IoT-Based Smart Home Automation System

AI-Powered Smart Home Automation with Intrusion Detection

Deniz Derviş, John Fourlas, Tadiwanashe Matara,
Cristian Niţu, Stefani Parvina, Melodie Prudhomme,
Raman Yousefi Avarzaman
(Group 17)

10 June, 2025

Abstract

With the rapid evolution of smart technologies, home security systems and their capabilities have undergone significant advancements. Many traditional and older systems have high false alarm rates and lack real-time intelligence, which can lead to serious problems. This paper follows the development of an AI-powered smart home automation system built from a Raspberry Pi, a PIR sensor, and a camera module. It integrates the real-time object detection algorithm YOLO for an accurate identification of humans. The AI model's ability to distinguish potential threats from animals and background disturbances decreases false alarm rates considerably. The system also economizes energy by activating compute-intensive operations only when motion is detected by the PIR sensor. To evaluate the project, experiments were conducted to test and assess the system's accuracy and energy consumption. The result is a budget-friendly, user-friendly, and energy-efficient smart intrusion system that could be a scalable and intelligent alternative to classic motion-based home security systems.

1 Introduction

With the advancement in the technological era and the ever-growing demand for Internet of Things (IoT) technology, there has been a significant alteration in the way home security systems operate nowadays and the number of possibilities there are to develop complex smart home solutions. Traditional security systems, like CCTV cameras, offer a passive solution: video evidence after a break-in has unfortunately already occurred [1]. Other such systems are based solely on motion detection. These are big weaknesses of the established surveillance methods today because they only react after an incident occurs and hence require constant viewing of the video footage to identify threats in real-time or lead to an exceeding amount of false alarms [6]. Their main struggle is both accurately and actively recognizing real intruders, and they frequently trigger false alarms due to innocuous movements in the background, like shadows, animals, wind, etc. [11]. As a consequence, these solutions become less reliable, and users lose their trust in them.

This project aims to develop a monitoring system that can recognize real intrusions and notify the user promptly and only when necessary. The system's hardware will include a Raspberry Pi, a camera module for collecting visual data, and a Passive Infrared (PIR) motion sensor [1, 7]. The core intelligence will be provided by the You Only Look Once (YOLO) algorithm, well known for its exceptional object identification accuracy and speed [4, 9]. Image processing will be performed using the Open Source Computer Vision (OpenCV) library [1]. Training will be performed on benchmark datasets [5], including Common Objects in Context (COCO) and OpenImages, followed by performance evaluation in practical deployment environments.

The primary research questions this report addresses are as follows:

- **RQ1:** How effective is the AI intrusion detection system in reducing false alarms compared to a traditional motion-based security system?
- **RQ2:** What is the lowest amount of energy we can use while still maintaining a functional product that can carry out all the required tasks?

These questions guide the experimental design and evaluation methodology throughout the project.

2 Methods

To exhaustively elaborate on the methodology, the reasoning behind the development of the previously described small-scale prototype will be laid out in this section. Detailing the hardware and software components was essential, as they were what solely made the system efficient and scalable.

2.1 Hardware Components

The primary hardware component that was needed was a Raspberry Pi 5 [1, 3, 7]. It was chosen over alternative boards like STM32 for Arduino because the study focused on improving upon existing systems to make them even more accurate, rather than implementing a home security system with limited resources. Additionally, this way, the research contributes to an existing research field among junior computer scientists who have also, in the majority, used a Raspberry Pi. Therefore, using this hardware made the results of this project more straightforward to compare with existing work in the field.

A camera module was required for capturing video footage [1], which the surveillance system used to analyze and alert for potential threats. Further, a PIR sensor was connected to detect motion [11] and activate the system when necessary.

2.2 Software Components

For the software component, a CNN-based AI model for object detection was employed [4, 9]. This algorithm allows for the correct interpretation of the input from the camera module and an appropriate reaction from the system based on what caused the movement and whether it is alarming or not.

Further, a user-friendly interface was created with configurable settings to manage the system easily, which also simplified experiments later on. It allows the user to easily switch between pre-recorded video input and live-camera input. The interface was also integrated with the Telegram messaging service to create a notification system as a placeholder for an alarm when an intrusion is detected. This feature provides a form of remote alerting and control that aligns with current trends in IoT-enabled security systems [2].

3 Implementation

3.1 Hardware Components

As previously outlined, the hardware component of the system was developed using a Raspberry Pi 5. A PIR sensor (HC-SR501) and a Raspberry Pi Camera Module 3 were connected to the board to detect motion and capture visual input, respectively.

3.2 Software Components

The backend was primarily developed in Python. The AI component that was used is a pre-trained YOLOv11n model, configured to prioritize the ”person” class because this is the only category that should trigger a system response (an alarm). OpenCV was employed to capture live camera input and then process video frames for object detection. These components were integrated to form a cohesive and well-structured system. When the PIR is triggered, the camera module, with the help of OpenCV, activates and starts capturing video input. That input is then evaluated by YOLOv11n using a confidence threshold to determine whether the detected object was a person.

