Real Time Energy Monitoring System with AI Driven Automation

Dr S Sri Gowri [1], K. G. Venkata Krishna [2]
[1]*Professor, ECE department, SRK INSTITUTE OF TECHNOLOGY, Enikepadu, Vijayawada 521108*
[2] *Assistant Professor (C), Department of Electronics and Communication Engineering, Krishna University College of Engineering and Technology, Krishna University, Rudravaram, Machilipatnam.*

***Abstract*—This paper presents the design and implementation of an intelligent smart home IoT system that integrates embedded hardware, computer vision, and cloud-based alerting into a unified real-time monitoring and automation platform. The system is built around an ESP32 microcontroller, which functions as the central sensor hub communicating with a Python-based Flask web server through USB serial using a structured JSON protocol. The architecture is modular, reliable, and scalable, making it suitable for both residential and small-scale commercial applications. The hardware layer incorporates a DHT11 temperature and humidity sensor, MQ-2 gas sensor, flame sensor, dual ultrasonic sensors for distance measurement, a PIR motion sensor, and a voltage sensor. A key feature is the dual-mode human detection capability: sensor-based detection using PIR and ultrasonic sensors, and AI-powered detection using YOLOv8 for real-time person identification via a live webcam feed. The system also incorporates a Telegram-based instant alert mechanism for critical safety events including gas leakage, fire, and abnormal temperature. Simulation results confirm that the system achieves reliable real-time monitoring, accurate multi-sensor data fusion, and effective AI-driven automation for smart home environments.**

***Index Terms*—ESP32, IoT, YOLOv8, Smart Home Automation, Real-Time Monitoring, Flask, Telegram Alerts, Sensor Fusion, Energy Monitoring, Computer Vision.**

I. INTRODUCTION

Smart home technology has emerged as one of the most significant advancements at the intersection of embedded systems, wireless communication, and artificial intelligence [1]. The concept of a home that can sense environmental conditions, respond intelligently to occupant behavior, and proactively issue safety alerts has evolved from a theoretical vision into an affordable and deployable solution. However, most commercially available smart home platforms operate as isolated, single-purpose devices smart thermostats, security cameras, and gas detectors each communicate through separate applications, resulting in fragmented data pipelines and limited system intelligence [4].

This paper addresses that gap by presenting a centralized smart home IoT architecture that aggregates data from multiple heterogeneous sensors, applies intelligent decision-making, and delivers real-time alerts through a unified web dashboard. The system is built around the ESP32 microcontroller [1], a cost-effective dual-core SoC with built-in Wi-Fi and Bluetooth, which interfaces with environmental sensors and actuator relays. A Python-based Flask server [6] acts as the backend processing unit, exposing RESTful APIs to a browser-based dashboard. The integration of YOLOv8 [3] for camera-based person detection and Telegram Bot API [10] for push notifications distinguishes this system from conventional sensor-only approaches.
The remainder of this paper is structured as follows: Section II reviews related literature. Section III describes the proposed system architecture. Section IV details the hardware components. Section V explains the methodology. Section VI presents the hardware schematic. Section VII describes the workflow. Section VIII presents simulation results and conclusions.

II. LITERATURE SURVEY

The development of smart home IoT systems has been explored extensively across embedded systems, computer vision, and cloud computing domains [4]. The foundational three-layer IoT architecture perception, network, and application was formalized by Gubbi et al., providing the conceptual basis for the system described here, with the ESP32 at the perception layer, USB serial as the network layer, and Flask forming the application layer.

Research by Shi et al. on edge computing highlights four key benefits over cloud-centric architectures: reduced latency, improved privacy, lower bandwidth consumption, and greater reliability [9]. These advantages are particularly relevant for real-time hazard detection where cloud round-trips introduce unacceptable delays. Privacy concerns arising from continuous transmission of household video and sensor data to remote servers further motivate the local-processing approach adopted in this work.

Environmental sensing for smart homes has been widely studied. DHT-series sensors have been validated for indoor temperature and humidity monitoring, while MQ-2 sensors require threshold tuning due to sensitivity to ambient conditions [7]. Flame sensors based on infrared radiation detection have been identified as cost-effective solutions for residential fire safety. Occupancy detection studies indicate that combining PIR and ultrasonic sensors reduces false positives compared to either sensor alone, though neither can reliably count occupants [4].

