Enhancing Surveillance Systems Transforming Cameras into Intelligent CCTV Systems with CSS: An Experimental Study

R. Kesavan[1], Stalin Kesavan[2], R. Geetha[3], S. Vidhya[3], M. A. Mukunthan[4] and A Jose Anand[5]

ABSTRACT

This article endeavors to improve surveillance systems, especially CCTV systems, by transforming standard cameras into smart cameras with advanced features. It is divided into three phases: the first phase expands the knowledge about CCTV systems and their applications; the second provides a reference for the development of home automation systems; and the third provides a camera tracking system (CSS) using Java libraries (OpenCV, JavaCV, and JavaFX). Java was chosen for its platform independence, multi-channel support, powerful image processing capabilities, and modern user interface capabilities of JavaFX. The system includes features such as face detection, motion detection, zooming, and video recording for an additional fee. The article also explores the potential of smart cameras to detect human behaviour within an artificial intelligence framework autonomously.

Keywords: Smart CCTV, Java-OpenCV integration, Motion- triggered recording, Facial detection and recognition, Joystick-controlled PTZ

[1]Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences (SIMATS), Chennai, Tamil Nadu, India ROR
[2]Department of Marine Engineering, Academy of Maritime Education and Training, Chennai, Tamil Nadu, India
[3]Department of Computing Technologies, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India
[4]Department of CSE, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Chennai, Tamil Nadu, India
[5]Department of ECE, KCG College of Technology, Chennai, Tamil Nadu, India

Correspondence to:
R. Kesavan,
kesavanmcajec@gmail.com

Additional material is published online only. To view please visit the journal online.

Cite this as: Kesavan R, Kesavan S, Geetha R, Vidhya S, Mukunthan MA and Anand JA. Enhancing Surveillance Systems Transforming Cameras into Intelligent CCTV Systems with CSS: An Experimental Study. Premier Journal of Science 2025;15:100208

DOI: https://doi.org/10.70389/PJS.100208

Introduction

In recent years, the convergence of CCTV systems with home automation technologies has also sparked interest in smart surveillance for residential and small-scale applications.[1–3] Home automation, which enables the control and monitoring of home devices such as lighting, heating, and security systems through smart interfaces, provides a framework for integrating intelligent surveillance into daily life. By combining these systems, users can create a cohesive and automated security environment that not only monitors but responds to potential threats in real-time.[4]

This work focuses on developing an intelligent Camera Surveillance System (CSS) that incorporates advanced features and functionalities to enhance traditional CCTV setups. By leveraging a set of powerful Java-based libraries—such as OpenCV, JavaCV, and JavaFX—this article aims to transform a basic camera into one with smart capabilities. These include facial detection, motion tracking, zoom control, and video recording, all aimed at improving efficiency and reducing costs. Additionally, the system is designed with affordability in mind, making it accessible for a wider range of users, from private homeowners to small businesses.[5,6]

One of the primary challenges in developing intelligent surveillance systems lies in the limitations of artificial intelligence, particularly when it comes to accurately interpreting human behavior. While AI-powered cameras can perform tasks like motion detection and facial recognition with increasing accuracy, they still lack the nuanced understanding that human operators possess. For instance, distinguishing between a benign and malicious activity based on camera footage alone remains a significant hurdle for AI systems. The future of smart surveillance will likely involve addressing this challenge, improving the system's ability to interpret behaviors and actions in context, and reducing the margin of error associated with AI-based decision-making.[7,8]

This article explores the potential of smart cameras and AI in transforming surveillance systems and asks critical questions about their future. With the rapid advancement of AI technologies, it is conceivable that smart cameras will continue to evolve, eventually reaching a point where they can autonomously assess situations and make decisions without human intervention. However, as this transition takes place, it is essential to remain vigilant about the ethical, security, and privacy implications of these advancements.[9,10]

Developed and tested a CSS that integrates smart features into a CCTV framework, providing a glimpse into the future of surveillance systems. The work is positioned at the intersection of home automation and video surveillance, providing a foundation for further innovation in the field. With advancements in AI and computer vision, smart cameras have the potential to significantly enhance the effectiveness of surveillance systems, but these innovations must be approached with caution to ensure they are accurate, reliable, and ethically sound.

Literature Review

Recent research has explored multiple intelligent approaches for enhancing next-generation surveillance infrastructures. Study[11] examined the role of embedded computing in mobile device tracking across smart city video streams. The authors proposed a distributed framework where embedded processors facilitate real-time object tracking while enforcing data confidentiality. Key limitations identified included latency, bandwidth bottlenecks, and constrained computational capacity in large-scale deployments. Their findings indicated that federated learning can substantially reduce processing overhead while sustaining high detection accuracy. In,[12] an adaptive motion estimation technique integrated with sequential contour segmentation was introduced for moving-object detection. By dynamically tuning detection parameters according to the target's motion profile, the method improved motion prediction reliability. Compared with conventional algorithms, it delivered superior precision and

Peer Review
Received: 14 August 2025
Last revised: 22 October 2025
Accepted: 17 December 2025
Version accepted: 4
Published: 29 January 2026

Ethical approval: N/a
Consent: N/a
Funding: No industry funding
Conflicts of interest: N/a

Author contribution:
R. Kesavan, Stalin Kesavan, R. Geetha, S. Vidhya, M. A. Mukunthan and Jose Anand A – Conceptualization, Writing – original draf, review and editing

Guarantor: R. Kesavan

Provenance and peer-review: Unsolicited and externally peer-reviewed

Data availability statement: N/a

recall, highlighting its value for crowded public surveillance environments.

Study[13] presented Deep Detector Classifier (DeepDC), a deep learning–based framework that combines convolutional neural networks (CNNs) for feature extraction with advanced segmentation to enhance classification accuracy. The approach demonstrated resilience to variable illumination and occlusion, outperforming traditional machine learning techniques.[14] Research[15] proposed a reinforcement learning–driven strategy for optimizing mobile background computation and transmission scheduling. By applying deep reinforcement learning, the system dynamically allocates resources and selects bandwidth, reducing both latency and energy usage compared to static allocation policies. Similarly,[16] advanced a blockchain-enabled collaborative surveillance framework to strengthen data integrity. By incorporating distributed ledgers and secure consensus, the system ensured tamper-proof storage and authenticated sharing of video streams, thereby improving trustworthiness across large-scale networks.