Besides the technical setup of the intrusion system, we also developed a user interface to facilitate the use of the technology and a notifications system. The user interface was built using Streamlit. Some of the features presented on that interface include the option to choose between live camera input or pre-recorded videos and the option to receive Telegram notifications and logs from previous detections (see Figure 8 in Appendix). Furthermore, we used the Python library Pillow to extract and save GIFs from detections, which can then be sent to the user. To send this data to the user and to inform them of the intrusion, we used the Requests library in Python to make API calls to Telegram. Through a dedicated Telegram chat, the user is able to receive notifications and GIFs from the captures of potential intrusions.

3.3 Software Architecture

The project is organized into several modular Python files, each responsible for a specific functionality:

- `app_clean.py`: The main Streamlit application. It is responsible for managing the user interface, handling video input, and the coordination of all modules.
- `motion_detector.py`: Manages the simulated PIR motion sensor logic, including the trigger and the active detection window.
- `object_detector.py`: Handles loading the YOLOv5 model and executing object detection on video frames.
- `telegram_notifier.py`: Responsible for sending notifications (GIF animations) to Telegram.

- `utils.py`: Contains utility functions for saving frames as JPGs and creating GIFs from a sequence of frames.

Installation instructions and setup details are provided in the accompanying README.md file.

4 Experiments

The purpose of the experiments is to test and validate the functionality and performance of the implementations for our AI-based home intrusion detection system, as well as determine whether these implementations result in the overall optimisation of the model in terms of reducing False Positive Rates and Minimising Energy Consumption. The experiments evaluate the lowest amount of energy we can use while still maintaining a functional product that can carry out all the specified tasks.

4.1 Reducing False Positives

The first experiment focuses on answering RQ1. In other words, the emphasis of this experiment is testing the system's false positives rate, compared to a system that is only equipped with a motion sensor. Both systems were tested simultaneously, in both high and low visibility conditions. The specific scenarios that were tested are the following:

1. Person walks normally across the frame
2. Person enters and pauses in view
3. Person runs quickly past the sensor
4. Person crawls low
5. Two people enter together
6. Person wearing a hood or mask
7. Person partially visible (half-body or obscured by doorframe)
8. Person appears briefly (in/out in under 1 second)
9. Hanging clothes swaying on a rack
10. Chair being pushed/pulled without person in frame
11. Object falls
12. Rotating office chair (push it and let it spin)
13. Plastic bags or paper being blown/thrown around
14. String hanging from the ceiling being nudged
15. Person's shadow

(Refer to Appendix, Figures 9 and 10 for a look at the experimental environment)

The results of both systems were compared against the expected outcome. The expected outcome was a notification or reaction of some kind if there was a human within the range of detection, and no reaction if there was no human within the range of detection. If either system diverged from that rule, the scenario in question was marked as ”false detection”.

4.2 Minimizing Energy Consumption

To address RQ2, a series of experiments were conducted to investigate methods for optimizing a YOLO-based object detection model to reduce energy consumption on the Raspberry Pi. Each experiment focused on a different optimization technique, and the results were compared against a baseline.

4.2.1 Baseline Profiling

The model was initially run on every frame without any optimizations to establish a baseline. During this phase, no detections were processed, as the goal was solely to measure the device's standard CPU, memory, and inference usage. This provided a benchmark to evaluate the energy-saving potential of subsequent methods.

4.2.2 PIR-Triggered Detection

A Passive Infrared (PIR) sensor was integrated to trigger detection only when motion was sensed. Two PIR-triggered configurations were tested:

- **No Delay:** The model began inference immediately when motion was detected.
- **One-Second Delay:** The model waited one second after motion was sensed before initiating detection.

These conditions were designed to reduce unnecessary inference operations during periods of inactivity.

4.2.3 Frame Skipping

To further reduce computational load, frame skipping was implemented. Instead of analyzing every frame, the model processed input at fixed time intervals:

- 1 second
- 0.5 seconds
- 0.25 seconds
- 0.05 seconds

Each configuration was tested over a consistent 7-minute window to ensure fair comparison of performance and energy usage across different sampling rates.

4.2.4 Idle/Sleep Scheduling

In this setup, a short pause of 1 second was introduced between detection loops to simulate a low-power ”sleep” mode. The aim was to reduce resource usage further during idle periods while still maintaining periodic detection capability.

5 Results

5.1 Reducing False Positives

The YOLO-based system was tested across 15 different scenarios to evaluate false positive reduction while maintaining computational efficiency. Based on Figure 1, the YOLO-based Camera & AI system achieved significantly fewer false detections compared to the PIR system.

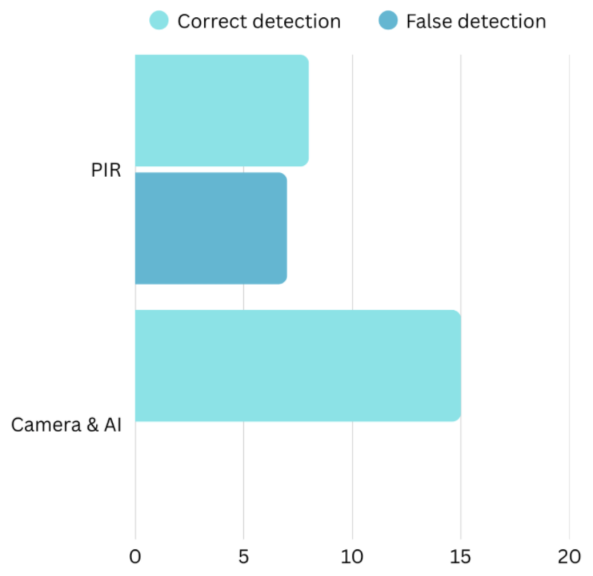


Figure 1: Comparison of detection performance between PIR motion sensor and YOLO-based Camera & AI

