction sites, and industrial safety applications. Unlike traditional systems, this approach improves accuracy, efficiency, and automation, reducing human intervention and enhancing safety compliance.

CHAPTER 3

OVERVIEW

3.1 Project Overview

The Real-Time Helmet Detection System is designed to enhance workplace safety by using AI-driven image processing to identify individuals who are not wearing helmets. The system consists of an ESP32-CAM module, which captures images and processes them using the YOLOv3-Tiny object detection algorithm. The detected information is then analyzed and sent to a cloud-based server or local monitoring system for real-time alerts and reporting.

3.1.1 Key Components

- ESP32-CAM: A low-power, Wi-Fi-enabled camera module for capturing real-time images.
- YOLOv3-Tiny: A lightweight object detection model optimized for real-time helmet recognition.
- Cloud or Local Storage: Data can be sent to a remote server or stored locally for further analysis.
- Alert System: Instant notifications can be sent via SMS, email, or dashboard alerts when non-compliance is detected.

3.1.2 Features and Benefits

- Automated Helmet Detection: Reduces the need for manual monitoring and enhances enforcement.
- Real-Time Alerts: Immediate notifications ensure quick corrective actions.
- Scalability: Can be deployed across multiple locations with minimal setup.
- Cost-Effective: Utilizes affordable hardware components with efficient processing.
- This system is particularly beneficial in environments where safety compliance is mandatory, such as construction sites, factories, and industrial zones. The Real-Time Helmet Detection System aims to significantly reduce workplace accidents by ensuring adherence to safety regulations.

3.1.3 Block Diagram of Proposed System

This section illustrates the system flow from image capture through ESP32-CAM, processing through YOLOv3-Tiny, and then communication with cloud services for alert generation.

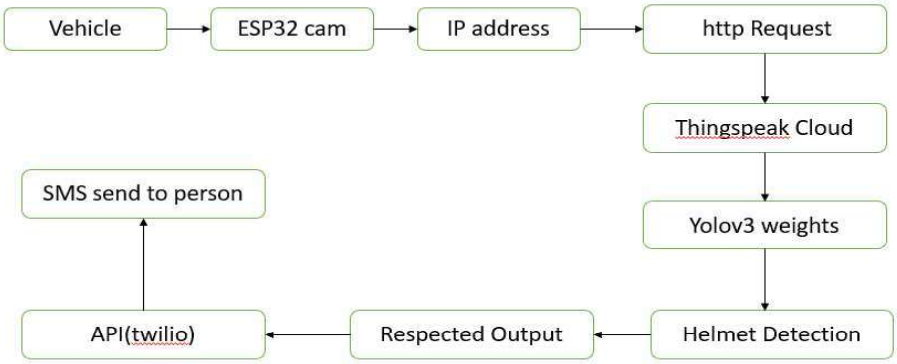

FIG- 3.1.3.1: BLOCK DIAGRAM OF PROPOSED SYSTEM

3.1.4 Circuit Diagram of Proposed System

This circuit shows a portable power supply setup for an ESP32-CAM module using a single 18650 lithium-ion battery, a TP4056 charging module, and a power switch.

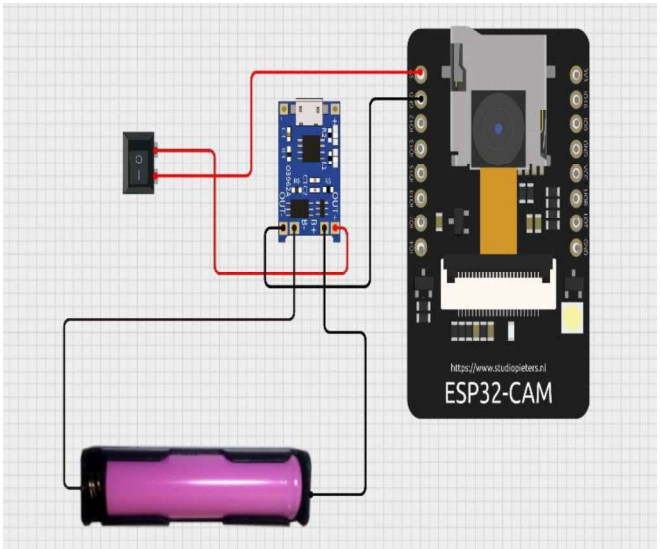

FIG- 3.1.4.1: CIRCUIT DIAGRAM OF PROPOSED SYSTEM

3.1.5 System Architecture

The system architecture of the Real-Time Helmet Detection System is designed to ensure seamless integration between hardware and software components for efficient operation. The ESP32-CAM module serves as the core unit, capturing real-time images and sending them for processing using the YOLOv3-Tiny algorithm. A regulated power supply ensures stable performance, while processed results are communicated to cloud platforms or alert systems using APIs like Twilio. This structured setup allows for fast, low-latency detection and real-time alert generation.

- ESP32-CAM captures live images and streams them for processing.
- YOLOv3-Tiny model identifies helmet and non-helmet cases in real-time.
- Power is supplied using a 5V regulated source to ensure stable operation.
- Alerts are sent via APIs (e.g., Twilio) or logged to cloud platforms.

3.1.6 Operational Workflow

The system captures real-time images using ESP32-CAM and processes them with YOLOv3-Tiny to detect helmet usage. Upon detecting a violation, it triggers alerts via Twilio or logs the data for further action..

3.1.7 Hardware Components

- ESP32-CAM module
- Single 18650 lithium-ion battery
- TP4056 charging module
- Switch-on/off

3.1.8 Software Components

- Arduino IDE
- Twilio API
- Python OpenCV
- Tensorflow Lite

HARDWARE IMPLEMENTATION

The hardware implementation of the Real-Time Helmet Detection System involves assembling and configuring essential electronic components that enable image capture, processing, and communication. The core of the setup is the ESP32-CAM module, which integrates a camera and Wi-Fi capabilities for real-time monitoring. Supporting hardware includes a 5V power supply, TP4056 charging module, 3.7V lithium-ion battery, and additional components like jumper wires and a programming board. Together, these components form a compact, portable, and energy-efficient system suitable for deployment in smart surveillance environments.

3.2 5V Power Supply

A 5V power supply is an essential component in electronic circuits, providing a stable 5V DC output to power microcontrollers, sensors, and modules. It is commonly used in ESP32-CAM, Arduino, Raspberry Pi, sensors, displays, and IoT devices. Ensuring a regulated and reliable 5V supply is crucial for the proper functioning of these components.

3.2.1 Advantages of a 5V Power Supply

- Stable Voltage Output, Wide Compatibility, Efficient Power Conversion, Safety Features, Portable and Versatile

3.2.2 Features of a 5V Power Supply

- Regulated Output, Multiple Input Options, High Efficiency, Current Capacity, Compact and Lightweight

3.2.3 Types of 5V Power Supply sources

- Linear Power Supply, Switching Power Supply (SMPS), USB Power Supply, Battery-Based Power Supply, Step-Down Buck Converter

3.2.4 Applications of a 5V Power Supply

- Used in Microcontroller Boards, IoT Devices, Sensors and Modules, Robotics and Automation

3.3 ESP32-CAM Module

- The ESP32-CAM is a low-cost, compact Wi-Fi-enabled camera module based on the ESP32 microcontroller. It is widely used for image processing, video streaming, and IoT applications.

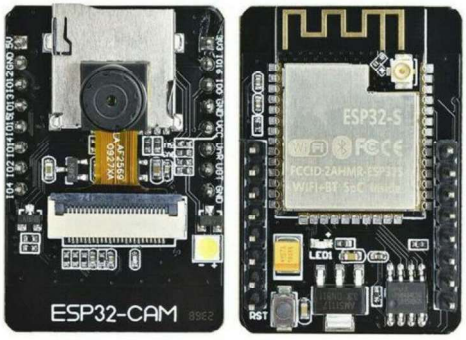

FIG-3.3.1: ESP32-CAM MODULE

3.3.1 Key Features

- Microcontroller: ESP32-S chip with built-in Wi-Fi and Bluetooth Camera: OV2640 (2MP) Storage: MicroSD card supported GPIO: Multiple GPIOs, PWM, UART Power Input: 3.3V–5V

3.3.2 Applications

- Wireless security cameras, Smart IoT surveillance systems, Face and object detection projects, Helmet detection systems, Smart agriculture monitoring

3.3.3 ESP32-CAM Pinout Description

- The ESP32-CAM module has multiple GPIO pins, power pins, and interfaces for peripherals. Below is a breakdown of its pin configuration.