Computer vision-based person detection has advanced rapidly. The YOLO (You Only Look Once) architecture [2] introduced a single-pass object detection paradigm that dramatically improved inference speed. YOLOv8 [3] represents the current state of the art, with the nano variant specifically optimized for real-time deployment on CPU-only consumer hardware. Studies confirm that models pre-trained on the COCO dataset generalize well to indoor environments without domain-specific fine-tuning [8].

Alerting system design must balance sensitivity and specificity to avoid alarm fatigue, a problem well-documented in industrial and clinical monitoring [9]. Cooldown mechanisms and priority-based filtering are recommended approaches. Telegram has been identified as a superior notification channel compared to email or SMS due to instant delivery, end-to-end encryption, and a free, well-documented bot API [10].

III. PROPOSED SYSTEM ARCHITECTURE

The proposed Smart Home IoT System is a fully integrated home automation and safety monitoring solution designed to overcome the limitations of conventional standalone systems. As illustrated in Fig. 1, the architecture comprises five major subsystems: (1) multi-sensor hardware node on an ESP32, (2) serial communication layer using JSON over UART at 115200 baud, (3) Flask-based multithreaded backend, (4) YOLOv8 computer vision module, and (5) Telegram notification service.

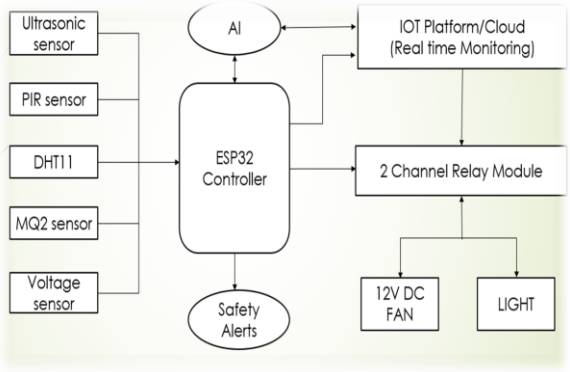

Fig. 1. Block diagram of Real-Time Energy Monitoring System with AI-Driven Automation.

The ESP32 continuously reads all sensor peripherals and serializes the data as a JSON object every 500 ms. The Flask backend deserializes incoming packets in a dedicated reader thread, updates a thread-safe shared state structure, evaluates safety thresholds, and serves live data through RESTful API endpoints. A separate thread runs YOLOv8 inference on webcam frames at approximately 14 fps, annotating detected persons with bounding boxes and confidence scores. Processed frames are delivered to the dashboard via MJPEG streaming, eliminating the need for proprietary plugins.

The system supports bidirectional communication: the backend can transmit text commands (FAN_ON, LIGHT_OFF, AUTO_ON, etc.) to the ESP32, which executes them immediately and acknowledges via serial. An automatic control mode allows the microcontroller to manage relays autonomously

based on local sensor thresholds, providing a fail-safe response even when the host server is temporarily unavailable.

IV. HARDWARE REQUIREMENTS AND PIN CONFIGURATION

The system employs a carefully selected set of components that collectively provide comprehensive environmental sensing, actuation, and local display. The ESP32 DevKit (NodeMCU ESP32) serves as the central controller by virtue of its dual-core Xtensa LX6 processor, 34 programmable GPIO pins, 12-bit ADC channels, and integrated Wi-Fi/Bluetooth [1]. Table I summarizes the complete GPIO pin allocation.

TABLE I. ESP32 Pin Configuration

Pin No.	GPIO	Component	Function
5	GPIO 4	DHT11	Temperature & Humidity Data
6	GPIO 34	Flame Sensor	Analog Fire Intensity Input (ADC1)
7	GPIO 35	MQ-2 Gas Sensor	Analog Gas Concentration (ADC1)
7	GPIO 32	Voltage Sensor	ADC Voltage Divider Input
28	GPIO 5	Ultrasonic 1 TRIG	Distance Sensor 1 Trigger
27	GPIO 17	Ultrasonic 1 ECHO	Distance Sensor 1 Echo Input
29	GPIO 18	Ultrasonic 2 TRIG	Distance Sensor 2 Trigger
30	GPIO 19	Ultrasonic 2 ECHO	Distance Sensor 2 Echo Input
11	GPIO 27	PIR Sensor	Motion Detection Digital Input
36	GPIO 23	Fan Relay (IN1)	2-Channel Relay Control
35	GPIO 22	Light Relay (IN2)	2-Channel Relay Control
21 & 22	GPIO 21/22	LCD (I2C)	SDA/SCL — 16x2 Display (0x27)

The DHT11 sensor [Table II] employs a single-wire digital protocol and delivers ±2°C / ±5%RH accuracy, sufficient for residential comfort monitoring. The HC-SR04 ultrasonic sensor [Table III] provides 2–400 cm non-contact distance measurement at 40 kHz, and two units are deployed to extend spatial coverage for occupancy detection.