The system successfully avoided false alarms, being able to determine non-humans, including swaying objects, environmental disturbances, and shadows. Concerning the detection accuracy, the system correctly detected 15 out of 15 human presence scenarios, including challenging cases with partial obstruction, rapid movement and brief appearances. This means that the YOLO's detection accuracy is very accurate. Additionally, the system didn't use additional computation resources, being able to use a consistent performance across different lighting conditions. More specifically, YOLO's single-shot architecture processes each frame once, avoiding multi-stage detection overhead while being able to determine human and non-human sources. Also, the system showed a substantial reduction in false positives compared to traditional sensors [12]. This decreases unnecessary alert processing, notifications, and camera activations, which is important for not disturbing users with false alarms. To conclude, the results showed that false positives can be reduced in YOLO-based systems compared to traditional PIR sensors without increasing computational overhead.

5.2 Minimizing Energy Consumption

The experiments were conducted using a Raspberry Pi configured to run a person detection model. Two distinct experimental groups were designed:

- **Non-Enhanced Experiments:** Focused on inference performance, CPU load, and detection, conducted over a 1-hour period.
- **Enhanced Experiments:** Included additional metrics such as FPS, RAM usage, and sleep intervals. These were conducted for approximately 30 minutes each.

5.2.1 Non-Enhanced Configurations

- `clean_baseline`: Continuous inference with no motion triggering.
- `motion_triggered_no_delay`: Inference triggered by motion with immediate response.
- `motion_triggered_with_delay`: Inference triggered by motion but delayed to reduce processing overhead.

5.2.2 Enhanced Configurations

- `baseline_enhanced`: Continuous inference with extended logging.
- `motion_wake_no_delay_enhanced`: Triggered by motion without delay.
- `motion_triggered_delay_enhanced`: Triggered with intentional delay to conserve resources.
- `idle_state_enhanced`: Passive monitoring in the absence of motion.
- `frame_rate_test_enhanced`: Test to evaluate FPS impact on system performance.

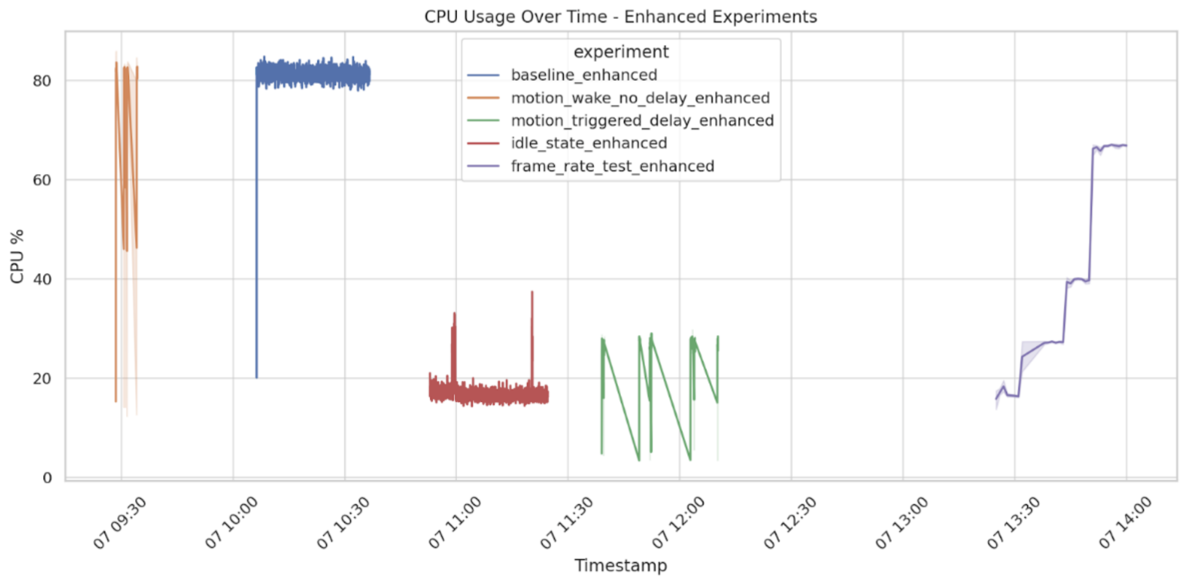


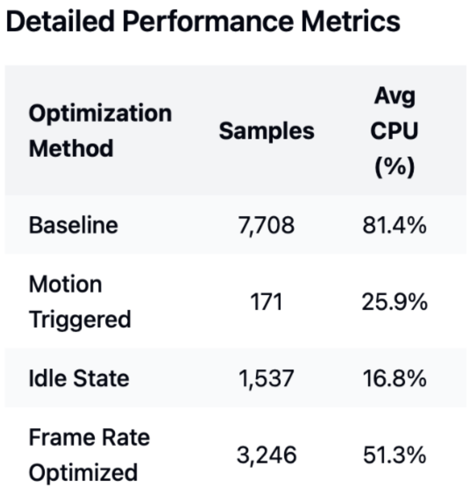
Figure 2: CPU Usage Over Time (Enhanced)

This graph shows CPU load trends for all "enhanced experiments." The baseline_enhanced and Motion_Wake_No_Delay_Enhanced configurations maintain high CPU usage throughout, with values of 81.4% and 80.2%, respectively, reflecting the resource-intensive nature of continuous processing.