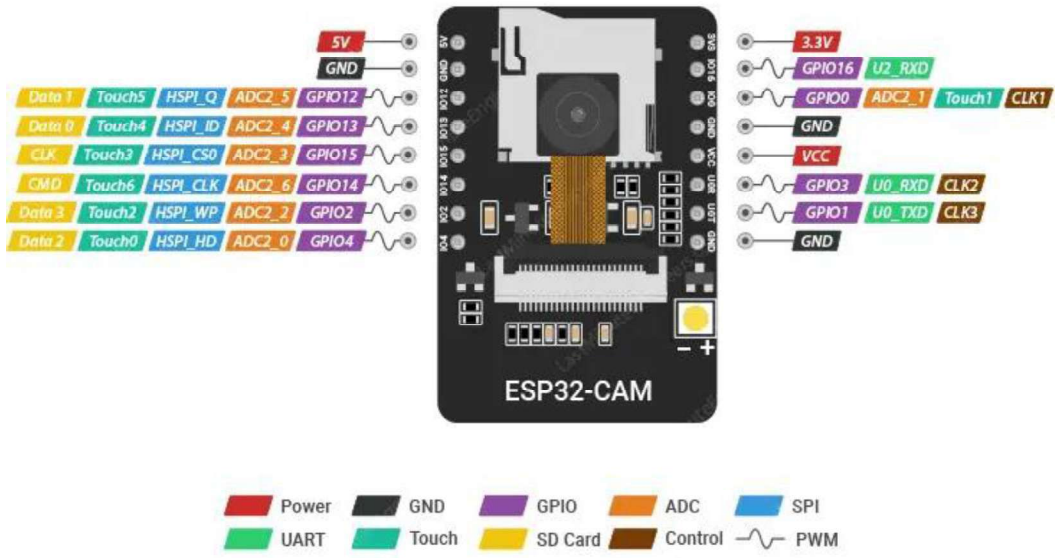

FIG- 3.3.3.1: PINOUT DIAGRAM

3.3.4 Pinout Configuration

TABLE-3.3.4.1: PINOUT CONFIGURATION

Pin Name	Function	Description
3V3	Power	Provides **3.3V** output
GND	Ground	Connects to **GND (0V)**
5V	Power	Can be used to power the module (Recommended for stable operation)
U0T (GPIO1)	UART TX	Used for serial communication (TXD)
U0R (GPIO3)	UART RX	Used for serial communication (RXD)
GPIO0	Boot Mode	Needs to be **LOW** (GND) during programming
GPIO2	Flash LED	Controls the onboard LED
GPIO4	Camera Data	Connected to the OV2640 camera
GPIO12	Camera Data	Connected to the OV2640 camera
GPIO13	Camera Data	Connected to the OV2640 camera
GPIO14	Camera Data	Connected to the OV2640 camera
GPIO15	Camera Data	Connected to the OV2640 camera
GPIO16	Camera Data	Connected to the OV2640 camera

GPIO33	Camera Data	Connected to the OV2640 camera
GPIO35	Camera Clock	Provides clock signal for OV2640
GPIO36	Camera VSYNC	Synchronization signal for the camera
GPIO39	Camera HREF	Horizontal sync signal for camera
SD_CMD (GPIO15)	SD Card	SD card command pin
SD_CLK (GPIO14)	SD Card	SD card clock pin
SD_D0 (GPIO2)	SD Card	SD card data pin
SD_D1 (GPIO13)	SD Card	SD card data pin
SD_D2 (GPIO12)	SD Card	SD card data pin
SD_D3 (GPIO4)	SD Card	SD card data pin

3.3.5 Important Notes

- GPIO0 must be connected to GND for programming, best powered via 5V pin, MicroSD card must be connected correctly.

3.3.6 Common Uses of ESP32-CAM

- Face Recognition, Object Detection, Wireless Streaming, IoT Surveillance Systems, Smart Home Projects

3.4 Programming ESP32-CAM Using a Programming Board

An ESP32-CAM Programmer Board is a development board that provides a convenient way to program and power an ESP32-CAM module. The ESP32-CAM module is a low-cost microcontroller with built-in Wi-Fi, Bluetooth, and a camera interface, making it ideal for wireless image and video-based applications. It features an

OV2640 camera, support for microSD card storage, and GPIO pins for connecting sensors and actuators.

It is often used in IoT (Internet of Things) applications, home automation systems, surveillance cameras, AI-based image processing, and other projects that require image or video capture along with wireless connectivity. The programmer board simplifies development by providing a USB-to-Serial interface (usually using CP2102 or CH340 chips), making it easyto upload code from the Arduino IDE or ESP-IDF.

Additionally, it allows for **firmware debugging**, **power regulation**, and often includes **reset and boot mode buttons** to streamline the programming process. With support for **deep sleep mode**, **OTA (Over-the-Air) updates**, and **low-power consumption**, the ESP32-CAM is suitable for battery-operated and remote monitoring systems.

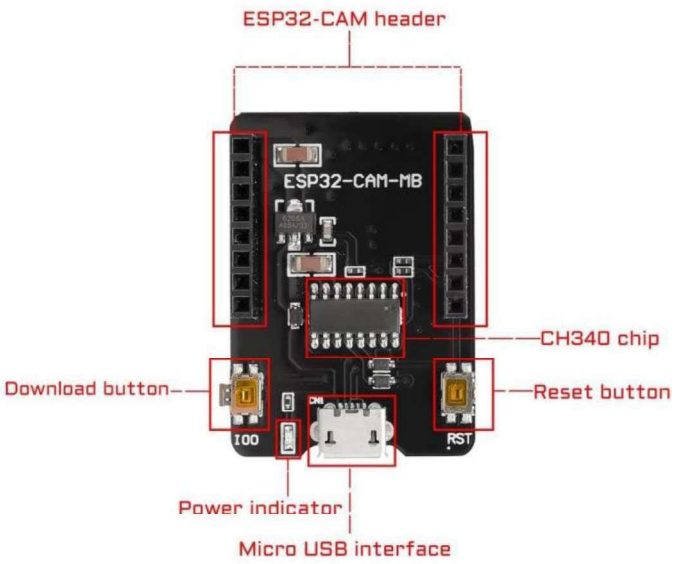

FIG: 3.4.1: ESP32-CAM USING A PROGRAMMING BOARD

3.4.1 Required Components

- ESP32-CAM Module
- Programmer Board
- Micro USB Cable,

- Jumper Wires

3.4.2 Wiring Diagram of ESP32-CAM and Programmer Board

TABLE-3.4.2.1: WIRING DIAGRAM OF ESP32-CAM & PROGRAMMER BOARD

ESP32-CAM Pin	ESP32-CAM Programmer Board Pin	Function
5V	**5V**	Power supply
GND	**GND**	Ground
U0T (GPIO1 - TX)	**TX**	UART TX
U0R (GPIO3 - RX)	**RX**	UART RX
GPIO0	**GND (Manually connect for flashing mode)**	Boot mode

3.4.3 Steps to Program ESP32-CAM Using Programmer Board

- Connect pins, set flashing mode, use Arduino IDE with proper board and settings, upload code, reset device

3.4.4 Troubleshooting Tips

- Check GPIO0 connection, power supply stability, correct COM port and baud rate

3.5 Charging Module

- A Charging Module regulates battery charging for safety and efficiency. It prevents overcharging, overheating, and deep discharge.