TABLE II. DHT11 Pin Configuration

Pin No.	Pin Name	Function
1	VCC	Power Supply (3.3V / 5V)
2	DATA	Digital Output to ESP32 (GPIO4)
3	NC	Not Connected
4	GND	Ground

TABLE III. HC-SR04 Ultrasonic Sensor Pin Configuration

Pin No.	Pin Name	Function
1	VCC	Power Supply (5V)
2	TRIG	Trigger Input from ESP32
3	ECHO	Echo Output to ESP32
4	GND	Ground

The MQ-2 gas sensor operates through a tin-dioxide (SnO_2) sensing element whose resistance decreases upon contact with combustible gases (LPG, methane, propane, hydrogen). Table IV lists its connections. The flame sensor detects infrared radiation in the 760–1100 nm spectrum; its analog output is scaled 0–100% in firmware [Table V]. The voltage sensor employs a 30 kΩ / 7.5 kΩ resistor divider enabling measurement of up to 25 V [Table VI]. The 2-channel relay module [Table VII] provides optocoupler-isolated switching for AC line loads. The 16×2 LCD with I2C interface (address 0x27) [Table VIII] displays real-time system status without consuming parallel GPIO lines.

TABLE IV. MQ-2 Gas Sensor Pin Configuration

Pin No.	Pin Name	Function
1	VCC	Power Supply (5V)
2	GND	Ground
3	A0	Analog Output to ESP32 (GPIO35)
4	D0	Digital Output (Optional, Not Used)

TABLE V. Flame Sensor Pin Configuration

Pin No.	Pin Name	Function
1	VCC	Power Supply (3.3V / 5V)
2	GND	Ground
3	A0	Analog Output to ESP32 (GPIO34)
4	D0	Digital Output (Optional, Not Used)

TABLE VI. Voltage Sensor Pin Configuration

1	VCC	Power Supply (5V)
2	GND	Ground
3	OUT	Analog Output to ESP32 (GPIO32)
4	VIN+	Input Voltage Positive Terminal
5	VIN−	Input Voltage Negative Terminal

TABLE VII. 2-Channel Relay Module Pin Configuration

Pin No.	Pin Name	Function
1	VCC	Power Supply (5V)
2	GND	Ground
3	IN1	Fan Relay Control (GPIO23)
4	IN2	Light Relay Control (GPIO22)
5–7	COM1/NO1/NC1	Relay 1 Output Terminals
8–10	COM2/NO2/NC2	Relay 2 Output Terminals

TABLE VIII. 16×2 LCD Display (I2C) Pin Configuration

Pin No.	Pin Name	Function
1	VCC	Power Supply (5V)
2	GND	Ground
3	SDA	I2C Serial Data (ESP32 GPIO21)
4	SCL	I2C Serial Clock (ESP32 GPIO22)

V. METHODOLOGY

The development methodology followed an incremental, subsystem-first engineering process. The ESP32 firmware was developed in Arduino C++ using the ArduinoJson library for serialization. Sensor interfaces were validated independently before integration. The DHT11 is polled every 2 seconds (its minimum stable interval), while ultrasonic and analog sensors are sampled at 500 ms intervals.

The Flask backend operates a multi-threaded architecture with four concurrent threads: (1) serial reader thread for JSON deserialization and state update; (2) YOLOv8 inference thread for frame capture and person detection; (3) MJPEG streaming thread; and (4) the Flask WSGI server thread. Shared state is protected by Python threading.Lock() objects to prevent race conditions [6].

Threshold-based safety rules govern automatic actuation: the fan activates when temperature exceeds 30°C, gas concentration exceeds 50%, or a person is detected within 100 cm; the light activates on occupancy. Telegram alerts [10] are dispatched asynchronously on separate threads to avoid blocking the main pipeline, with a 60-second per-condition cooldown to prevent notification flooding.