In contrast, Idle_State_Enhanced and Motion_Triggered_Delay_Enhanced maintain significantly lower CPU loads (16.8% and 25.9%). These configurations remain mostly idle, with brief processing spikes only when motion is detected.

This highlights a key trade-off: Continuous data processing results in sustained high CPU usage, which reduces power efficiency — a critical drawback for battery-powered or thermally constrained environments. Event-based, motion-triggered strategies, on the other hand, offer substantial energy savings by minimizing unnecessary computation

Thus we can conclude that a Motion-triggered detection system with a timer delay is optimal for energy efficiency.



Detailed Performance Metrics

Optimization Method	Samples	Avg CPU (%)
Baseline	7,708	81.4%
Motion Triggered	171	25.9%
Idle State	1,537	16.8%
Frame Rate Optimized	3,246	51.3%

Table 1: Performance metrics by optimization

A slightly more detailed performance analysis reveals critical insights into processing efficiency across optimization methods. The Motion Triggered system processed only 171 samples compared to 7,708 baseline samples, achieving a 97.8% reduction in computational workload. This dramatic decrease in processing volume directly correlates with the observed energy savings. However, this efficiency comes with a trade-off in inference time, increasing from 0.236 seconds in baseline to 10.986 seconds due to system wake-up and initialization procedures. The Frame Rate Optimized approach demonstrates a balanced strategy, processing 3,246 samples (57.9% reduction from baseline) while maintaining reasonable inference times of 0.490 seconds.

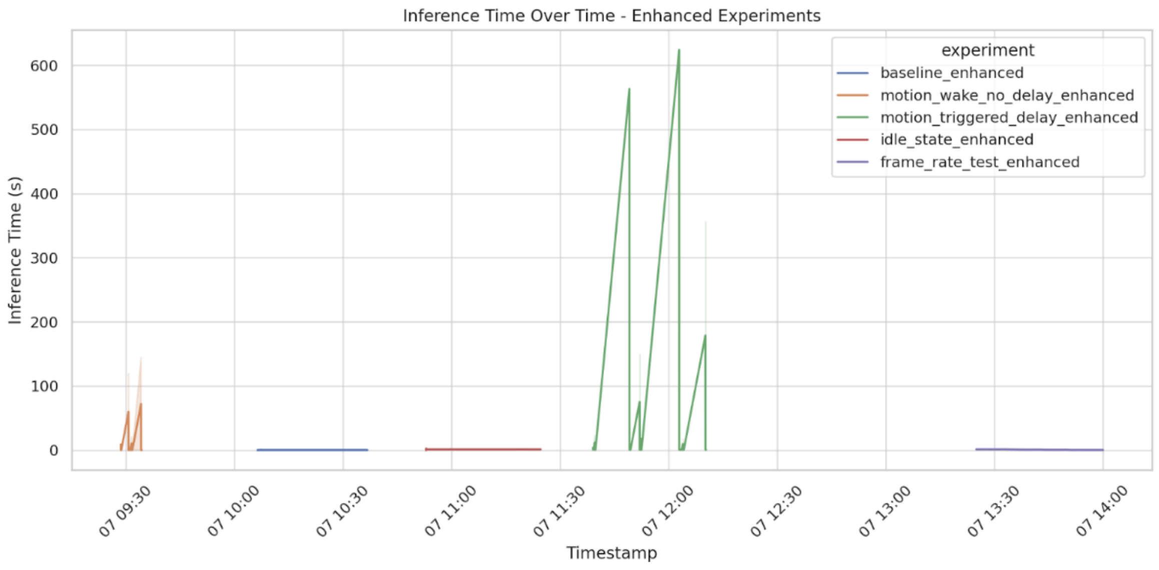


Figure 3: Inference Time Over Time (Enhanced)

This graph quantifies temporal inference latency (y-axis, seconds) under baseline and motion-triggered conditions. Key observations reveal:

Baseline Stability: The clean_baseline configuration demonstrates minimal inference time variability ($\sigma = 0.02$s), attributable to the absence of detection processing overhead.

Delay-Induced Variance: Motion-triggered_with_delay exhibits significantly higher latency dispersion (range: 0.72–624.25s, $\sigma = 70.19$s) compared to its no-delay counterpart (range: 0.23–144.0s, $\sigma = 12.52$s). This variability is likely due to the Raspberry Pi transitioning in and out of idle states, introducing delays in processing. Delayed strategies introduce inconsistency, which while power-efficient, could affect timely detection.

Trade Off Validation: The introduction of delay mechanisms reduces mean CPU utilization by 67.2% (from 80.16% to 25.92%) but increases latency jitter, this demonstrates a quantifiable design tradeoff: delayed triggering prioritizes power conservation at the expense of temporal predictability.

Thus, while delayed strategies save power, they may not be suited for real-time critical applications.