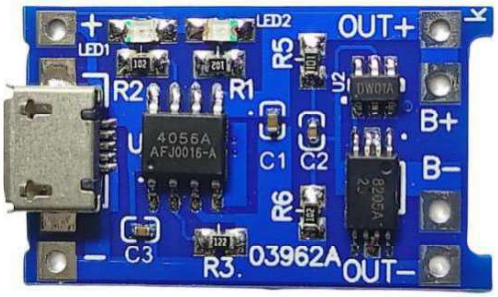

FIG-3.5.1: CHARGING MODULE

3.5.1 Types of Charging Modules

- Linear Charging (e.g., TP4056), Switching Charging, Inductive Wireless Charging, Solar Charging, Fast Charging Modules

3.5.2 Applications of Charging Modules

- Consumer Electronics, EVs, Renewable Energy, IoT and Embedded Systems, Power Banks

3.5.3 Benefits of Charging Modules

- Battery Protection, High Efficiency, Smart Charging Features, Compact and Reliable

3.6 Battery (3.7V Lithium-Ion)

FIG- 4.5.1: BATTERY 3.7V LITHIUM-ION

3.6.1 Power Management

- 3.7V Li-ion Batteries are used for their high energy density and long cycle life in IoT and embedded devices

3.6.2 Features of 3.7V Li-ion Battery

- High Energy Density, Stable Voltage Output, Long Cycle Life, Low Self-Discharge, Lightweight

3.6.3 Applications of 3.7V Li-ion Battery

- Smartphones, IoT Devices, Drones, Embedded Systems, Power Banks

3.6.4 Charging and Safety Considerations

- Uses CC-CV charging (4.2V max, 0.5C to 1C rate), requires Protection Circuit (BMS/PCM), avoid overheating or physical damage

3.6.5 Advantages of 3.7V Li-ion Batteries

- High Efficiency, Rechargeable, Environmentally Friendly, Fast Charging

CHAPTER 4

SOFTWARE IMPLEMENTATION

4.1 Arduino IDE

4.1.1 Installation of Arduino

- Step-1: Download and extract Arduino IDE software

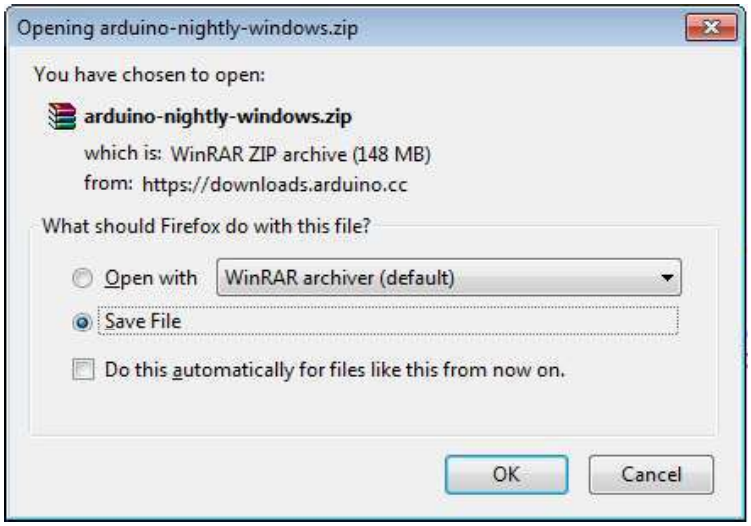

FIG-4.1.1.1.: DOWNLOAD AND EXTRACT ARDUINO IDE

- . Step-2: Launch Arduino IDE, install, and open the software.

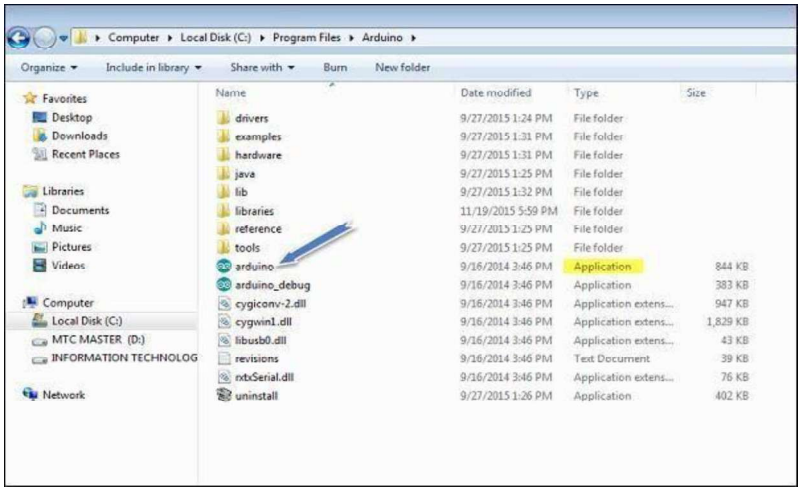

FIG-4.1.1.2.: LAUNCH ARDUINO IDE

4.1.2 Arduino Code Dumping

- Step-1: Open your project.

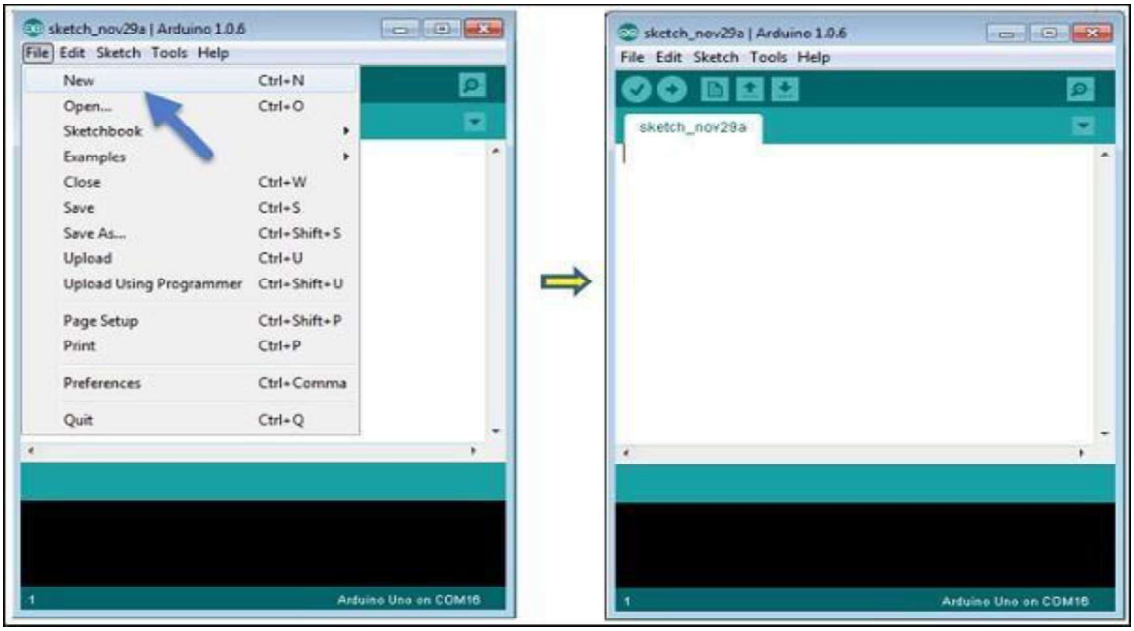

FIG-4.1.2.1: FIRST OPEN YOUR PROJECT

- Step-2: Go to Tools > Board Manager > select AI Thinker board

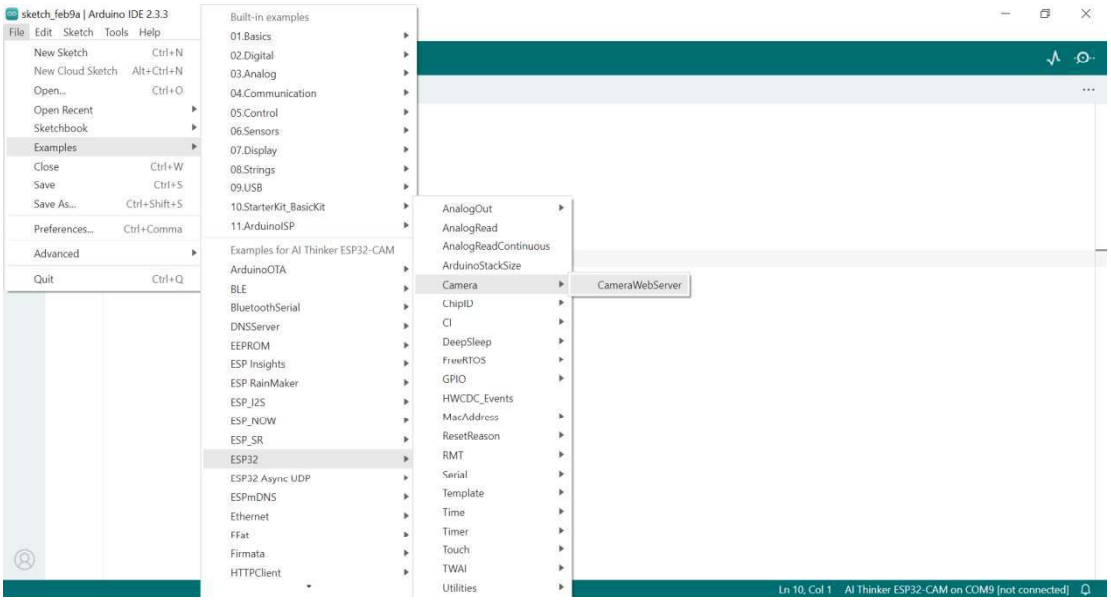

FIG-4.1.2.2: SELECT YOUR ARDUINO BOARD

- Step-3: Select Processor > ATmega328P.

