Real-Time Breathing Anomaly Detection In a Smart Room with Mobile Robot

Final Report

TU856
BSc in Computer science

Yasser Alshimmary
D21127124

Supervisor
Ciaran O'Driscoll

School of Computer Science
Technological University, Dublin

17/04/2025

Abstract

This project builds a smart hospital room system with a mobile robot to monitor patients in real time. The focus is on filtering noise from the sound sensor to improve the quality of health monitoring. In busy hospital rooms, background noise can make it hard to detect important patient sounds like coughing. To solve this, the system uses two filtering methods: Kalman Filter and Isolation Forest. Both methods are tested to see which one works better in removing noise and detecting abnormal sounds.

The smart room uses the CrowPi with sensors to collect light, temperature, and sound data. The MasterPi robot moves close to the patient to collect better sound readings. Three sound features are used: decibel (dB), root mean square (RMS), and zero-crossing rate (ZCR). These features help the system understand if a sound is normal or could be a sign of a problem.

The system uses Python, MQTT, and PostgreSQL to collect, filter, and store sensor data. A web dashboard shows alerts to hospital staff in real time. Test results show that Isolation Forest works better when sound features are filtered before training. The robot also improves accuracy by getting closer to the sound source. This project shows that combining sound filtering with robot movement can help monitor patients more accurately and reduce false alarms in hospital settings.

Declaration

I hereby declare that the work described in my dissertation is, except where otherwise stated, entirely This own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

___**Yasser Alshimmary**_______

Student Name

Date 06/12/2024

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Mr. Ciaran O'Driscoll, for his guidance, patience, and support throughout this project. His advice and feedback have been invaluable.

I also thank my family for their encouragement and understanding, especially during busy times. Their support motivated me to stay focused and complete my work.

Finally, I am grateful to the School of Computer Science at TU Dublin for providing the resources and environment to help me carry out this project.

Table of Contents

1. Introduction

1.1. Project Background

Healthcare technology has improved quickly in recent years, especially after the COVID-19 pandemic. Hospitals had many problems, such as a high risk of infection and a shortage of staff. To help with this, some hospitals started using robots to take temperatures and deliver supplies, reducing direct contact with patients.
Robots have already been used for jobs like checking temperatures and delivering supplies. However, most of these robots were used separately, not as part of a full system. Today, many modern health monitoring systems use technologies like wireless sensors and the Internet of Things (IoT). These systems can send alerts when something goes wrong. [1]
One useful tool for building such systems is the Raspberry Pi. It is small, cheap, and powerful enough to collect sensor data, process it, and send it to healthcare workers in real time. This is helpful for elderly patients who want to be monitored from home instead of going to the hospital often.[2]
Some advanced systems already combine robots with patient monitoring. For example, Boston Dynamics' "Dr. Spot," is a robot that checks body temperature without touching the patient. It moves around and helps reduce infection risks.[3]
This project builds on those ideas. It aims to develop a smart hospital room combined with a mobile robot. The smart room uses sensors to check the room and the patient's health all the time. The robot helps by going closer to the patient to get clearer readings and avoid background noises from the room. This makes patient monitoring more accurate, especially in noisy hospital areas. Filtering out background noise is very important because unclear sensor readings can cause delays in medical care or trigger false alerts. This project aims to improve the quality of data so hospital staff can make better and faster decisions.

1.2. Project Description:

This project aims to develop a smart hospital room system that works with a mobile robot to monitor patients. The system will continuously check patients' vital signs, provide real-time data, and help reduce the workload on hospital staff. It combines a smart room with sensors and a mobile robot that can collect additional details from patients.

The smart room uses a **CrowPi Raspberry Pi** as the main platform. This device connects to sensors that measure important data such as room temperature, room light, Sound, and patient health data. The smart room continuously monitors this data to detect any abnormal readings. If something unusual is found, the system can send alerts to hospital staff, allowing them to act quickly.

The mobile robot used in this project is the **MasterPi Hiwonder AI Vision Robot Arm**. This robot is equipped with wheels for movement, a robotic arm for tasks, and a camera for vision. It is programmed to navigate the room, avoid obstacles, and visit specific patients when needed. The robot can perform close-up checks, such as measuring breathing rates, temperature, or sound, to confirm any alerts from the smart room. It also works together with the smart room by sharing its findings.