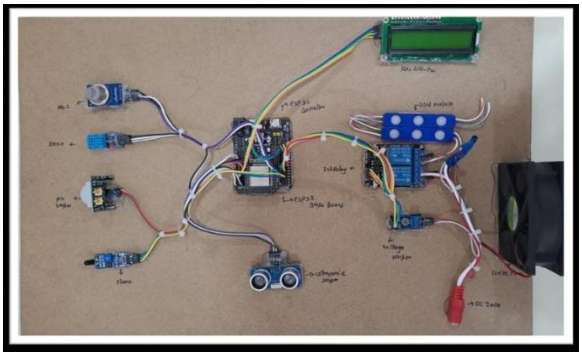

Fig. 2. Assembled hardware prototype showing all sensor modules, ESP32 controller, 2-channel relay, 16×2 LCD, and 12V DC fan.

VI. HARDWARE SCHEMATIC

In Fig. 3 presents the complete ESP32 schematic. The DHT11 DATA line is pulled up to 3.3V via a 4.7 kΩ resistor. The HC-SR04 ECHO output, rated at 5V logic, is voltage-divided to

approximately 3.3V using a 1 kΩ / 2 kΩ resistor network before connecting to the ESP32 GPIO input. The MQ-2 and flame sensors are connected directly to ADC1 pins (GPIO34, GPIO35), which are input-only and immune to the boot-strapping concerns affecting ADC2. The voltage sensor passes its analog output through a 100 Ω series resistor as an ESD precaution before reaching GPIO32. The 2-channel relay JD-VCC pin is powered by the 5V rail (jumper removed from VCC) to provide optocoupler isolation, protecting the ESP32 from relay transients. The LCD is addressed at 0x27 via the I2C bus shared on GPIO21 (SDA) and GPIO22 (SCL).

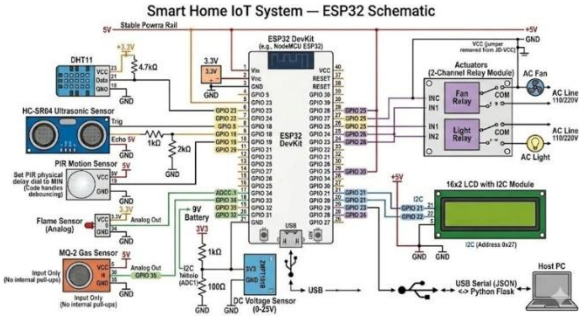

Fig. 3. Complete ESP32 hardware schematic showing sensor connections, relay actuation, LCD display, voltage sensor, and USB serial interface to host PC.

VII. SYSTEM WORKFLOW

The complete operational workflow is illustrated in Fig. 4. On power-up, the system initializes all peripheral subsystems in sequence: LCD, DHT11, serial port, camera, and YOLOv8 model. If any optional subsystem (camera or ML model) fails to initialize, the system continues with a graceful degraded mode and reports the missing capability on the dashboard — this fault-tolerant startup prevents a single hardware absence from disabling the entire platform.

In steady-state operation, the ESP32 transmits a JSON sensor snapshot every 500 ms. The Flask serial reader thread parses each packet, updates the shared state, and evaluates threshold conditions. When a threshold is breached, the alert dispatcher checks the per-condition cooldown timer and, if sufficient time has elapsed, sends a formatted Telegram message on a background thread. Concurrently, the YOLO thread processes the latest webcam frame and updates the person count. The dashboard polls the /api/data endpoint every 2 seconds via JavaScript fetch(), rendering updated sensor cards, status badges, occupancy indicators, and the live MJPEG stream.