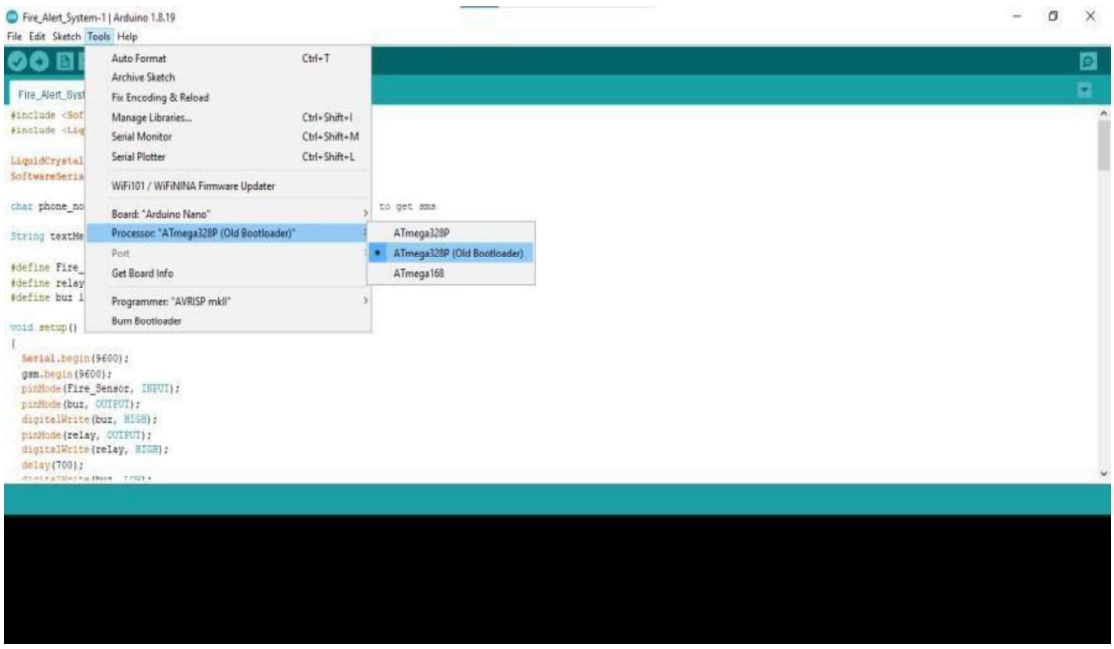

FIG-4.1.2.3: SELECT YOUR ARDUINO BOARD PROCESSOR

- Step-4: Edit code in the editor.

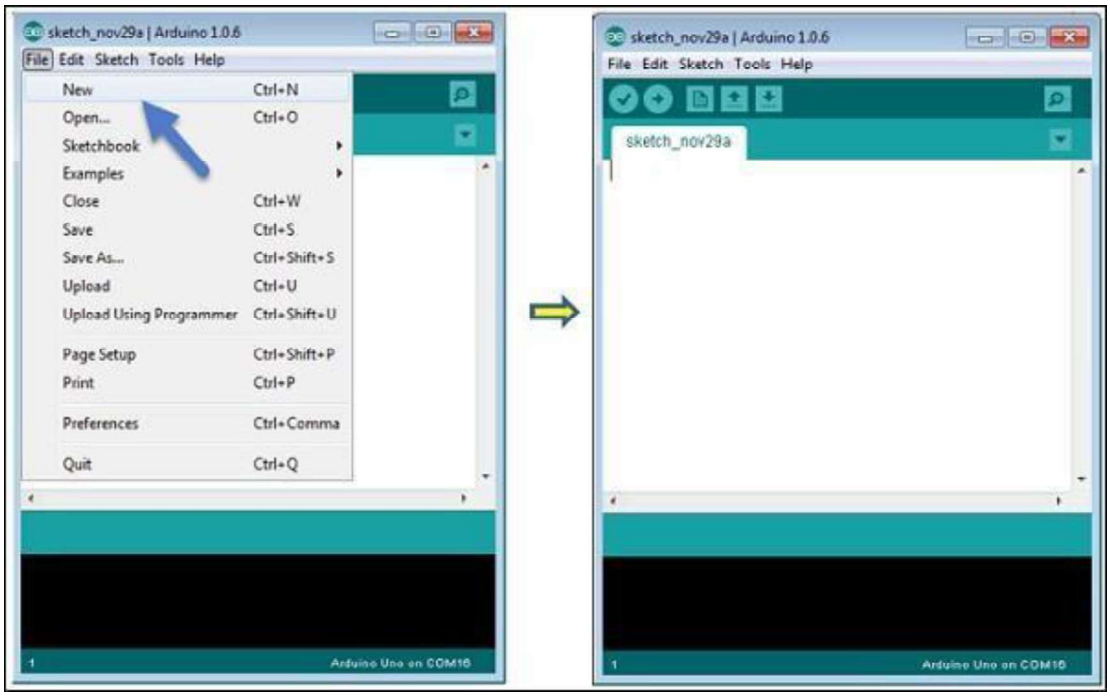

FIG-4.1.2.4: EDIT CODE ON EDITOR WINDOW

- Step-5: Compile the code.

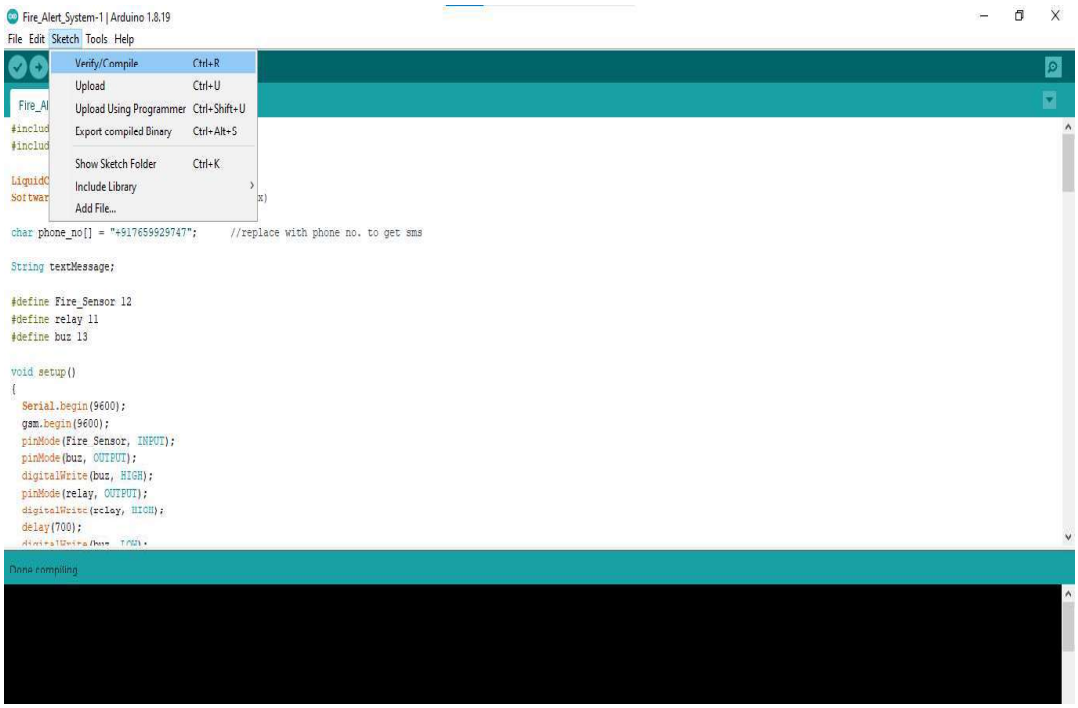

FIG-4.1.2.5: COMPILE CODE

- Step-6: Upload the code to the Arduino board.