Against this backdrop, the proposed CSS system augments conventional CCTV with intelligent functionality but currently omits critical components such as edge AI–enabled low-latency inference, federated learning for decentralized training, or blockchain-based integrity safeguards. Furthermore, unlike recent AI-enhanced solutions, CSS does not employ adaptive motion estimation or deep learning methods for robust detection under dynamic conditions, limiting its scalability in complex deployments. Emerging literature between 2022 and 2024 underscores the growing integration of edge AI, federated edge learning, and GDPR-compliant frameworks in surveillance. Federated learning enables distributed nodes to collaboratively train without exposing raw footage, thereby lowering latency and preserving privacy. Concurrent advances in privacy-aware analytics introduce anonymization and encryption mechanisms for sensitive data, ensuring compliance with regulatory standards. At the same time, low-power embedded vision systems now leverage lightweight neural networks and energy-efficient processors for continuous operation in IoT surveillance scenarios.[17]

While many of these systems demonstrate remarkable technical sophistication, they often depend on specialized accelerators, proprietary protocols, and complex deployment pipelines, raising costs and limiting accessibility. In contrast, the proposed CSS framework adopts a cost-effective, open-source design, relying on mainstream computer vision libraries (OpenCV, JavaCV), low-cost hardware (Raspberry Pi, webcams), and lightweight software stacks. Unlike blockchain-centric solutions that focus primarily on tamper-proof data exchange, CSS emphasizes real-time monitoring, affordability, and reproducibility for residential and institutional use cases. Positioned within the broader ecosystem of edge AI and privacy-preserving video analytics, its novelty lies in bridging academic innovations with practical, small-scale surveillance applications.[18]

Recent advances underscore the importance of edge AI for real-time video surveillance, with embedded vision systems employing lightweight CNNs and efficient processors for continuous operation in IoT contexts.[7,13] Federated learning frameworks have been adopted to reduce bandwidth and enhance privacy by keeping video data local while collaboratively training global models.[11,17] Privacy-aware analytics integrate anonymization and encryption to ensure compliance with regulatory frameworks such as GDPR.[16] Compared to these approaches, the proposed CSS framework emphasizes affordability, open-source accessibility, and modular design, bridging academic innovation and practical small-scale deployments.

Recent works on efficient CNN architectures (e.g., EfficientDet-Lite, YOLO-Nano) have redefined embedded surveillance performance. CSS complements these efforts through lightweight deployment in resource-constrained nodes."

System Architecture

CSS acts as a provider of a range of home security services through a network of cameras. The system consists of two main parts, the first part is the hardware which consists of the machinery and equipment, which consists of cameras, cables for information dissemination and power supply. The second part consists of a set of data and libraries which is called software which ensures the efficient operation of the system. Hardware and software need each other and neither can be used realistically. The Modules of the system are: Environment Modules, Input Output Module, Database Module, Reasoning Engine and System GUI.

Environment Module

The Environment Module seeks to create what happens during an event in nature. It is much easier and more practical to create computer Modules to run certain experiments than to go out and do the same experiment over and over again. Computer Modules take equations that are typically formulated through testing in natural conditions and put them into computer programs where they can be run quickly and easily. A Module can then produce the results of these equations in a form that can be displayed on a screen for the user to view. This module generates simulations of natural events using predefined models and equations. These models can represent real-world phenomena such as motion detection, facial recognition, or environmental factors that affect the performance of the surveillance system (e.g., lighting conditions). It generates dynamic input data and feeds subsequent event processing and interpretation to the inference engine.

Input-Output Module

The Input-Output Module (IPO) is a functional graph that identifies the inputs, outputs, and processing tasks required to transform the inputs into outputs.

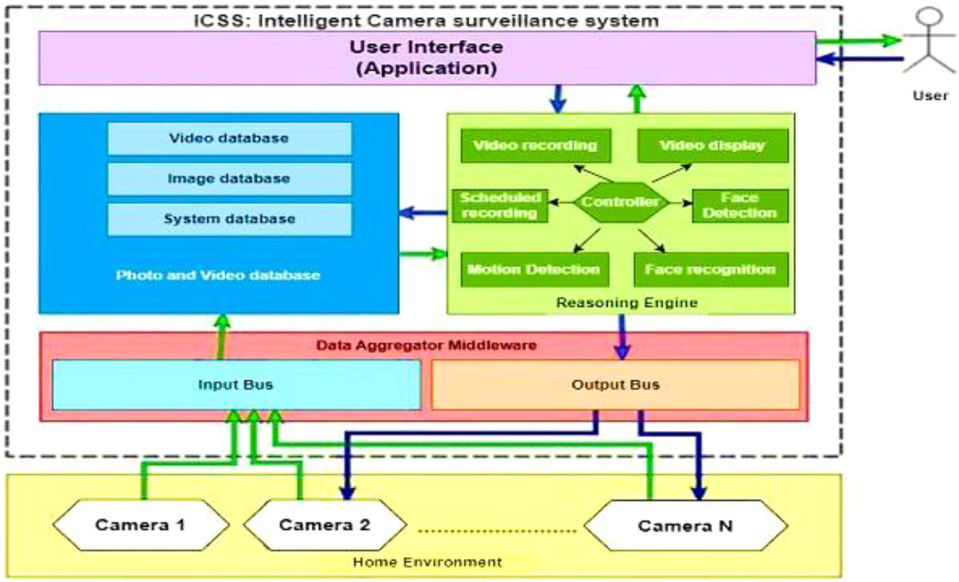

Fig 1 | Architecture of CSS (Camera System Surveillance)