One important part of this project is reducing background hospital sounds that may affect sensor readings. The robot helps by going closer to the patient to get clearer readings and avoid background noises from the room. This makes patient monitoring more accurate, especially in noisy hospital areas.
The smart room and the robot will talk to each other using a wireless connection between a client and a server. This allows them to share data in real time. With this setup, the robot can check or give more details about any health problems the smart room finds.
This project tries to make patient care better, lower the chance of infections, and save time and effort for hospital staff. It uses low-cost tools like CrowPi and MasterPi, which makes the system easy to use and grow. This makes it a good and helpful idea for today's healthcare needs.
The project has three main areas:

- Data Collection and Filtering
- Robot Navigation, Vision and Obstacle Avoidance.
- Communication

Unlike other systems that only monitor patients or use robots separately, this project combines both into one solution. The smart room uses smart room to check health data all the time, while the MasterPi robot moves around to take close-up readings and confirm alerts. It allows for faster responses, better safety, and more complete monitoring than other systems.

1.3. Project Aims and Objectives

This project compares Kalman Filter with Anomaly Detection and Isolation Forest to find the best method for removing noise and detecting abnormal sensor readings in real-time. The goal is to build a smart hospital system that combines a smart room and a mobile robot to monitor patients. This system will help hospital staff by providing real-time health updates, reducing delays, and improving patient safety.

the project focuses on three main areas:

- Data Collection and Filtering: The smart room and robot use sensors to collect health data. The system uses filtering methods to clean the data and find any unusual changes. The goal is to evaluate which method is better for cleaning sensor noise and detecting real abnormal values in real-time patient monitoring systems.

- Robot Navigation, Vision, and Obstacle Avoidance: The robot is programmed to move safely in the room using vision, Navigation, and obstacle sensors. It detects patients and gets close to them to collect better data.

- Communication: The smart room and robot communicate with Wi-Fi to allows them to send sensor data, alerts, and commands in real time. A web dashboard shows this information to hospital staff, so they can take action quickly.

1.3.1. Objectives:

The following is a list of the main objectives:

- Design a system where the smart room and robot work together for patient monitoring.
- Ensure the robot can move safely and collect accurate health data.
- Build a wireless client-server communication system to send alerts and share data between the smart room, robot, and hospital staff.
- Use filtering methods to reduce background hospital noises in sensor readings and detect health problems more accurately.
- Test and evaluate the full system in a simulated hospital room to check how it works under real conditions.

1.4. Project Scope

This project focuses on developing a smart hospital room and a mobile robot that work together to monitor patients and support hospital staff. The smart room uses sensors to check room conditions such as temperature, light level, sound, and patient health data. If something unusual is detected (like loud noise or a sudden drop in temperature), it sends an alert to hospital staff.

One of the most important parts of this project is comparing two techniques for filtering and anomaly detection. Kalman Filter with Anomaly Detection is used to clean noisy data and detect unexpected changes in real time. Isolation Forest is used to detect rare abnormal values but does not remove small noise. This comparison helps to understand which method is more accurate and reliable for patient monitoring, where both clean data and quick response are important.

The mobile robot moves around the room to perform close-up checks on the patient. It could collects health data such as body temperature, heart rate, oxygen level, and sound. It also helps reduce sensor noise by getting closer to the patient, which is especially useful for improving the accuracy of readings.

This system can be useful in real-life situations such as ICUs (Intensive Care Units), isolation rooms, and elder-care centres, where patients need to be watched all the time and staff need to limit physical contact for safety.

In the future, the system can be expanded to work across multiple rooms or even be adapted for home healthcare, making it a flexible and scalable solution.
What does project not cover:

- Not for Advanced Medical Diagnosis: The system will not diagnose illnesses.
- No Physical Medical Treatments: The system will not suggest treatments.
- Limited to Indoor Hospital Rooms: This project is limited to monitoring in one room, not across an entire hospital.
- Not Focused on Long-Term Data Storage: While the system can share real-time data with hospital staff, it is not designed to store or analyse long-term health records.