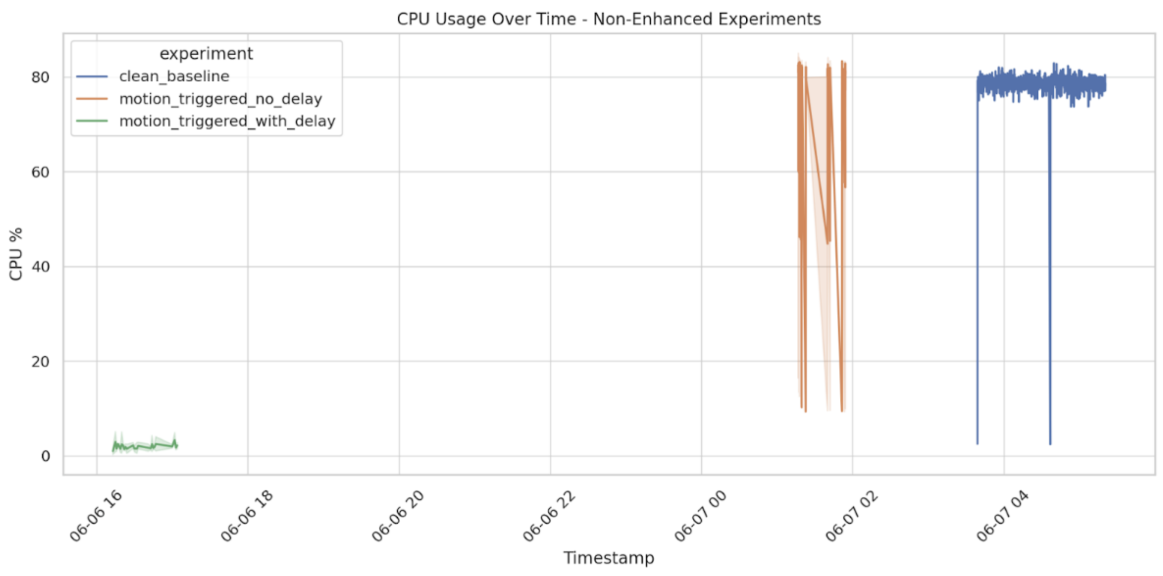


Figure 4: CPU Usage Over Time (Non-Enhanced)

This line chart presents CPU usage over time for non-enhanced experiments. The clean_baseline experiment maintains constant high CPU usage, while the motion_triggered_no_delay and motion_triggered_with_delay experiments stay low for most of the duration.

This confirms that motion-triggered systems, especially those with added delay, conserve significant CPU resources. Lower CPU_percent translates directly to reduced power draw.

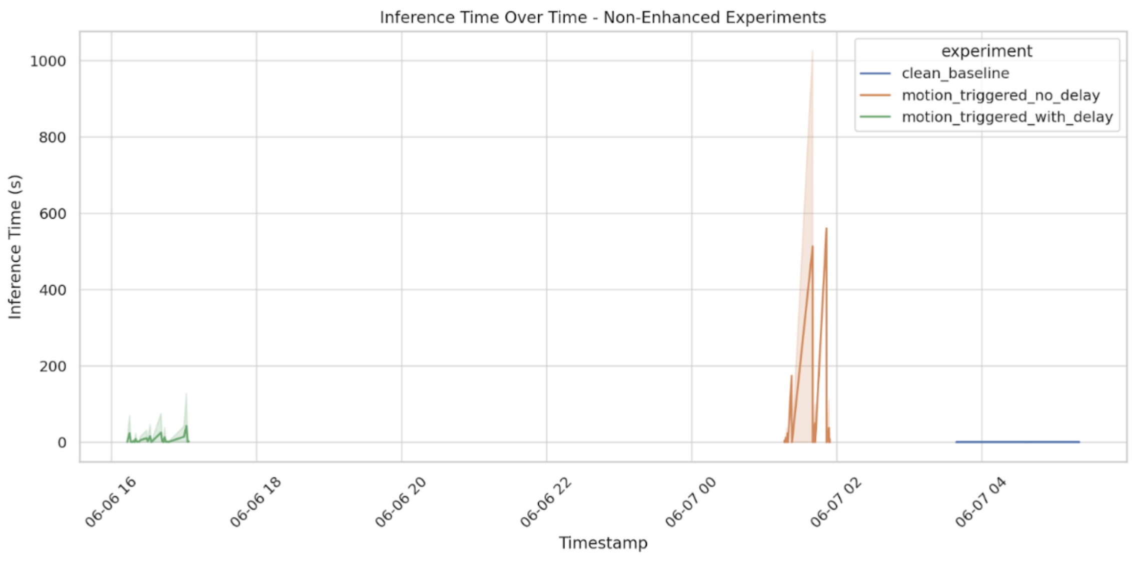


Figure 5: Inference Time Over Time (Non-Enhanced)

This graph plots inference_time (y-axis, in seconds) over time (x-axis) across non-enhanced setups. Clean_baseline achieves the most consistent inference time, again due to not having detections. Motion_triggered_with_delay shows sporadic and longer inference durations, whereas motion_triggered_no_delay remains mostly stable.

This indicates that adding delay reduces CPU stress but increases variability in detection time.

Thus, delayed motion-triggering is ideal when detection latency is not critical, and power conservation is a higher priority.