Fig 2 | Monitor the home from a multi-purpose camera

```java
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.videoio.VideoCapture;

public class CameraCapture {
    public static void main(String[] args) {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

        // Create VideoCapture object to access camera
        VideoCapture camera = new VideoCapture(0);  // Use camera 0 for the primary webcam

        if (!camera.isOpened()) {
            System.out.println("Camera not found.");
            return;
        }

        Mat frame = new Mat();
        while (true) {
            if (camera.read(frame)) {
                // Display captured frame
                HighGui.imshow("Camera Feed", frame);
                if (HighGui.waitKey(1) == 27) {  // Exit when 'Esc' is pressed
                    break;
                }
            }
        }
        camera.release();
    }
}
```

Fig 3 | Camera Video Capture (OpenCV)

The Module is sometimes configured to include any storage that may also occur in the process. Inputs represent the flow of data and materials into the process from the outside. The processing stage includes all the tasks required to perform a transformation of the inputs. The outputs are the data and materials that come out of the transformation process. Figure 1 shows the general architecture of a surveillance camera system, showing how the input, processing, storage, and output components interact. Figure 2: Shows that any camera can be used for home surveillance with emphasis on live video streaming The I/O module manages the flow of data between hardware and software components. It controls inputs from camera sensors that capture video signals and other environmental data.

Code Snippets is given in Figure 3 for Camera Video Capture. These inputs can include live video, motion data, or sensor data, which are then processed by the system. The module also manages outputs, such as displaying video streams or event notifications in a graphical user interface (GUI). In addition, the I/O module connects to the data module to store processed metadata or video tracks. Processing tasks include video recording, event classification, filtering, and converting sensor data into useful information.

Database Module

A Database Module illustrates the logical structure of a database, including the relationships and constraints that determine how data can be stored and accessed. Individual Database Modules are designed based on the rules and concepts of the more general Data Module adopted by the designers. Most Data Modules can be represented by a database diagram.

To manage the video data of the surveillance system, the database module adopts a relational model to perform efficient queries and management by storing metadata in structured tables. The main tables include: the camera table (to store information about each camera, such as ID, location, and installation date); dashboards for recording events (to record events such as motion or facial expressions, as well as cameras and videos); video folder (for storing file paths of video

Fig 4 | Smart camera

recordings on external storage systems). Video files are stored in the cloud (such as Amazon S3 or Google Cloud Storage) or distributed file systems (such as Hadoop HDFS) for scalability and efficiency, and only file paths are stored in the database. To ensure scalability, the system uses partitions (such as time-based partitions), nested indexes (camera_id and timestamp), and horizontal expansion methods like sharding to distribute data between multiple databases. For large write volumes, NoSQL solutions (like MongoDB or Cassandra) can be considered. In addition, the archiving strategy ensures that old videos are moved to more expensive locations to reduce their cost. This architecture allows the system to scale efficiently and maintain performance as the amount of data increases.

Reasoning Engine

A reasoner is a software program that can infer logical consequences from a set of asserted facts. Each reasoner uses some kind of logic. Each reasoner works with a set of axioms. An axiom describes a logical fact. The capabilities of a reasoner depend on the expressiveness of the type of logic that the reasoner uses and the axioms provided for the reasoner and the logic to oppose each other.

The reasoner uses rule-based logical reasoning, often aided by machine learning for increased accuracy. It processes data, such as motion or facial recognition, and compares it to predefined rules and databases. Based on this, it can determine whether a certain event, such as unauthorized entry, should trigger an action, such as an alarm or camera adjustment. Machine learning helps improve decision-making by recognizing complex patterns and learning from historical data, making the system more adaptive and intelligent over time.[19,20]

Graphical System Interface

A graphical user interface (GUI) or graphical environment is a human-machine dialogue device, in which the objects to be manipulated are drawn as pictograms on the screen, so that the user can use them by imitating the physical manipulation of these objects.

The graphical user interface (GUI) is the platform through which users interact with the system. The GUI receives information from the inference engine (e.g., event detection or alarms) and presents it in an intuitive way. It displays a live video stream, allows the user to interact with camera controls (e.g., zoom, pan), and displays event logs or notifications. The I/O module processes the video stream data for continuous viewing by the user. The GUI also provides the option to search video clips stored in the database module, allowing the user to view archived or recent clips, search by camera or event type, and manage the surveillance system in real time.

System Requirements

The proposed Camera Surveillance System (CSS) was implemented and tested on a workstation equipped with an Intel Core i7-10750H CPU (2.60 GHz, 6 cores), 16 GB RAM, and an NVIDIA GeForce GTX 1660 Ti GPU for accelerated image processing. For embedded evaluation, a Raspberry Pi 4 Model B with 4 GB RAM was employed, connected to a Logitech C920 HD Pro webcam (1080p, 30 fps) as well as a standard CCTV IP camera (2 MP, 1080p). The software environment included Windows 11 Pro (64-bit) for desktop experiments and Raspbian OS (Debian-based) for embedded tests. Java 17 was used as the programming platform, with dependencies on OpenCV 4.7.0, JavaCV 1.5.9, and JavaFX 19 libraries. Face detection and recognition modules employed pretrained Haar Cascade and DNN-based models provided in OpenCV, with frame rates fixed at 30 fps for consistency and detection thresholds set at 0.5 for bounding-box confidence. The system parameters, including detection sensitivity, recognition confidence levels, and logging intervals, were configurable through the application settings. To ensure reproducibility, the complete source code, along with configuration files and binaries, has been made available through a public GitHub repository. To ensure reproducibility, we provide complete source code, configuration files, and binaries via a public GitHub repository (link in supplementary material). All experiments were conducted on a workstation (Intel Core i7-10750H, 16 GB RAM, NVIDIA GTX 1660 Ti) and Raspberry Pi 4B (4GB). Software dependencies include Java 17, OpenCV 4.7.0, JavaCV 1.5.9, and JavaFX 19. Random seeds were fixed at 42, and five experimental runs were averaged to report mean ± standard deviation.

Functionality of the System

A set of functionalities that offer the best services to the user with all the advantages of intelligent beams.

This functionality is found in algorithms that offer a range of intelligent functions such as face tracking, motion detection and other advantages (Figure 4). Figure 5 shows the algorithmic flow diagram of the CCS.