1.5. Project Management:

The project is designed to guide the development of the Smart Hospital Room system using a step-by-step approach. The plan is divided into clear phases, ensuring steady progress and enough time for testing and refinement.
The project follows the Incremental Development Methodology, which allows each module to be developed, tested, and integrated before moving to the next stage. The updated Gantt chart visually represents the timeline, tasks, and milestones.

1.6. Thesis Roadmap:

- Introduction: Gives a short summary of the project, explains the problem in hospitals, and shows how the smart room and robot system can help improve patient care and reduce staff workload.

- Literature Review: Reviews other healthcare technologies like IoT systems, robots, and sensors to show what has already been done and how this project is different by combining smart rooms with mobile robots and real-time data filtering.

- System Design: Describes how the smart room and mobile robot are designed using low-cost tools. It explains how hardware and software work together to collect, filter, and share sensor data to support patient monitoring.

- Prototype Development: Shows how the system is built step by step, including setting up sensors, programming the robot, and creating the communication system. This helps prove that the design can work in real life.

- Testing and Evaluation: Tests if the system can reduce noise in sensor data, detect health problems quickly, and help the robot move safely. This section checks if the project goals are met in real-world-like conditions.

- Conclusion and Future Work: Summarizes the main results and how the project helps in healthcare. It also gives ideas for future improvements, like using the system in more rooms or for home care.

1.7. Conclusion:

This project aims to solve real problems in hospital care by combining a smart room with a mobile robot. It uses low-cost devices to collect health data, filter noise, and share alerts in real-time. The system helps hospital staff respond quickly and reduce infection risks. One of the most important parts is filtering sensor data to find real health problems and avoid false alarms. The project compares Kalman Filter with Isolation Forest to find the best method for this task. It includes real-time data collection, robot movement, and machine learning. The clear plan, low-cost tools, and focus on accuracy make the project a strong and useful solution. It is a full system that can improve patient care and safety.

1. Literature Review

2.1. Introduction

This section looks at research and technologies that support this project. The purpose of this review is to understand what has already been done in the field of patient monitoring and to identify the technologies can be used in this project. This will help to ensure that the project is designed effectively and improves upon existing solutions. This design supports fast alert response and improves data accuracy by reducing noise in sensor readings, which is important in noisy hospital environments.

2.2. Alternative Existing Solutions to Your Problem

here are several existing solutions that aim to monitor patient vital signs and improve healthcare systems. These solutions have advanced in technology, but they still have limitations compared to the goals of This project. Below are five examples of such projects:

2.2.1. Dr. Spot by Boston Dynamics:

Dr. Spot is a robot equipped with infrared cameras and sensors to measure heart rate, oxygen saturation, respiratory rate, and temperature without contact. It is designed for mobility and adaptability in hospital triage settings. The robot has good vision capabilities and helps reduce infection risks by minimizing physical contact.
Limitations: Dr. Spot does not include a smart room system for continuous monitoring and is mainly designed for emergencies or pandemic situations.
Comparison: This project improves on this by integrating smart room functionality for constant health tracking, making it suitable for routine hospital care. [4]

2.2.2. Home-care nursing controlled mobile robot with vital signal monitoring:

The Home-Care Nursing Controlled Mobile Robot uses sensors like ECG, temperature, and oxygen saturation sensors to collect vital data. Its system design includes a mobile robot capable of navigation and assisting patients with mobility. Data is processed and displayed for caregivers.

Limitations: it does not focus on automated alerts or integration with a centralized system. Its strength lies in providing real-time patient monitoring in home-care environments.
It is limited to static data collection and lacks the ability to monitor non-contact indicators like skin colour or integrate with hospital systems.
Comparison: This project addresses these gaps by combining robotic capabilities with smart room technology to provide a more comprehensive hospital-based solution.[5]

2.2.3. The Non-Contact Video-Based Vital Sign Monitoring System

It uses video cameras with ambient light to measure heart rate and respiratory rate. The system is designed for non-invasive, contactless monitoring in hospitals and reduces discomfort and infection risks. Its vision capabilities are strong, relying on image processing algorithms to capture subtle changes in facial blood flow.