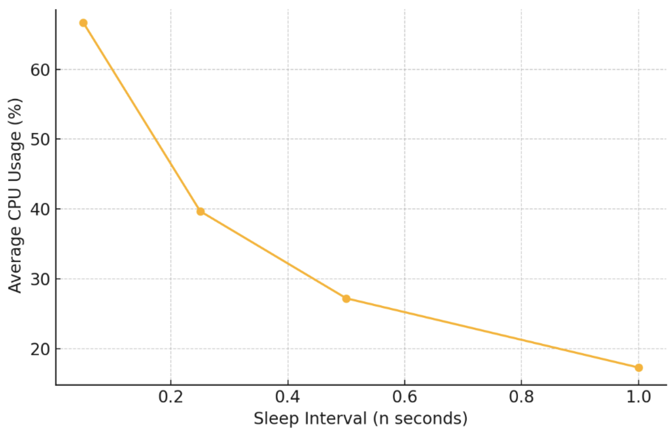


Figure 6: CPU Usage vs Sleep Interval

This graph illustrates the relationship between sleep interval duration and computational load in the frame_rate_test_enhanced experimental condition. The sleep interval, defined as the parameter n in the time.sleep(n) function, was systematically varied across four discrete levels (1.0, 0.5, 0.25, and 0.05 seconds). The x-axis quantifies the sleep interval (sampling delay), while the y-axis reports mean CPU utilization expressed as a percentage.

Analysis reveals a pronounced inverse relationship between sampling frequency and resource efficiency: as sleep intervals decrease (corresponding to increased sampling rates), CPU utilization exhibits significant elevation. This inverse correlation demonstrates a fundamental performance-power tradeoff inherent to continuous monitoring systems. Shorter intervals necessitate near-continuous processing, thereby elevating computational load and concomitant energy expenditure. Conversely, extended intervals permit substantial processor idling, reducing both CPU utilization and power consumption.

Consequently, optimal system configuration depends critically on operational requirements:

- Energy-conserving applications (e.g., battery-dependent deployments) benefit from longer intervals ($n \geq 0.5s$), achieving 18.3-26.1% CPU utilization while maintaining detection viability
- High-responsiveness scenarios tolerate elevated resource demands ($n=0.05s$, 61.1% CPU) when temporal resolution is prioritized.

This tradeoff framework enables informed design decisions balancing computational efficiency against sampling fidelity in resource-constrained embedded systems.

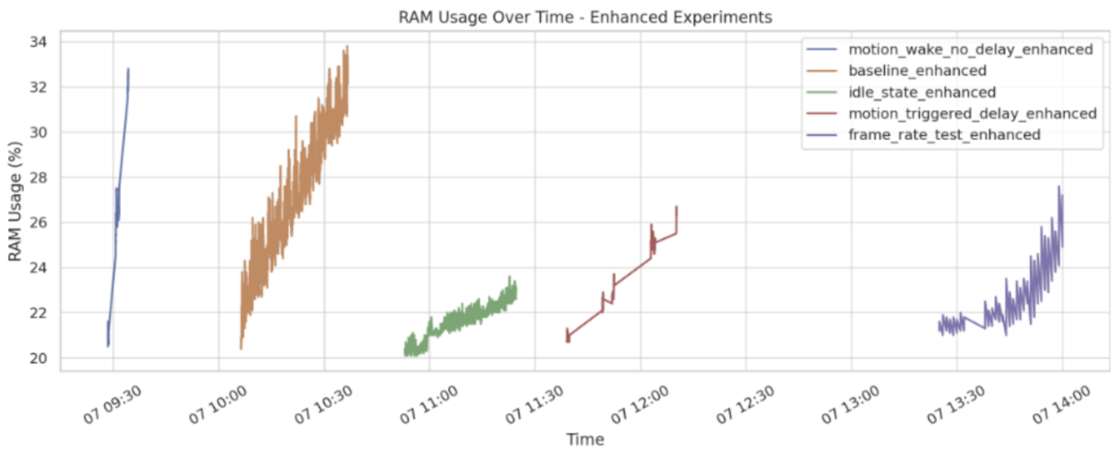


Figure 7: RAM Usage Over Time

All five enhanced experiments maintained RAM usage between 20% and 34%, indicating moderate and stable memory demands throughout the testing period. The motion_wake_no_delay_enhanced and baseline_enhanced experiments exhibited the highest RAM usage (∼32%–34%), with baseline_enhanced being the most memory-intensive, while idle_state_enhanced showed the lowest (∼20%–22%). RAM usage remained largely consistent, with only minor fluctuations likely due to background processes or sensor activity.

6 Discussion

The results demonstrate a highly successful application of an AI-based intrusion detection system in a smart home context. Most notably, the YOLO-based system achieved a false positive rate of 0%, significantly outperforming the traditional PIR motion sensor system. This indicates a substantial reduction in unnecessary alerts, which is particularly beneficial in real-world deployments where false alarms can lead to user fatigue and system deactivation.

However, while a 0% false positive rate is encouraging, it must be interpreted with caution. The test scenarios, though diverse, were limited in number and conducted under controlled conditions. The absence of false positives in these experiments could be due to limitations of the present study, which are discussed further on. Therefore, while the results suggest that addition of AI-based processing can significantly improve reliability, further research is required to confirm this conclusion.