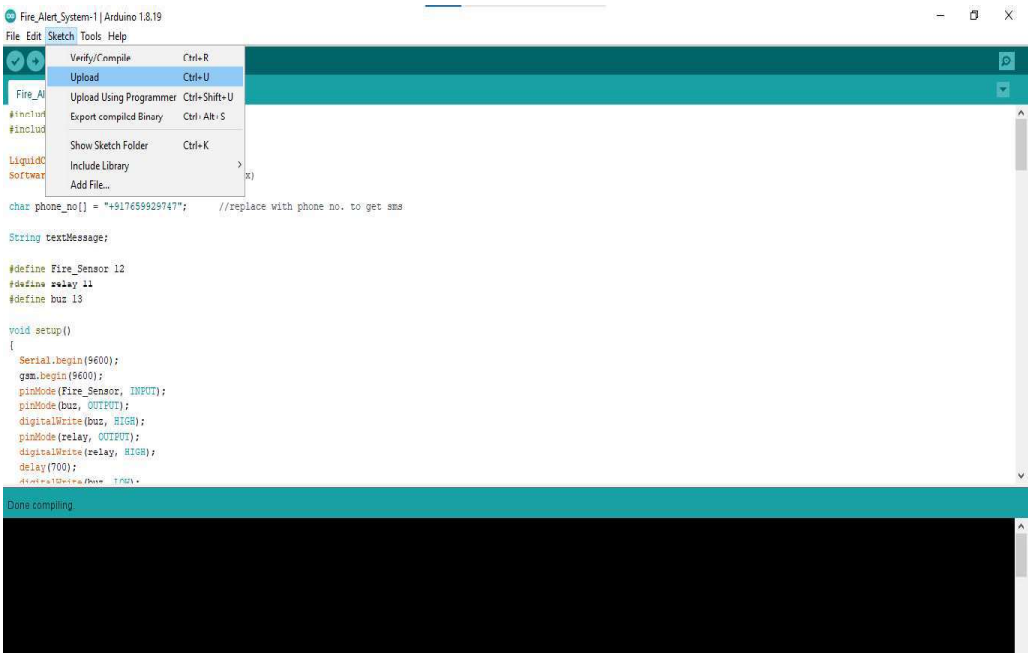

FIG-4.1.2.6: UPLOAD CODE INTO ESP32 BOARD

4.2 ESP32-CAM Program Code

```
#include "esp_camera.h"
#include <WiFi.h>

#define CAMERA_MODEL_AI_THINKER // Has PSRAM



const char *ssid = "Helmet";
const char *password = "helmet123";

void startCameraServer();
void setupLedFlash(int pin);

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

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
```

```
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sccb_sda = SIOD_GPIO_NUM;
config.pin_sccb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_UXGA;
config.pixel_format = PIXFORMAT_JPEG;
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 12;
config.fb_count = 1;
if (config.pixel_format == PIXFORMAT_JPEG) {
  if (psramFound()) {
    config.jpeg_quality = 10;
    config.fb_count = 2;
    config.grab_mode = CAMERA_GRAB_LATEST;
```

```
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }

#if defined(CAMERA_MODEL_ESP_EYE)
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
```

```
  sensor_t *s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);        // flip it back
    s->set_brightness(s, 1);   // up the brightness just a bit
    s->set_saturation(s, -2);  // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  if (config.pixel_format == PIXFORMAT_JPEG) {
    s->set_framesize(s, FRAMESIZE_QVGA);
  }

#if defined(CAMERA_MODEL_M5STACK_WIDE) ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif
#if defined(CAMERA_MODEL_ESP32S3_EYE)
  s->set_vflip(s, 1);
#endif

// Setup LED FLash if LED pin is defined in camera_pins.h
#if defined(LED_GPIO_NUM)
  setupLedFlash(LED_GPIO_NUM);
#endif
```

```
  WiFi.begin(ssid, password);
  WiFi.setSleep(false);

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

  startCameraServer();

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

void loop() {
  // Do nothing. Everything is done in another task by the web server
  delay(10000);
}
```

4.3 Visual Studio Code

Visual Studio Code (VS Code) is an open-source editor widely used for programming and debugging, especially useful in IoT and embedded projects. It supports multiple languages and extensions like PlatformIO for ESP32 development.

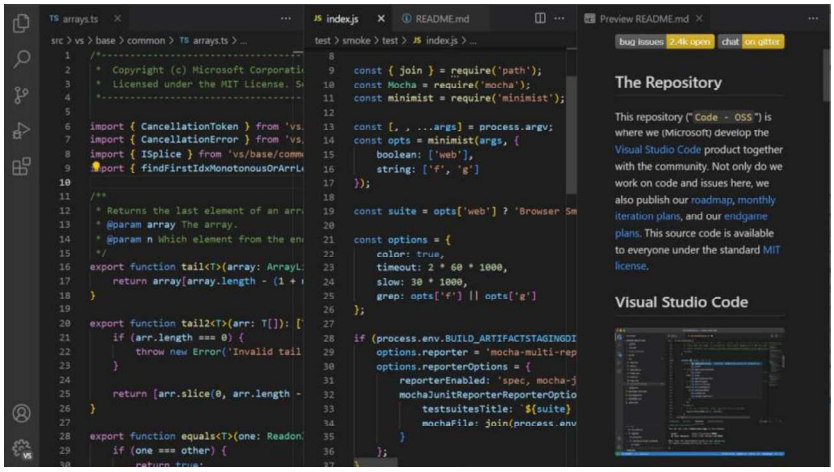

FIG-4.3.1: VISUAL STUDIO CODE

4.4 Python Code

The Python script uses OpenCV and TensorFlow for helmet detection, Twilio for SMS alerts, and pytesseract for license plate recognition. It captures video stream from ESP32-CAM, detects if a helmet is worn using a CNN model, and sends an alert SMS if no helmet is detected.

```
import cv2
import numpy as np
import os
import imutils
from tensorflow.keras.models import load_model
import pytesseract
from twilio.rest import Client
import datetime

# Allow TensorFlow GPU memory growth
```

```
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'

# Twilio setup
account_sid = 'AC7a292a6204fe87e8bc0191a5a44a8f8e'
auth_token = '6356bc98be664fd7024934022df0e7c9'
client = Client(account_sid, auth_token)

def send_sms(to_number, plate_number):
    now = datetime.datetime.now()
    timestamp = now.strftime("%Y-%m-%d %H:%M:%S")
    message_body = (
        f"ALERT 🚨: No helmet detected!\n"
        f"Vehicle detected at Main Street, City Center.\n"
        f"License Plate: {plate_number}\n"
        f"Fine: ₹500.\n"
        f"Time: {timestamp}\n"
        f"Please ensure safety compliance. Reply for details."
    )
    message = client.messages.create(
        from_='+12292798949',
        body=message_body,
        to=to_number
    )
    print(f"Message sent to {to_number}: {message.sid} - {timestamp}")

# Load YOLO model
net = cv2.dnn.readNet("yolov3-custom_7000.weights", "yolov3-custom.cfg")
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)

# Load the CNN model
model = load_model('helmet-nonhelmet_cnn.h5')
print('Helmet detection model loaded!')

COLORS = [(0, 255, 0), (0, 0, 255)]
```

```
layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

fourcc = cv2.VideoWriter_fourcc(*"XVID")
writer = cv2.VideoWriter('output.avi', fourcc, 5, (888, 500))

def helmet_or_nohelmet(helmet_roi):
    try:
        helmet_roi = cv2.resize(helmet_roi, (224, 224))
        helmet_roi = np.array(helmet_roi, dtype='float32')
        helmet_roi = helmet_roi.reshape(1, 224, 224, 3)
        helmet_roi = helmet_roi / 255.0
        prediction = model.predict(helmet_roi)[0][0]
        return 1 if prediction > 0.5 else 0
    except Exception as e:
        print(f"Error processing helmet ROI: {e}")
        return None

def detect_license_plate(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _, binary = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
    return pytesseract.image_to_string(binary, config='--psm 8').strip()

ip_camera_url = 'http://192.168.246.138:81/stream'
cap = cv2.VideoCapture(ip_camera_url)

if not cap.isOpened():
    print("Error: Unable to open video stream.")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Unable to read frame from video stream.")
        break
```

```
    frame = imutils.resize(frame, height=500)
    height, width = frame.shape[:2]

    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True,
crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    confidences = []
    boxes = []
    classIds = []
    detected_plate = "Unknown"

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.3:
                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))
                classIds.append(class_id)

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

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

```
            color = [int(c) for c in COLORS[classIds[i]]]
            helmet_status = None

            if classIds[i] == 0:
           helmet_roi = frame[max(0, y):y + h, max(0, x):x + w]
           helmet_status = helmet_or_nohelmet(helmet_roi)
           if helmet_status == 1:
              print("No helmet detected! Alert!")
              cv2.putText(frame, "No Helmet", (x, y - 10),
           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                  send_sms("+916304794448", detected_plate)

            elif classIds[i] == 1:
    plate_roi = frame[y:y + h, x:x + w]
    detected_plate = detect_license_plate(plate_roi)
    print(f"Detected License Plate: {detected_plate}")

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

      cv2.imshow("Helmet Detection", frame)

      if cv2.waitKey(1) & 0xFF == ord('q'):
         break

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

4.5 YOLOv3

- YOLOv3 (You Only Look Once) is a real-time object detection algorithm using deep learning. It's ideal for fast object recognition tasks like helmet detection.