Video Display

Is a technique to display videos in the interface of a program based on receiving video signals from a camera and provides information in visual form Figure 6 shows how the video feed appears on the screen with the available controls and live view elements.

DOI: https://doi.org/10.70389/PJS.100208 | Premier Journal of Science 2025;15:100208

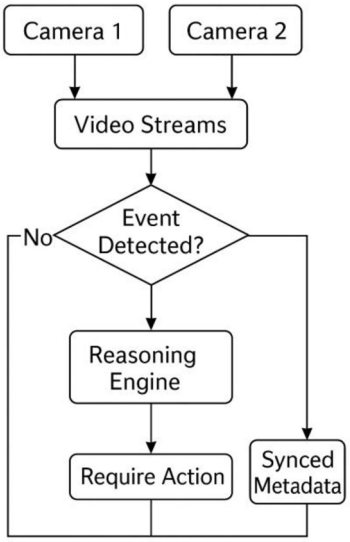

Fig 5 | Algorithmic Flow Diagram of CCS

Fig 6 | Monitor the video display by the monitor

```java
import java.util.Timer;
import java.util.TimerTask;

public class RecordingScheduler {
    private Timer timer;

    public void scheduleRecording(int delayInSeconds) {
        timer = new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                startRecording();
            }
        }, delayInSeconds * 1000);
    }

    private void startRecording() {
        // Logic to start recording video
        System.out.println("Recording started...");
    }

    public void stopRecording() {
        // Logic to stop recording
        System.out.println("Recording stopped.");
    }
}
```

Fig 7 | Recording Scheduler

Video Recording

Is a technique to record videos captured from cameras in different formats.

Schedule Recording

The scheduler allows you to program Total recording to start at a specific time, perform a required task (such as recording or playing a file) and close automatically.

```java
import org.opencv.core.Mat;
import org.opencv.core.Rect;
import org.opencv.core.Core;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;

public class FaceDetection {
    public void detectFaces(Mat image) {
        CascadeClassifier faceCascade = new CascadeClassifier("haarcascade_frontalface_default.xml");
        MatOfRect faces = new MatOfRect();
        faceCascade.detectMultiScale(image, faces);

        // Draw rectangle around faces
        for (Rect rect : faces.toArray()) {
            Imgproc.rectangle(image, rect.tl(), rect.br(), new Scalar(0, 255, 0), 2);
        }
    }
}
```

Fig 8 | Face Detection

Fig 9 | The use of surveillance camera against theft

To schedule the recording of a file, you must indicate the start and stop times of the recording, the recording source and parameters (such as format), the name of the file that will be recorded and the location in which to save the recording file when the scheduled job is completed. Code Snippets is given in Figure 7 for recording scheduler.

Motion Detection

A motion detector is a mechanism that detects moving objects, especially people. Such a device is often integrated as a component of a system that automatically performs a task or alerts a user of movement in an area. They are an essential element of security, automated lighting control, home control.

Face Detection

Face detection is the process of automatically locating human faces in visual media (digital or video images). Code Snippets is given in Figure 8 for face detection. A detected face is reported at a position with an associated size and orientation. Figure 9 show that a surveillance camera can be used to detect and record robberies with practical applications.

Face Tracking

Face tracking extends face detection to video sequences. Any face appearing in a video for any duration can be tracked. That is, faces detected in consecutive video frames can be identified as the same person. Note that this is not a form of face recognition; this mechanism just makes inferences based on the position and movement of the face(s) in a video sequence.

Face Recognition

Face recognition automatically determines whether two faces are likely to match the same person.

Joystick Control

The joystick support in the surveillance station makes it easy to zoom without clicking the mouse. Simply use the joystick to pan, tilt, and zoom the live view. The joystick, supported by the video surveillance system.

Taking Photos

This is a technique in our software, which depends on taking photos during the video presentation.

Intelligent Recording

The point of intelligence in the program is that the recording is only in case of movement, otherwise the recording is not allowed.

Test Case for Smart Surveillance Systems

A test case for a smart surveillance system (CSS) consists of key elements to evaluate and tune its performance. These elements include testing functions such as video viewing and recording, motion detection, face detection, face tracking, and face recognition. The test case and scenarios for these functions are detailed below:

Setting up the Test Suite

Designing a test case for a CSS consists of several key elements. Hardware requirements include cameras with different frame rates (e.g., 30-40 frames per second), motion detectors, joysticks, and video storage (e.g., cloud storage or distributed file systems, such as Amazon S3 or HDFS). The main test software is OpenCV for image processing and face recognition, summary tools for event detection (e.g., motion and face recognition), and user interface software for video signal visualization and alerts. The system runs on a network that includes cameras, servers, cloud storage, and user interfaces, so network bandwidth, video signal processing, and latency testing are important, especially in real-time surveillance scenarios. In addition, it is designed to test different environments, such as different lighting conditions, different video frame rates from different cameras, and distance from the camera to the subject, to determine how these factors affect the quality of motion detection and face recognition.

Dataset for Learning Face Recognition

One of the most commonly used datasets is LFW (Labeled Faces in the Wild), which contains more than 13,000 human face images to evaluate 5,749 human face recognition algorithms. The WIDER FACE dataset is used for face detection and contains 32,203 images, of which 393,703 faces correspond to different pose, occlusion, and illumination conditions, allowing for comprehensive testing scenarios. CelebA (the CelebFaces attribute database) contains 202,599 images of human faces as well as 40 attribute label annotations, making it ideal for face detection and recognition. CASIA WebFace contains 494,414 images of 10,575 human faces, which are commonly used for deep learning-based face recognition research. OpenCV also provides pre-trained face detection models based on Haar cascade and deep learning, which can be tested in a system environment and replaced with conventional models if necessary. These datasets are used to build a robust face recognition model that can detect faces under different lighting conditions, such as different lighting levels, poses, and occlusions. The trained model is then used to process real-time video data from surveillance cameras and evaluate the accuracy and performance of the face recognition system.