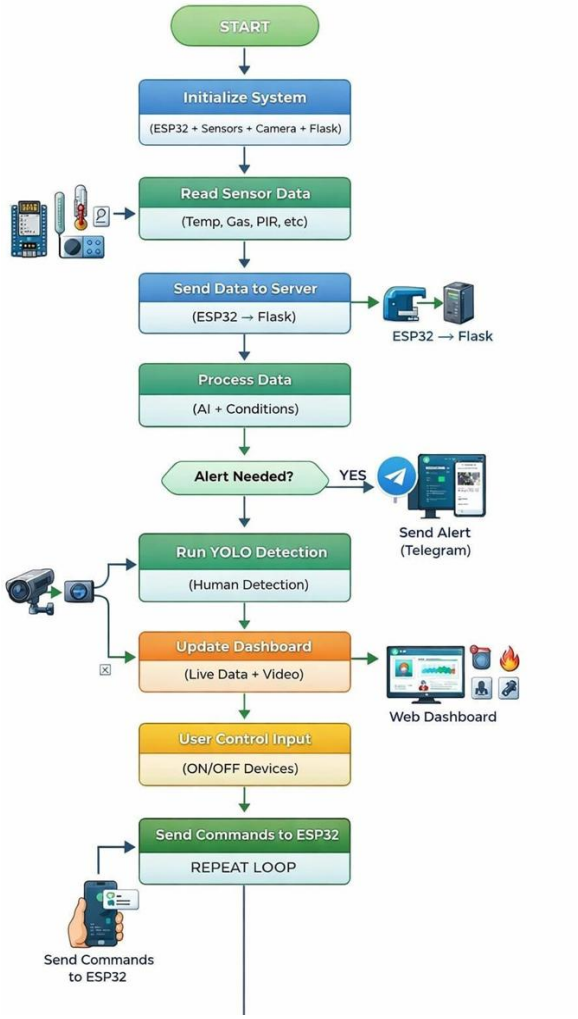

Fig. 4. System workflow from initialization through sensor reading, data processing, YOLO detection, dashboard update, and command dispatch loop.

VIII. SIMULATION RESULTS

The integrated system was subjected to functional testing across all subsystems. The ESP32 transmitted stable JSON packets at the configured 500 ms interval, with no packet loss observed over a 30-minute continuous test. Fig. 5 shows the web dashboard during normal operation, displaying live sensor readings including a temperature of 32.8°C, humidity at 71.9%, gas at 2%, and flame at 0%. The serial connection status, YOLO inference state, and camera health are reflected in real-time status badges in the header bar.

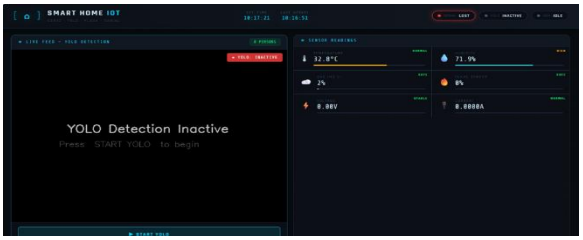

Fig. 5. Web dashboard showing live sensor readings (Temp: 32.8°C, Humidity: 71.9%, Gas: 2%, Flame: 0%), YOLO feed, and system status indicators.

Fig. 6 illustrates the occupancy detection and device control panel. The ultrasonic sensor reported a distance of 50 cm, triggering the person-detected indication. The device control section confirms the Fan Relay and Light Relay toggling ON, with the system alert log recording the fire detection event (flame sensor: 98%), Telegram test dispatch, and relay state transitions with timestamps.

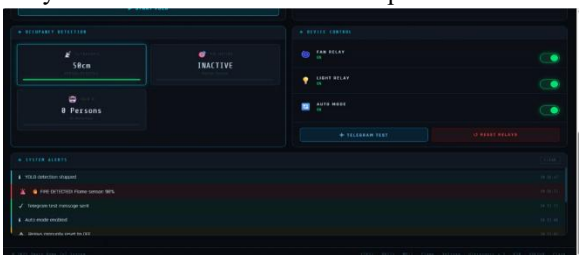

Fig. 6. Occupancy detection (ultrasonic: 50 cm), device control toggles, and system alerts log showing FIRE DETECTED event at flame sensor 98%.

Fig. 7 shows the hardware prototype in the powered-on state. The 16×2 LCD displays live readings (32.7°C, H:72%, Person:YES, AUTO mode), the relay module LEDs confirm active switching, and the 12V DC fan is operational. Fig. 8 presents the Telegram notification log captured during the experiment, demonstrating successful delivery of person detection alerts (YOLO: 1–3 persons), system online confirmations, and a critical FIRE ALERT (flame sensor: 98%) on April 20, 2026 at 10:16 AM.

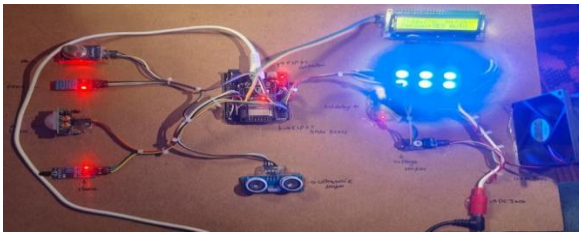

Fig. 7. Hardware prototype in operating state: LCD showing temperature, humidity and occupancy; relay LEDs illuminated; DC fan running.