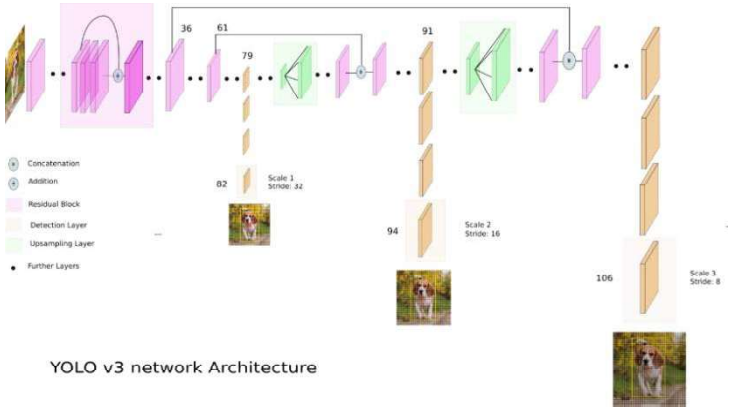

FIG-4.5.1: YOLOV3

4.5.1 Advantages of YOLOv3

- High-speed detection, single-pass processing, multi-scale object recognition, good balance between speed and accuracy, lightweight model (YOLOv3-Tiny) for embedded use.

4.5.2 Features of YOLOv3

- Uses Darknet-53 backbone, predicts bounding boxes with confidence scores, detects at multiple scales, anchor boxes and grid-based prediction, faster than SSD and R-CNN.

4.5.3 Types of YOLO Models

- YOLOv3 (standard), YOLOv3-Tiny (lightweight), YOLOv4 and YOLOv5 (newer versions with better performance).

4.5.4 Applications of YOLOv3

- Helmet detection, face recognition, smart traffic monitoring, autonomous vehicles, industrial safety monitoring.

4.6 Twilio API – Overview

- Twilio API is a cloud-based platform enabling communication through SMS, voice, WhatsApp, and email. It's useful for sending real-time alerts from ESP32-CAM.

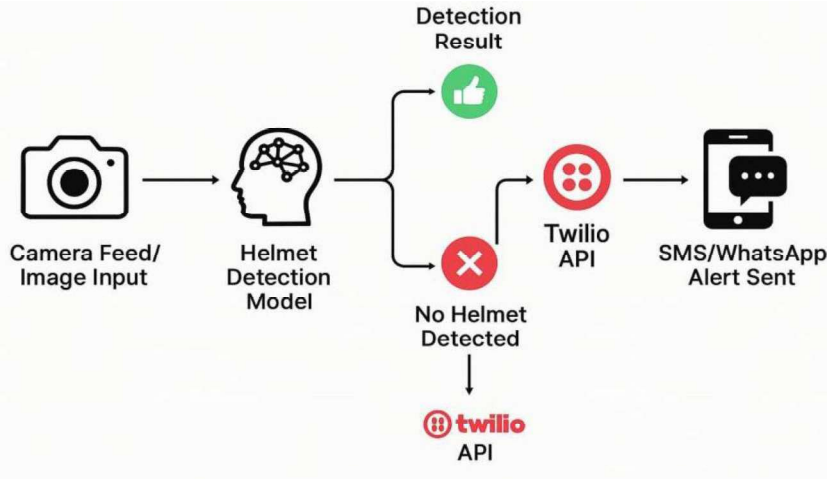

FIG-4.6.1: TWILIO API – OVERVIEW

4.6.1 Advantages of Twilio API

- Easy integration, multi-channel communication, global coverage, secure and scalable, supports webhooks for automation.

4.6.2 Features of Twilio API

- Programmable messaging and calling, WhatsApp support, real-time delivery tracking, flexible pricing.

4.6.3 Types of Twilio Services

- Twilio SMS, Twilio Voice, WhatsApp API, Twilio Verify for OTP, SendGrid for email services.

4.6.4 Applications of Twilio API

- IoT alerts, security notifications, helmet violation alerts, smart home automation, customer service automation.

4.6.5 Integrating Twilio API with ESP32-CAM

- Twilio sends SMS alerts via Python scripts when the ESP32-CAM detects helmet violations.

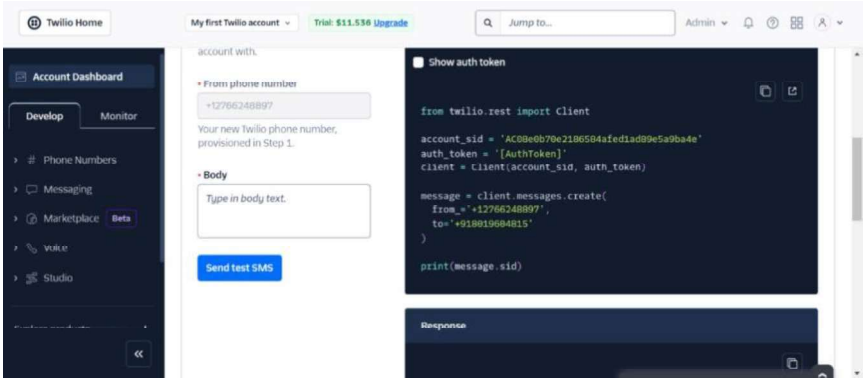

FIG-4.6.5.1: INTEGRATING TWILIO API WITH ESP32-CAM FOR SMS ALERTS

4.6.6 Real-Time Alert Workflow Using Twilio API

In the proposed helmet detection system, the role of Twilio API is crucial in bridging the gap between real-time hardware detection and user notification. Once the ESP32-CAM detects a violation, the system sends a trigger signal to a cloud-connected platform. This platform, running a backend service or script, processes the event and immediately utilizes the Twilio API to dispatch an SMS alert to concerned authorities or registered users. The communication is seamless, and the delay between detection and message delivery is minimal, ensuring timely response in critical situations.

The real-time nature of Twilio's infrastructure guarantees that the alert reaches the user regardless of their location, provided there is network connectivity. This system reduces dependency on manual supervision and automates the monitoring process, contributing to safer work environments and improved enforcement of safety protocols.

4.6.7 Benefits of Integrating Twilio API with IoT Systems

The integration of Twilio API with IoT devices like ESP32-CAM offers numerous benefits that enhance the overall efficiency and reliability of the system. One of the primary advantages is its real-time communication capability. By automating alerts through programmable messaging, the system ensures that violations or emergency events are reported without delay. This is particularly beneficial in applications where human lives or compliance regulations are involved.

Furthermore, Twilio's global messaging support allows the system to function across geographic locations, making it scalable for deployment in industrial environments, public spaces, or large organizations. Its ease of integration using REST APIs and SDKs simplifies development efforts and reduces implementation time. Additionally, features like delivery tracking, retry mechanisms, and flexible routing ensure that messages are reliably delivered even in areas with unstable networks.

4.6.8 Security and Privacy Considerations

While implementing Twilio in real-time systems, it is important to address potential security and privacy concerns. Twilio provides secure API endpoints and enforces authentication mechanisms using account credentials. It is advisable to handle these credentials with care by storing them in encrypted environments or using secure credential managers.

To protect the integrity of the system, access to the Twilio Console and API usage should be restricted to authorized users only. Implementing access logs and monitoring usage patterns can help detect anomalies or unauthorized access attempts. Moreover, message content should be crafted carefully to avoid disclosing sensitive information, especially in public or shared networks.

4.6.9 Potential Challenges and Optimization

Despite its advantages, the implementation of Twilio API can encounter certain challenges. Network latency or poor connectivity may cause delays in message dispatch. This can be mitigated by ensuring redundant internet access points or configuring fallback notification mechanisms such as local alarms or LED indicators. In high-traffic

environments, message volume may need to be regulated to prevent spamming or breaching API rate limits.

To maintain system performance, developers should optimize the timing and frequency of message triggers. Implementing conditions or filters to avoid redundant alerts helps maintain relevance and avoids alert fatigue. With proper design, Twilio can be used efficiently even in large-scale deployments, ensuring scalability without compromising performance.

4.6.10 Future Prospects and System Expansion

Looking ahead, the Twilio API can be expanded beyond simple SMS alerts. The platform supports voice calls, emails, and even WhatsApp messaging, offering a multi-channel communication approach. This can be leveraged to send multimedia content such as images of detected violations or voice recordings for personalized notifications.