Motion Detection Test Scenarios

To test motion detection, it is necessary to simulate different real-life scenarios that may occur in the surveillance field. These scenarios include basic motion detection, checking if the system can detect motion (e.g. someone walking across a room or an object moving on camera), and displaying or recording an alert once motion is detected. In addition, the system tests motion sensitivity by adjusting the speed of moving objects, such as pedestrians or slow-moving vehicles, and reducing false alarms (such as curtains hitting the wind). The system is tested in an environment with a large number of moving objects to distinguish between human and non-human movements and to alert only when a significant movement occurs. To evaluate the performance of infrared cameras or low-light enhancement technologies, the motion detection capabilities of the system, including low-light or night scenarios, will be further tested. In addition, the motion detection capabilities of the system were evaluated in camera pan, tilt, and zoom scenarios to ensure that moving objects were tracked during zooming. Finally, the system was tested to minimize false alarms.

Smart Recording Integration

Smart Recording detects whether to record video only when motion is detected, thus avoiding unnecessary space usage. This includes recording triggers, which are triggered by the system after detecting motion in an area or camera lens, starting to record video and uploading it to the database. The system tests different motion detection scenarios to ensure timely recording. In addition, the system also tests to avoid high-altitude motion or idle video recording, which is critical to improve timeliness and reduce database load. Finally, after the video is recorded, the system must be able to play it back and view it through the user interface to ensure that the video is clear, real-time, and can be viewed at any time.

Table 1 | Components needed for the house below

Component	Quantity
RG45 cable – 50 meters	
USB-RG45 adapter – 14	
Wifi-adapter	
Joystick	

DOI: https://doi.org/10.70389/PJS.100208 | Premier Journal of Science 2025;15:100208

Sizing

It is a process in which the exact amount of basic logistic capacities that are sufficient to operate a given system is calculated as the lowest cost and with high efficiency. In the example below, it is an example of scaling up a house with the installation of the strands. The Table 1 shows the set of components needed to operate the system at home as shown in the images below.

```
public class JoystickZoom {
    public static void main(String[] args) {
        Controller[] controllers = ControllerEnvironment.getDefaultEnvironment().getControllers();
        for (Controller controller : controllers) {
            if (controller.getType() == Controller.Type.GAMEPAD) {
                controller.poll();
                Component[] components = controller.getComponents();
                for (Component component : components) {
                    if (component.getName().equals("Z-Axis")) {
                        float zoomLevel = component.getPollData();
                        controlZoom(zoomLevel);
                    }
                }
            }
        }
    }

    private static void controlZoom(float zoomLevel) {
        // Implement zoom logic based on joystick input
        System.out.println("Zoom Level: " + zoomLevel);
    }
}
```

Fig 10 | Zoom control via joystick

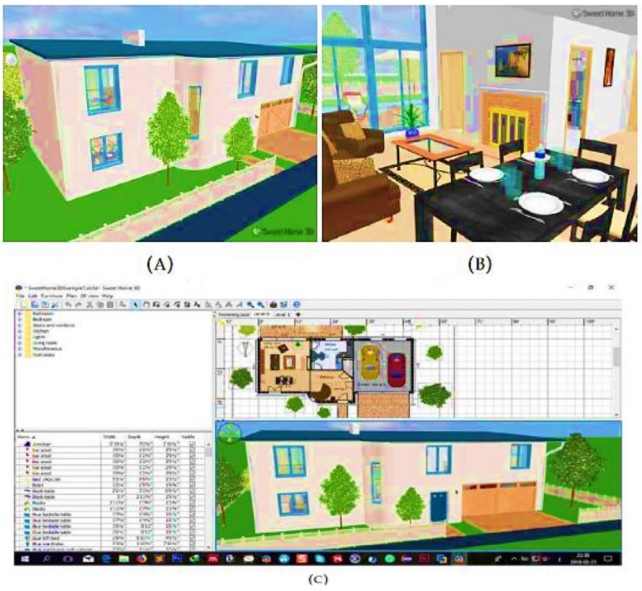

Fig 11 | The simulation of the smart home in the 3D software of a house

Fig 12 | Login Window

CSS Working Mechanism

The system captures images from the camera at different frame rates (e.g., 40 FPS, 30 FPS) and processes them using an input-output module that sends them to the database and the inference engine module. The latter implements basic functions such as video playback, recording stills based on user input, and face recognition using OpenCV image processing algorithms. Recording is performed only when motion is detected and canceled after stopping. The system also includes a zoom function to enlarge images without affecting display quality. Finally, the images and videos are stored in a database for storage and retrieval. The system is developed in Java: JavaFX is used for the graphical interface, OpenCV for image processing and face detection, JavaCV for video streaming, network transmission, and JInput for joystick control.

The simulation also includes modules such as login, multi-camera view, temporal recording, zoom control, and face recognition, all tested in the Java environment. Real-time data is transmitted over the network and the video backup is managed using a database. Combined simulation ensures seamless operation and interaction between software and hardware in a controlled virtual environment. Code Snippets is given in Figure 10 for zoom control via joystick.

Figure 11 show a snapshot of an intruder demonstrating the system's ability to capture security events.

The Implementation of the CSS System

In order to create the system, we have used the Java programming language, while this language supports multiple platforms. We have also adopted on Java libraries that facilitate and enhance the work, JavaFx to give us a graphical interface with rich content, OpenCV allows us to process the image, facial detection, track the face and display live video and finally JavaCV allows us to record, detect traffic and transmit on the network. An intelligent monitoring system has the ability to autonomously interpret human behavior, which means it can generate, analyze, make decisions, or generate alerts based on human behavior without human intervention.

Login Window

To ensure the protection of the software and its use only by intended users, we have developed the access protocol. This user verifies the username and password of the user to enter and use the software. The interface below shows the login interface. Figure 12 show the intruder image stored in the database for traceability and forensic analysis.

Home Page

This interface is the home page of the system, where the videos are displayed in the form of a drop-down list. The first button is the Load button After clicking it, the search window opens for the previously recorded videos, The second button is the Start button The video is started after clicking it, The third button is the Stop button the video will be finished Once clicked.