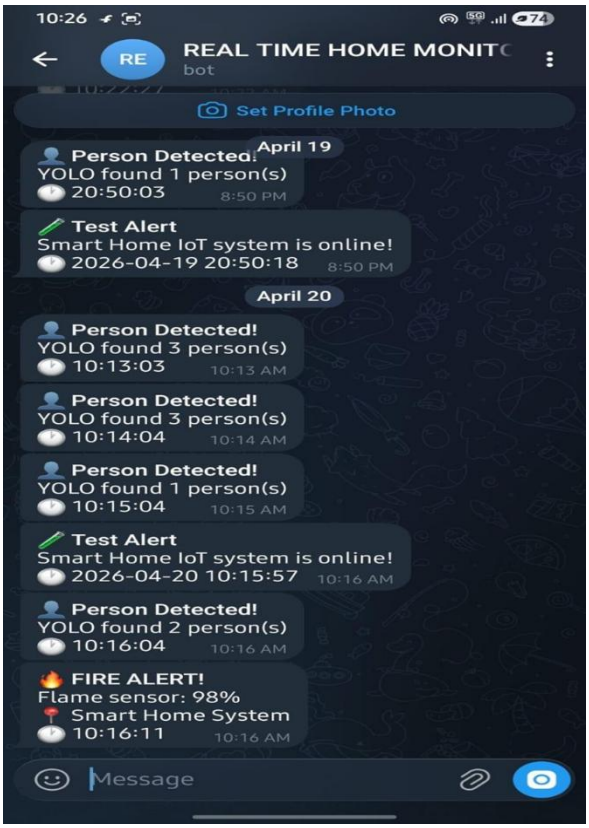

Fig. 8. Telegram bot notification log demonstrating person detection alerts, system online test messages, and critical FIRE ALERT (Flame sensor: 98%).

IX. CONCLUSION

This paper presented a comprehensive Smart Home IoT System that successfully integrates multi-sensor environmental monitoring, AI-based person detection via YOLOv8, real-time web dashboard visualization, and Telegram-based push alerting into a unified, locally-hosted platform. The system overcomes the key limitations of commercial IoT solutions fragmentation, cloud dependency, and privacy concerns by processing all sensor data on-premises using a Flask multithreaded backend, while restricting external communication to essential safety notifications.

Simulation results confirm reliable JSON data acquisition at 500 ms intervals without packet loss, ~14 fps YOLO person detection on CPU hardware, sub-3-second Telegram alert delivery, and seamless bidirectional relay control. The fault-tolerant initialization mechanism ensures continued operation even when optional subsystems such as the camera or

ML model are unavailable. The modular architecture supports straightforward extension with additional sensors, actuators, advanced AI models, or mobile applications with minimal code changes.

Future work will focus on time-series data logging with InfluxDB for historical trend analysis, face recognition for security-aware occupancy management, OTA firmware updates for the ESP32, and integration with the Matter smart home protocol for cross-ecosystem compatibility.

REFERENCES

[1] Maier, A., Sharp, A., & Vagapov, Y. (2017). Comparative analysis and practical implementation of the ESP32 microcontroller for IoT applications in smart home systems. International Journal of Electrical and Computer Engineering, 7(5), 2483–2491.

[2] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779–788.

[3] Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLOv8. Ultralytics Inc. https://github.com/ultralytics/ultralytics

[4] Ghayvat, H., Mukhopadhyay, S., Gui, X., & Suryadevara, N. (2015). WSN and IoT based smart homes and their extension to smart buildings. Sensors, 15(5), 10350–10379.

[5] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer vision with the OpenCV library. O'Reilly Media.

[6] Grinberg, M. (2018). Flask web development: Developing web applications with Python (2nd ed.). O'Reilly Media.

[7] Kodali, R. K., Jain, V., Bose, S., & Boppana, L. (2016). IoT based smart security and home automation system. International Conference on Computing, Communication and Automation (ICCCA), 1286–1289.

[8] Brownlee, J. (2019). Deep learning for computer vision: Image classification, object detection, and face recognition in Python. Machine Learning Mastery.

[9] Longari, S., Zanero, S., & Maggi, F. (2020). Real-time monitoring and anomaly detection in IoT environments using machine learning techniques. IEEE Internet of Things Journal, 8(3), 1938–1952.

[10] Telegram. (2023). Telegram bot API documentation. Telegram Messenger Inc. https://core.telegram.org/bots/api