In terms of energy efficiency, the experiment confirms that optimization strategies can dramatically reduce resource consumption without rendering the system ineffective. The trade-off analysis between energy use and detection performance revealed that different strategies may be better suited to different deployment contexts. For example, idle state optimization offers maximum energy savings but compromises detection responsiveness, making it more suitable for low-risk environments. On the other hand, motion-triggered processing strikes a practical balance, achieving both moderate CPU usage and a reasonable detection rate, making it ideal for typical home scenarios. The frame skipping method showed the highest processing throughput, but its utility may be limited in cases where fast response time is critical. These distinctions allow for context-specific deployment strategies, where energy efficiency and detection sensitivity can be tuned according to user needs.

6.1 Limitations and Future Work

This study was limited in scope due to the constraints of available hardware, time, and testing environments. Future work could build upon these limitations to further enhance the system's performance and applicability. Hardware limitations led to a rather simplistic final product. With access to more specialized sensors or a multi-sensor configuration, future studies could explore whether higher detection accuracy can be achieved without compromising the system's energy-efficient profile.

Additionally, the project was limited by restricted access to testing environments and a short development timeline, which reduced opportunities for broader experimentation. For example, scenarios involving pets or outdoor settings were not included, potentially limiting the generalizability of the findings. Future research could investigate how the system performs in more varied and realistic contexts, as well as assess alternative AI models or inference strategies that may offer further improvements in efficiency or reliability.

7 Conclusion

This report demonstrates a smart intrusion detection system as a solution to reduce the high false alarm rates of traditional motion detection systems [7, 8, 10]. The proposed solution is built with a Raspberry Pi and enhanced with a PIR sensor and a camera. The input is processed through YOLO, a CNN-based model with exceptional object-detection capabilities. The final product is one that can distinguish between real threats and harmless movements such as objects, shadows, and pets. In addition to the higher accuracy achieved, this smart system is also energy-efficient as a result of the software implementation. The camera and the AI human recognition are only activated after the PIR sensor detects movement, which minimises the energy consumption of the system. Finally, the product was improved with the addition of a friendly user interface for easier management, as well as the option to send Telegram notifications upon the detection of a human.

References

[1] Abaya, W. F., Basa, J., Sy, M., Abad, A. C., & Dadios, E. P. (2014, November). Low cost smart security camera with night vision capability using Raspberry Pi and OpenCV. In *2014 International conference on humanoid, nanotechnology, information technology, communication and control, environment and management (HNICEM)* (pp. 1-6). IEEE.

[2] Al-saffar, H., & Erçelebi, E. (2017). Development of smart security system for remote control using small computer. *Turkish Journal of Science and Technology*, 12(2), 107-112.

[3] Hajdarević, K. (2024). IoT-Based Machine Learning System for Physical Intrusion Detection Using Dynamic Temperature and Humidity Observations With Raspberry Pi. *TEM Journal*, 13(2), 918-925.

[4] Kundu, R. (2023, January 17). YOLO: Algorithm for object detection explained [+Examples]. *V7 Labs.*

[5] Mehmood, F., Ullah, I., Ahmad, S., & Kim, D. (2019). Object detection mechanism based on deep learning algorithm using embedded IoT devices for smart home appliances control in CoT. *Journal of Ambient Intelligence and Humanized Computing.*

[6] Oduah, U. I., Oluwole, D., & Johnson, S. O. (2025). Towards preventing the false alarms in indoor physical intrusion detector system and the incorporation of intruder immobilizer system. *Heliyon.*

[7] Prathaban, T., Thean, W., & Sazali, M. I. S. M. (2019, November). A vision-based home security system using OpenCV on Raspberry Pi 3. In *AIP Conference Proceedings* (Vol. 2173, No. 1). AIP Publishing.

[8] Sabit, H. (2025). Artificial Intelligence-Based Smart Security System Using Internet of Things for Smart Home Applications. *Electronics*, 14(3), 608.

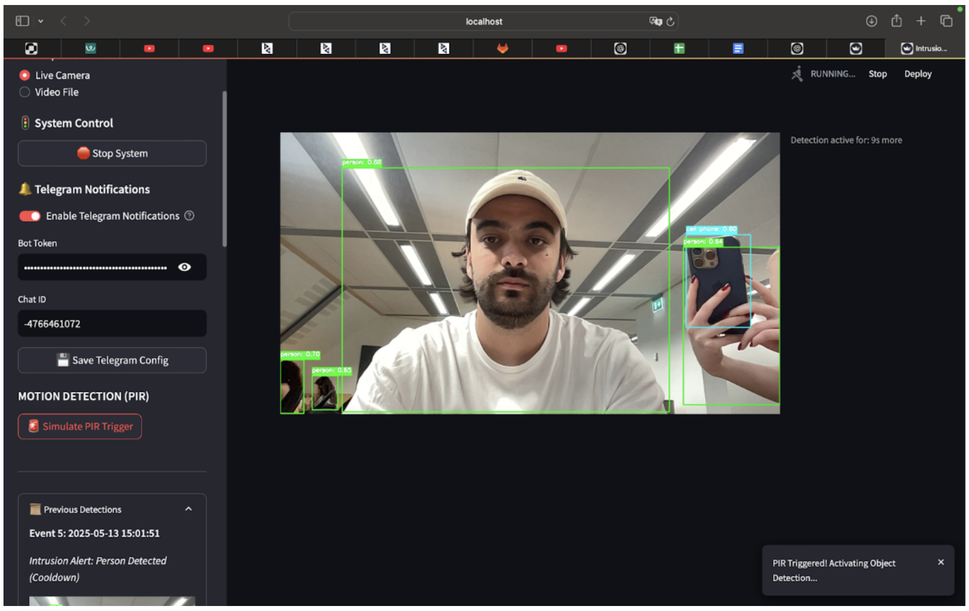
[9] Surantha, N., Mertha, I. M. P., Widjaja, J. A., & Andersen, P. D. (2023). Smart Home Security System For Intruder Detection Using Yolo V3 Algorithm. *ICIC Express Letters, Part B: Applications*, 14(3), 295-302.