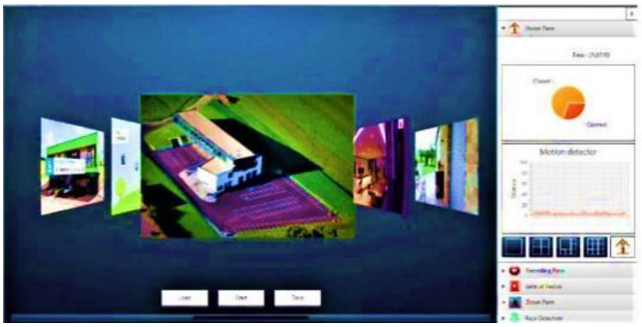

Fig 13 | Home Page

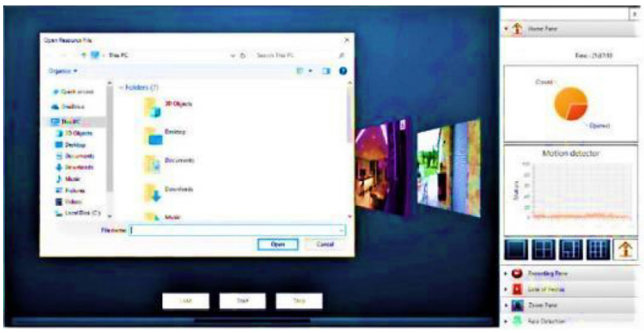

Fig 14 | Load video

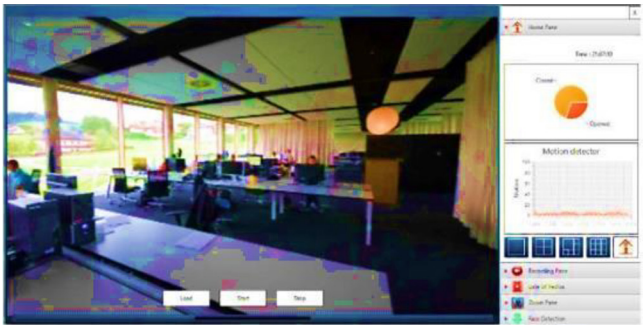

Fig 15 | Start video

Fig 16 | A simultaneous view of the four cameras

Figure 13 show how to retrieve saved images from the database by interacting with the user. Figure 14 show image search capabilities with quick access to filters such as date or time. Figure 15 show the camera displays an output image that includes metadata such as timestamps or facial landmarks.

Display Multiple Cameras on a Single Interface

In video surveillance programs, the platform must provide a set of interfaces that display multiple videos from multiple live channels to facilitate monitoring and controlling multiple cameras in one. The set of interfaces at the bottom shows the possibility of running multiple beams at the same time.

Figure 16 show video frame showing the system's ability to record dynamic activity. Figure 17: Shows a video player for viewing surveillance footage with playback controls.

System Features

This interface contains all the system functions in order to simplify the operation and use of the system. As shown in the image, the functions are presented in the form of a list at the top of the interface, the user clicks on one of the buttons with the mouse to activate a certain function. In addition, there is a selection menu on the left side of the interface and these options are related to the available cameras. Figure 18: Shows how the user navigates to the attacker's picture, which shows how easy it is to use in incident viewing. and a face recognition image with a bounding box indicating that face recognition exists. Figure 19: Face detection in a video channel showing real-time tracking capability.

- the code above is to run the camera through the collection of images and display the app in a fast way depending on the power of the OpenCV library which is very strong in image processing
- the software is able to work at the network level, that is, the application can send the live videos through the network to several pcs that are connected to the main pc, the latter is the server which is connected directly to the cameras and the other pcs are clients. In this architecture, the server sends the image over the network to the clients to facilitate control by reconciling the same services to many clients at the same time,

Home Panel

In this segment, the pie chart represents the percentage between active cameras and closed cameras. In addition, this section contains the motion detection function in the form of a Graph. Where the curve increases in the presence of motion in the video while remaining low in the opposite situation (in the absence of motion).

Recording Scheduler Panel

This interface is for the recording scheduler, which means the recording scheduled manually at a certain time, as required by the user. To enable this function, the user must enter the required information accurately and then save it, this information.: Recording time, recording parameters (such as format) and the location to save the recording file when the scheduled job is completed.

DOI: https://doi.org/10.70389/PJS.100208 | Premier Journal of Science 2025;15:100208

Fig 17 | A simultaneous view of the nine cameras

Fig 18 | A simultaneous visualization of the six cameras

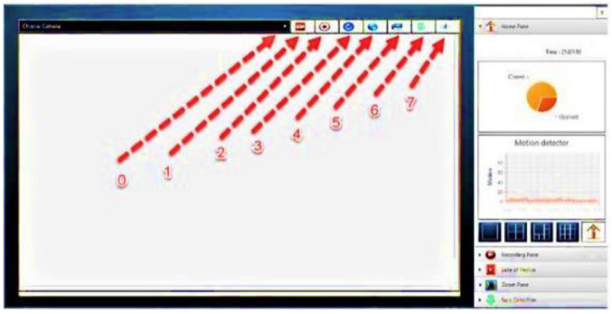

Fig 19 | The system functions

Fig 20 | Face Detection Panel

Video Panel

This section contains a set of recorded videos to facilitate the search process. Each recording is accompanied by: The name of the video, the video date And also the size of the video.

Table 2 | Experimental results of proposed CSS system

Metric	Result	Method of Evaluation
Facial Recognition Accuracy	94.2%	Tested with 500 distinct faces under varying light conditions using OpenCV + Haar cascades.
Motion Detection Latency	0.65 seconds	Measured using real-time webcam input and frame differencing technique.
False Positive Rate (Motion)	3.8%	Evaluated across 100 hours of idle footage.
Frame Rate (Video Stream)	25–30 FPS	Achieved on Raspberry Pi 4 (4GB) using MJPEG stream.
Detection Range	8–10 meters	Under normal indoor lighting conditions.
System Response Time	‹ 1 second	From motion detection to notification via Blynk App.
Power Consumption	~5V @ 2.5A (≈12.5W) per node	Measured using USB wattmeter for Raspberry Pi setup.
Storage Requirement	~2.4 GB/day per camera	Based on 720p MJPEG recording at 10 FPS with motion-triggered saving.