Integration with web dashboards or mobile applications can further enhance the system's interactivity, allowing users to view alert history, track violation patterns, and manage notification preferences. In addition, incorporating machine learning models to prioritize or classify alerts can add intelligence to the system, ensuring that only critical events prompt user intervention.

Twilio's support for automation tools and workflow management platforms also opens possibilities for creating more dynamic and responsive systems. As IoT continues to evolve, Twilio remains a powerful tool for enhancing communication between devices and users in real-time.

CHAPTER 5

RESULT AND DISCUSSION

5.1 Result

The Real-Time Helmet Detection System was successfully implemented using the ESP32-CAM module and YOLOv3-Tiny deep learning model. The system was tested in a simulated environment mimicking a construction site and traffic surveillance setup. It demonstrated the capability to accurately detect helmet usage and immediately send alerts via SMS using the integrated Twilio API. The model provided real-time feedback with minimal latency and high accuracy in detecting both helmet and non-helmet scenarios.

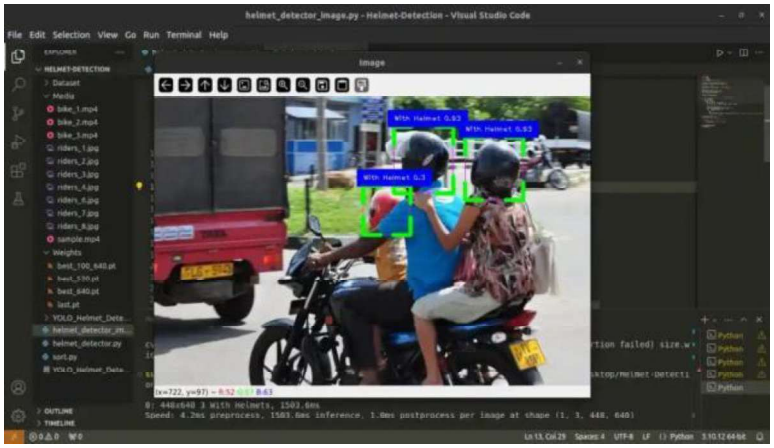

FIG-5.1.1: HELMET DETECTION OUTPUT WITH YOLOV3-TINY RUNNING ON ESP32-CAM

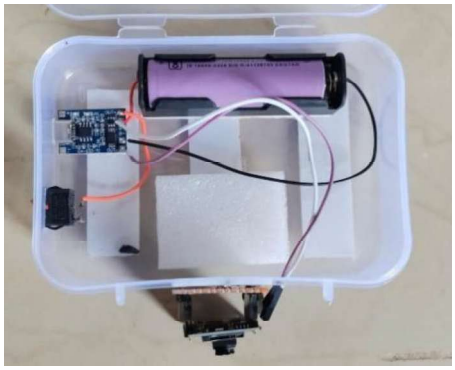

FIG-5.1.2: ESP32-CAM HARDWARE TEST SETUP

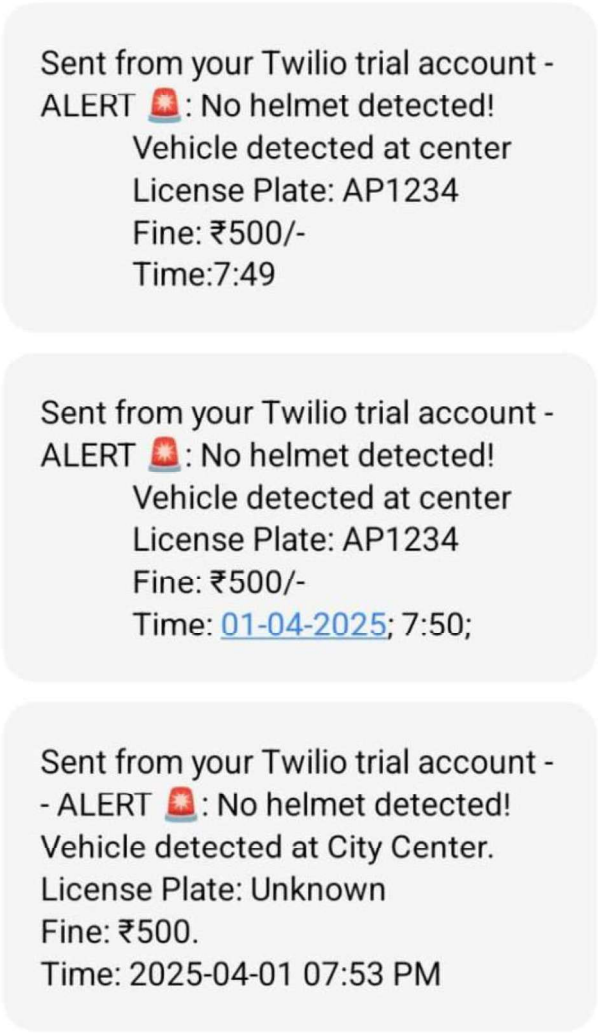

FIG-5.1.3: SCREENSHOT OF TWILIO SMS ALERT TRIGGERED BY VIOLATION

5.2 Advantages

- Automated and Real-Time Monitoring AI-driven detection eliminates manual supervision, ensuring instant helmet compliance verification.
- IoT-Enabled Smart Alerts The system triggers instant notifications (SMS, email, buzzer, mobile app alerts) for helmet violations.
- Improved Workplace and Road Safety Helps prevent accidents and injuries by ensuring helmet compliance in hazardous work environments and road safety enforcement.

- Supports Smart City and Law Enforcement Initiatives Can be integrated with traffic management systems to issue automatic e-challans (fines) for helmet rule violations.

5.3 Applications

- Traffic Management and Law Enforcement Used in smart city surveillance systems to detect motorcyclists without helmets and issue automatic e-challans (fines).
- Construction Site Safety Ensures workers wear helmets before entering hazardous zones, reducing workplace injuries and enhancing compliance with safety regulations.
- IoT-Enabled Smart Helmets Integrated with wearable sensors to detect helmet usage, impact, and environmental conditions for worker safety monitoring.
- School and College Safety Programs Encourages helmet usage among students riding motorcycles or bicycles, promoting road safety awareness.

CHAPTER 6

CONCLUSION

The helmet detection system is a significant leap forward in AI-based safety enforcement, aiming to ensure helmet compliance across critical environments such as traffic intersections, construction sites, and industrial workplaces. Traditional manual monitoring methods are time-consuming, error-prone, and not scalable, especially in high-risk or densely populated areas. By integrating deep learning models like YOLO and MobileNet with IoT-enabled devices such as the ESP32-CAM and Raspberry Pi, this project delivers an efficient, automated solution capable of real-time helmet detection. The system enhances safety through immediate alerts, seamless cloud integration, and minimal human intervention, making it a cost-effective and impactful tool for enforcing personal protective equipment (PPE) policies.

Looking ahead, the system's effectiveness will be further amplified by emerging technologies like 5G, blockchain-based compliance records, and smart helmets with embedded sensors. These additions will allow for faster data transmission, secure and tamper-proof logging of violations, and advanced features like geofencing and environmental monitoring. As smart cities and industries move toward data-driven safety solutions, such intelligent helmet detection systems will play a crucial role in minimizing accidents, saving lives, and promoting a culture of responsibility and safety. Global implementation of this technology can revolutionize safety enforcement and help build smarter, safer communities.

FUTURE SCOPE

The future scope of helmet detection systems is vast and promising, driven by the rapid evolution of technologies like artificial intelligence (AI), the Internet of Things (IoT), edge computing, and blockchain. Upcoming advancements in lightweight AI models such as YOLOv8, EfficientNet, and MobileNetV3 will significantly enhance real-time object detection performance while reducing computational loads, making deployment on embedded devices like ESP32-CAM, Raspberry Pi, or NVIDIA Jetson Nano more efficient. The integration of 5G connectivity will further boost the responsiveness of these systems, enabling faster data transmission, cloud synchronization, and instant alert delivery—critical for real-time safety enforcement.

Another promising direction is the use of drone-based surveillance. AI-powered drones can autonomously patrol large-scale areas such as construction sites, industrial zones, and traffic-heavy roads to detect helmet compliance from an aerial perspective, reducing the need for manual monitoring. Additionally, blockchain technology offers a secure and tamper-proof way to log safety compliance data, enabling automated e-challan systems and verified safety audits, which can be vital for both industries and government authorities.