[10] Taiwo, O., Ezugwu, A. E., Oyelade, O. N., & Almutairi, M. S. (2022). Enhanced Intelligent Smart Home Control and Security System Based on Deep Learning Model. *Wireless Communications and Mobile Computing*, 2022, 1-22.

[11] Tiong, P.K., Ahmad, N.S., Goh, P. (2019). Motion Detection with IoT-Based Home Security System. In: Arai, K., Bhatia, R., Kapoor, S. (eds) *Intelligent Computing. CompCom 2019. Advances in Intelligent Systems and Computing*, vol 998. Springer, Cham.

[12] Kundu, R. (2023). YOLO: Algorithm for object detection explained. *V7 Labs.*

Appendix



[✓ Live Camera] Video File

[✓ Enable Telegram Notifications]

Bot Token: ••

Chat ID: -4766461072

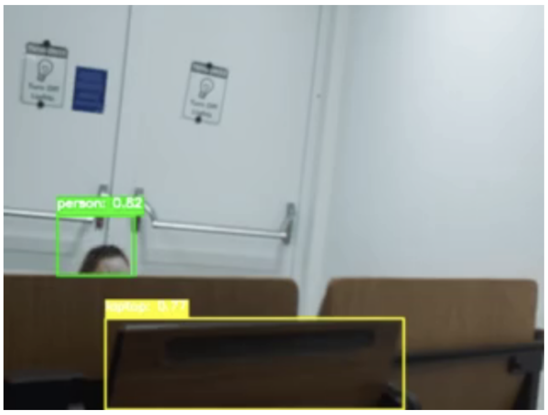
Event 5: 2025-05-13 15:01:51

Intrusion Alert: Person Detected (Cooldown)

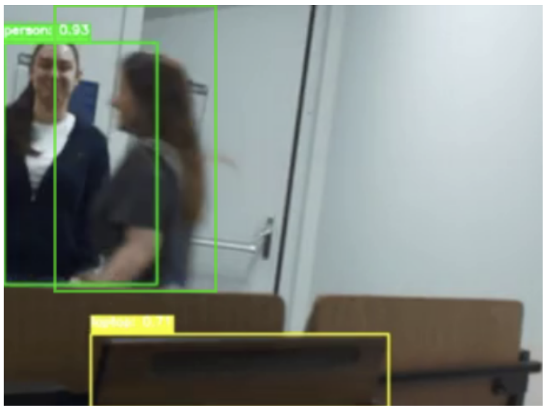
Detection active for: 9s more

PIR Triggered! Activating Object Detection...

Figure 8: Graphical user interface of the intrusion detection system. The live camera feed shows object detection in action, highlighting a person with a green bounding box. Telegram notifications and motion detection controls are also visible.

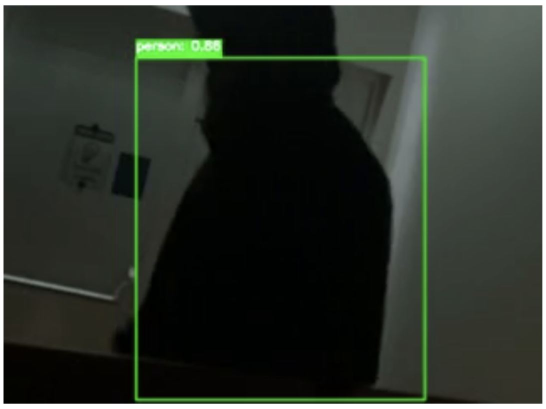


(a) Test scenario – A person crawling low under high visibility conditions to simulate a potential intruder while avoiding full-body visibility.

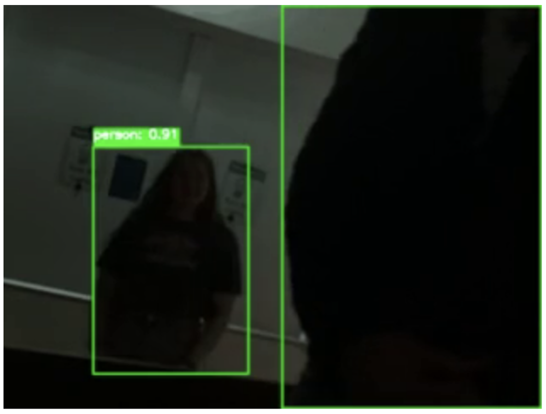


(b) Test scenario – Two people entering the frame simultaneously under high visibility conditions.

Figure 9: Test scenarios under high visibility conditions



(a) Test scenario – A person wearing a hood and walking through the monitored area under low visibility conditions.



(b) Test scenario – Two people entering the frame simultaneously under low visibility conditions.

Figure 10: Test scenarios under low visibility conditions