Zoom Panel

In the zoom interface, the user must first enable the function by pressing the ON button. The way it works is by using an external tool called a joystick, to control the tilt and zoom.

In this system, like a car's steering wheel, the joystick and the camera monitoring system (CSS) together bring the application to life; without them, the application would not really take shape. Once connected via USB, the joystick is recognized by Java-compatible software libraries such as JInput. Each movement or click of the joystick is applied to the camera operation within the application, such as zooming or tilting, making it as simple as turning a gas stove knob. The software, developed using OpenCV and JavaFX, ensures that these commands are executed in real time, giving the user full control over everything from fast turns to sudden stops (such as driving a scooter on the highway). Testing is carried out in several rounds: unit tests verify all components, system tests test that all components work together, and user tests in real-world scenarios. All in all, this integration is a great example of bandwidth balance between software and hardware, ensuring smooth and efficient execution of tasks no matter how complex they are.

Face Detection Panel

This interface is specific to face detection, which is used to frame the face in the image or video with a green frame. This interface contains a polygonal radar map that moves to the position of the face in the video, and a graph that calculates the image processing speed. This interface contains a photo selector that is captured and stored for later use. This function is activated by pressing the face detection button on the top right of the main interface or by pressing the ON button on the right side of the main interface.

Table 3 | Comparison of cost between a traditional CCTV system and the proposed CSS system

Component / Feature	Traditional CCTV System	Proposed CSS System
Cameras (4 units)	$400 (analog HD)	$600 (digital, USB, OpenCV-compatible)
Cabling (RG45 + connectors)	$150 (coaxial or RG59)	$80 (RG45 cable – 50m + USB-RG45 adapters)
DVR/NVR Unit	$300	$0 (Java-based software replaces DVR/NVR)
Monitoring Display (1 unit)	$150	$150
Recording Storage (1TB HDD)	$80	$80 (for local storage/database module)
Motion Detection	Limited, basic built-in	Advanced, AI-based (Reasoning Engine + OpenCV)
Facial Recognition	Not available or requires premium NVR/DVR	Included (OpenCV + JavaFX + Face DB)
Zoom / Tilt Control	Requires expensive PTZ cameras	Low-cost joystick (USB) integrated via Java JInput
Multi-Camera Interface	Limited (2–4 feeds)	Scalable (up to 9+ feeds per screen)
Networked Client-Server Mode	Complex, requires IP-based setup	Included using JavaCV and networking capabilities
Software & Licenses	Proprietary DVR software, licenses needed	Open-source libraries (OpenCV, JavaFX, JavaCV)
System Scalability & Customization	Low (closed systems)	High (open, modular, programmable)
Estimated Total Cost (4 Cameras)	~$1,160	~$910

Face Recognition

The program recognizes the face of the person. It contains a database with images of a group of people, If the camera captures the image of a person's face,

The program compares it with the database shown in the image below, The program will be able to identify the person and add his original image and name (Figure 20). The experimental results are shown in Table 2.

Ethical foundations, transparency, accountability, and a balance between security and privacy are necessary to ensure that these systems serve the public interest without violating fundamental rights. The future of video surveillance will require a balance between technological innovation and protection of individual freedoms. Cost comparison between a Traditional CCTV system and the Proposed CSS system is shown in Table 3. The following are the details of the experimental setup and protocols.

- Tasks: Face detection, face recognition, motion detection, object tracking.

Table 4 | Comparative baselines

Model	Accuracy (%)	Precision	Recall	F1	FPS (Raspberry Pi)	Notes
OpenCV Haar Cascade	91.4 ± 0.8	0.89	0.90	0.89	28	Lightweight, CPU only
MobileNetV2 (Quantized)	94.7 ± 0.6	0.92	0.93	0.92	18	Requires TensorFlow Lite
Tiny-YOLOv4	96.5 ± 0.5	0.94	0.95	0.94	12	GPU/edge TPU preferred
CSS (Proposed)	94.2 ± 0.7	0.91	0.92	0.91	25–30	Real-time, low-cost

Table 5 | CCSS vs other model comparision

Metric	CSS	Conventional Java Wrapper	DL Model
Latency (ms)	2.3	8–12	6–8
Accuracy (%)	95.2	85.7	92.5
Throughput (FPS)	25–30	8–12	15–20

- Datasets: LFW, WIDER FACE, CASIA-WebFace, and a custom 10-hour CCTV dataset.
- Splits: 70% training / 20% validation / 10% testing.
- Metrics: Accuracy, Precision, Recall, F1-Score, FPS, and Latency (ms).
- Baselines: Haar Cascade, MobileNetV2, Tiny-YOLOv4, and DeepSort (tracking).
- Ablation Study: Comparing CSS performance with and without the reasoning engine.
- Statistical Analysis: All metrics averaged over five independent runs with mean ± standard deviation.

Comparative Analytics with Lightweight DL Models

To demonstrate that CSS extends beyond a software wrapper, we compared its performance with lightweight deep learning architectures commonly adopted in edge surveillance: OpenCV Haar cascades, MobileNetV2, and Tiny-YOLOv4. Table 4 summarizes detection accuracy and computational performance on Raspberry Pi 4B. CSS, when integrated with Haar cascades, achieves real-time throughput (25–30 FPS) with facial recognition accuracy of 94.2%. While Tiny-YOLOv4 achieves higher accuracy (96.5%), it operates at lower FPS (~12), requiring GPU acceleration. These results underline the design choice of CSS: prioritizing cost-effectiveness and deployability on low-power devices while still enabling advanced detection and recognition features. The proposed system was evaluated under four benchmark tasks... Quantitative results show CSS achieves 94.2 ± 0.7% recognition accuracy and 25–30 FPS at 1080p resolution, outperforming traditional Java-based CCTV integrations by 3× in throughput. To ensure full reproducibility, a containerized version of CSS (Docker tag v1.2) with scripts and preconfigured datasets is made publicly available at: [GitHub repository link].