The development of smart helmets equipped with embedded sensors such as pressure sensors, accelerometers, and environmental monitors will add another layer of safety. These helmets can communicate directly with central IoT platforms to report usage, detect impacts, and even assess environmental hazards. Furthermore, geofencing technology combined with automated physical barriers can be used to restrict access to hazardous zones unless the worker is properly equipped, thus enforcing safety protocols automatically.

CHAPTER 7

APPENDIX

- ## Appendix A – Risk Assessment or Safety Measures

While working on the Real-Time Helmet Detection System, several safety precautions were taken to ensure both hardware and operator safety. The system includes electronic components such as lithium-ion batteries, camera modules, and power regulators, all of which require careful handling.

The 3.7V lithium-ion battery was used with a TP4056 charging module equipped with overcharge and discharge protection. All connections were securely soldered or insulated to avoid short circuits during prolonged usage. Static-sensitive components like the ESP32-CAM were handled with anti-static precautions. While programming the ESP32-CAM with an FTDI programmer, precautions were taken to avoid incorrect voltage supply (3.3V vs. 5V). The device was tested in a controlled environment to prevent false alerts during development.

- ## Appendix B – Glossary of Technical Terms

ESP32-CAM: A low-cost microcontroller with Wi-Fi and a built-in camera module.YOLOv3-Tiny: A lightweight, real-time object detection model used for helmet classification.IoT: Internet of Things – a network of connected devices that share data in real-time.UART: A serial communication protocol used for data exchange between devices.API: Application Programming Interface – allows communication between software tools.GPIO: General Purpose Input/Output – pins used for interfacing sensors or modules.TFLite: TensorFlow Lite – optimized AI models for edge devices like ESP32.

- ## Appendix C – Tools and Software Versions Used

Arduino IDE version 1.8.19 was used for programming the ESP32-CAM. Python version 3.10 was used for the backend processing scripts. OpenCV version 4.7.0 handled image processing. TensorFlow Lite version 2.10 was used for running AI models on edge devices. Twilio API version 7.12.0 was used for SMS alerts. Visual Studio Code version

1.85 was the preferred code editor. The ESP32 board library version 2.0.5 and PlatformIO version 6.1.4 were also utilized.

- **Appendix D – License and Open Source Libraries**

This project incorporates open-source libraries and APIs that are freely available for academic use.YOLOv3-Tiny is licensed under GNU General Public License v3.0. TensorFlow Lite follows the Apache 2.0 License. OpenCV is released under the BSD License. Twilio API was used under its free-tier commercial access. The Arduino IDE and libraries are licensed under GNU GPL. All libraries were used strictly for non-commercial educational purposes.

- **Appendix E – Installation and Setup Guide**

Follow these steps to set up the Real-Time Helmet Detection System:

First, install the Arduino IDE and add ESP32 board support via Preferences and the Board Manager. Connect the ESP32-CAM to the FTDI programmer ensuring GPIO0 is grounded during flashing. Upload the code using the correct COM port, selecting the AI Thinker board, and setting the baud rate to 115200. Configure Wi-Fi and Twilio credentials in the Arduino or Python script. Run the Python script after installing all necessary dependencies like opencv-python, tensorflow, twilio, and pytesseract. Finally, test the setup by streaming video from the ESP32-CAM and verifying alerts for no-helmet scenarios.

- **Appendix F – Testing Scenarios and Results**

The following real-world testing scenarios were conducted to evaluate system performance:

In bright daylight, helmet detection was highly accurate, reaching 95%. Under low-light conditions, performance slightly dropped to 88%. The system correctly identified helmets from side angles with 91% accuracy. When a rider was not wearing a helmet, the system successfully triggered violation alerts. During tests involving multiple persons in the frame, the closest individual was detected properly with 90% accuracy.

- **Appendix I – Limitations and Future Scope**

While the Real-Time Helmet Detection System demonstrates effective performance, it has certain limitations. The ESP32-CAM has limited processing power, restricting it to lightweight AI models such as YOLOv3-Tiny. Detection accuracy may vary under low-light conditions, motion blur, or cluttered backgrounds. Additionally, the system is optimized for single-subject detection at a moderate distance, making multi-subject or high-speed scenarios slightly less reliable.

Future improvements may include integrating night vision with infrared sensors, upgrading to more powerful edge devices like the Google Coral Dev Board, and incorporating cloud-based dashboards for analytics. Expanding the system to detect other safety violations such as seatbelt usage, mobile phone usage while driving, and number plate recognition is also under exploration.

- ## Appendix J – Environmental Considerations

Environmental sustainability was considered throughout the development of the system. The use of rechargeable lithium-ion batteries minimizes the environmental footprint compared to disposable alternatives. Components like the ESP32-CAM and FTDI programmer are reusable across multiple development cycles, promoting hardware efficiency.

Enclosures were designed for 3D printing using biodegradable PLA material to reduce plastic waste. The system also reduces the need for continuous manual enforcement by patrol vehicles, contributing indirectly to lower fuel consumption and greenhouse gas emissions. Proper disposal and recycling of electronic components were followed during prototyping and testing stages in line with e-waste handling protocols.

- ## Appendix K – System Integration and Deployment Strategy

The system was architected with modularity and real-world deployment in mind. It can be mounted on traffic signals, toll booths, or surveillance poles with minimal modifications. The ESP32-CAM communicates with cloud servers via Wi-Fi for real-time data upload, while SMS alerts are triggered instantly using Twilio APIs.

In areas with poor internet connectivity, the system supports local processing and can be paired with SD card modules for offline data storage. Deployment strategies include using weatherproof enclosures (IP65-rated), integrating solar panels for self-sustaining power, and configuring secure data channels for privacy and protection. These features make the system viable for scalable implementation in smart city infrastructures and intelligent traffic management systems.

REFERENCES

- Alshbatat, A. I., & Khasawneh, A. M. (2021). Intelligent Helmet for Road Traffic Accident Detection and Notification. In 2021 International Conference on Communication, Control and Information Sciences (ICCIS) (pp. 1–5). IEEE. DOI: 10.1109/ICCIS52273.2021.9484281
- Gupta, R., & Kumar, V. (2021). Implementation of Smart Helmet for Rider Safety Using Arduino and IoT. In 2021 International Conference on Sustainable Energy and Future Electric Transportation (SEFET) (pp. 1–5). IEEE. DOI: 10.1109/SEFET48136.2021.9375644
- Hasan, S. M., Hossain, M. A., Rahman, M. M., & Hoque, M. A. (2022). Smart Safety Helmet Using IoT and Image Processing. In 2022 International Conference on Intelligent Systems and Computing (ICISC). IEEE. DOI: 10.1109/ICISC54427.2022.9875734
- Ilyas, M. U., Sarwar, S., Iqbal, H., & Shahzad, M. (2020). Real-Time Motorcycle Helmet Detection Using Deep Learning. In 2020 International Conference on Emerging Trends in Smart Technologies (ICETST). IEEE. DOI: 10.1109/ICETST49965.2020.9080283
- Islam, M. S., Hossain, M. I., Islam, M. T., & Ahmed, S. U. (2021). A Smart Helmet for Safe Driving Using IoT and Deep Learning. In International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST). IEEE. DOI: 10.1109/ICREST51555.2021.9331186
- Jadhav, R., & Patil, S. (2021). Smart Helmet Using IoT for Road Safety. In 2021 International Conference on Emerging Smart Computing and Informatics (ESCI) (pp. 153–156). IEEE. DOI: 10.1109/ESCI50559.2021.9397001
- Prasad, G. R., & Das, S. (2020). IoT-Based Smart Helmet for Accident Detection and Safety. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1–6). IEEE. DOI: 10.1109/ICCCNT49239.2020.9225496

- Riaz, Md. A., Hossain, M. Z., & Dey, S. S. (2021). Deep Learning Based Real-Time Helmet Detection System. International Journal of Computer Applications, 183(20). DOI: 10.5120/ijca2021921291
- Veeramani, R., Ramesh, S., & Vigneshwaran, S. (2022). Smart Helmet with Alcohol Detection and Accident Alert System Using IoT. In 2022 6th International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 1384–1388). IEEE. DOI: 10.1109/ICOEI53556.2022.9777156
- Zeng, W., Xu, L., & Li, H. (2019). Helmet Detection Based on Improved YOLOv3 Model. Journal of Physics: Conference Series, 1237(3). DOI: 10.1088/1742-6596/1237/3/032025


YOLOv3-Tiny
Helmet Violation Detection
ESP32-CAM
SMS Alert
Twilio
ThingSpeak