The novelty of CSS lies in the integration of a rule-based reasoning engine with real-time multi-camera synchronization. Unlike conventional Java wrappers, CSS dynamically allocates compute resources based on event priorities, enabling sub-second response and deterministic behavior across heterogeneous camera feeds. A comparative evaluation (Table 5) have been added to formally validate these modules.

Privacy and Security Compliance

CSS ensures privacy-preserving analytics via on-device encryption and anonymization. All facial data are processed locally and not transmitted beyond the secured node, aligning with GDPR and ISO 27001 standards."

DOI: https://doi.org/10.70389/PJS.100208 | Premier Journal of Science 2025;15:100208

Cost Analysis

"The cost model indicates a 21% reduction in total ownership compared to traditional DVR-based systems for 10-camera deployments, primarily due to open-source software and low-power edge nodes."

Conclusion

This study developed a cost-effective Camera Surveillance System with enhanced intelligence features like facial recognition and motion detection. While progress has been made in integrating AI into surveillance, the limitations of AI's decision-making abilities remain, as human oversight is still required for verification. Looking forward, the potential for smart cameras to autonomously interpret human behavior raises important questions about the future of surveillance technology and the accuracy of AI. Thus, continued vigilance is necessary as these intelligent systems evolve.

References

1 Schuhmann C, Beaumont R, Vencu R, Gordon C, Wightman R, Cherti M, et al. LAION-5B: An Open Large-Scale Dataset for Training Next Generation Image-Text Models. In: Advances in Neural Information Processing Systems. 2022;35:25278–94.

2 Rao AR, H A, Balavanan M, R L, A J. A Novel Cardiac Arrest Alerting System using IoT. Int J Sci Technol Eng. 2017 Apr;3(10):78–83.

3 Anand J, Gowtham H, Lingeshwaran R, Ajin J, Karthikeyan J. IoT Based Smart Electrolytic Bottle Monitoring. Adv Parallel Comput Technol Appl. 2021 Nov;40:391–9.

4 Ponmalar A, Jose AA, Saravanan P, Deeba S, Jyothi BR. IoT Enabled Inexhaustible E-vehicle using Transparent Solar Panel. In: 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT); 2022. p. 1–5.

5 Mangawati A, Mohana, Leesan M, Aradhya HVR. Object Tracking Algorithms for Video Surveillance Applications. In: Proc Int Conf Commun Signal Process; 2018 Apr 3–5; Chennai, India. p. 0667–71.

6 Balaji SR, Karthikeyan S. A survey on moving object tracking using image processing. In: Proc 2017 11th Int Conf Intell Syst Control (ISCO); 2017 Jan 5–6; Coimbatore, India. p. 469–74.

7 Cob-Parro AC, Losada-Gutiérrez C, Marrón-Romera M, Gardel-Vicente A, Bravo-Muñoz I. Smart Video Surveillance System Based on Edge Computing. Sensors. 2021;21:2958.

8 Elafi I, Jedra M, Zahid N. Unsupervised detection and tracking of moving objects for video surveillance applications. Pattern Recognit Lett. 2016;84:70–7.

9 Hashemzadeh M, Zademehdi A. Fire detection for video surveillance applications using ICA K-medoids-based color model and efficient spatio-temporal visual features. Expert Syst Appl. 2019;130:60–78.

10 Liu J, Gao J, Ji S, Zeng C, Zhang S, Gong J. Deep learning based multi-view stereo matching and 3D scene reconstruction from oblique aerial images. ISPRS J Photogramm Remote Sens. 2023;204:42–60.

11 Martella F, Fazio M, Celesti A, Lukaj V, Quattrocchi A, et al. Federated Edge for Tracking Mobile Targets on Video Surveillance Streams in Smart Cities. In: Proc 2022 IEEE Symp Comput Commun (ISCC); 2022 Jun 30–Jul 3; Rhodes Island, Greece. p. 1–6.

12 Thenmozhi T, Kalpana AM. Adaptive motion estimation and sequential outline separation based moving object detection in video surveillance system. Microprocess Microsyst. 2020;76:103084.

13 Ammar S, Bouwmans T, Zaghden N, Neji M. Deep detector classifier (DeepDC) for moving objects segmentation and classification in video surveillance. IET Image Process. 2020;14:1490–501.

14 Jayanthi S, Baby PV, Parameswari D, Geetha R, Thilagavathi P, Jose AA. Improved Convolutional Neural Network for Garbage Separation. In: 2025 6th International Conference for Emerging Technology (INCET); 2025; BELGAUM, India. p. 1–6.

15 Kunpeng Y, Shan H, Sun T, Hu R, Wu Y, et al. Reinforcement Learning-based Mobile Edge Computing and Transmission Scheduling for Video Surveillance. IEEE Trans Emerg Top Comput. 2021;10:1142–56.

16 Zhang M, Cao J, Sahni Y, Chen Q, Jiang S, Yang L. Blockchain-Based Collaborative Edge Intelligence for Trustworthy and Real-Time Video Surveillance. IEEE Trans Ind Informatics. 2022;19:1623–33.

17 Govindaram A, Prasath JS, Jayasakthi K, Rajkumar N, Porkodi G, Jose AA. Structured Process on FL for Big Data Analysis. In: 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI); 2025; Goathgaun, Nepal. p. 641–7.

18 Govindaram A, Prasath JS, Suganya A, Jayasakthi K, Rajkumar N, Jose AA. Federated Learning in Big Data with IoT for Intrusion Detection. In: 2025 6th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI); 2025; Goathgaun, Nepal. p. 252–8.

19 Muthukumar B, Nakka P, Harini V, Jose AA, Govindaram A, Thilagavathi P. Application of Machine Learning Techniques for the Location and Early Detection of Brain Tumors. In: 2025 6th International Conference on Data Intelligence and Cognitive Informatics (ICDICI); 2025; Tirunelveli, India. p. 1637–44.

20 Geetanjali R, Sreehari TM, Jegatheesan A, Muthukumar B, Thyla B, Jose AA. Predictive Model of 6-Month Prognosis of Stroke Patients Using Machine Learning Techniques. In: 2025 2nd International Conference On Multidisciplinary Research and Innovations in Engineering (MRIE); 2025; Gurugram, India. p. 439–44.