Limitations: It cannot track other vital signs like temperature or oxygen saturation and lacks the flexibility of robotic mobility.
Comparison: This project addresses these weaknesses by combining vision-based monitoring with traditional sensor data and adding a mobile robotic platform for more dynamic applications.[6]

2.2.4. The Smart Healthcare Monitoring System Using Raspberry Pi on IoT Platform

It integrates sensors such as ECG, respiration, temperature, heart rate, and accelerometers with a Raspberry Pi. The system transmits data over the Internet for real-time monitoring and includes alert mechanisms for abnormal readings.
Limitations: While it is highly effective for IoT-based monitoring, it does not feature mobility, smart room integration, or hybrid monitoring through vision systems.
Comparison: This project extends its functionality by introducing a mobile robot with enhanced vision and seamless interaction within hospital environment[7]

2.2.5. The IoT-Based Health Monitoring System Using Raspberry Pi employs sensors

It uses sensors like heartbeat, body temperature, ECG, blood pressure, and patient position monitors connected to a Raspberry Pi. It processes and sends data via IoT to caregivers, providing remote monitoring capabilities. The system is affordable and efficient for home care or small medical setups.
Limitations: It lacks robotic mobility, vision-based analysis, and advanced AI algorithms.
Comparison: This project enhances this by adding a mobile robot with hybrid monitoring capabilities and better hospital integration[8]

2.2.6. Comparison with This Project

These existing solutions solve problems like remote monitoring and contactless health checks, but they have limits in how flexible, connected, and scalable they are. This project is different because it combines smart room technology with a mobile robot and uses both sensors and cameras for monitoring.
This combination allows the system to detect problems more accurately, send real-time alerts, and help staff respond faster.
A key improvement in this project is the use of filtering methods to reduce sensor noise, which helps detect real health problems more accurately.

2.3. Technologies researched

2.3.1 Programming Languages

- Python: Python is the primary programming language used in this project because of its simplicity and versatility. It supports a wide range of libraries and frameworks specifically designed for robotics, vision systems, and data processing.
What It’s Used For:

- Programming the MasterPi Hiwonder robot for navigation and object detection.

- Handling sensor data from the smart room using the CrowPi Raspberry Pi platform.
- Implementing algorithms.

Why Python:

- Python provides libraries such as OpenCV and NumPy for numerical computations, making it ideal for processing sensor data.
- Frameworks enable easy integration of machine learning algorithms.
- Python’s simplicity ensures faster development and easier debugging.

Support:

- Python is fully compatible with the Raspberry Pi OS, the operating system running on the CrowPi platform.[9]
- Libraries like gpiozero and pigpio make it easy to interact with the GPIO pins on the Raspberry Pi, essential for connecting and managing sensors.[10]

- JavaScript: In addition to Python, JavaScript is useful for building web-based interfaces for hospital staff to monitor patient data remotely. Can be integrated with Python via backend frameworks to create an interface for the system.

2.3.2 Operating Systems

2.3.2.1. Raspberry Pi OS: Raspberry Pi OS:

It is a Linux-based operating system used on the Raspberry Pi in this project. It is lightweight and works well with IoT devices. It supports Python programming, which is the main language used for writing code for the smart room. Raspberry Pi OS also supports libraries and tools like GPIO for working with sensors and devices connected to the CrowPi.

2.3.2.2. Windows:

Windows is used for running the server part of the system. It hosts the Node.js application that receives sensor data, stores it in the database, and sends alerts to hospital staff through a web dashboard. Windows provides a stable platform for development and testing, and it supports the required tools for building the server-side of the smart hospital system.

2.3.3 Frameworks and Libraries

OpenCV: OpenCV is a library used for computer vision tasks like detecting and recognizing objects. In this project, OpenCV helps the robot "see" by processing images from its camera. It’s also useful for tracking objects or obstacles.
NumPy: NumPy is a Python library for numerical calculations. It works well with OpenCV for handling and processing image data, making tasks like filtering or transforming images easier.
Pillow: Pillow is a library used for basic image processing, like resizing, rotating, or adjusting colours in images. It is useful for preparing image data before further analysis.
Matplotlib: Matplotlib is a library for creating graphs and charts. It can be used to visualize the data collected by sensors, such as changes in a patient’s vital signs over time.
HiwonderSDK: HiwonderSDK provides tools to control the MasterPi robot's hardware, such as its servos, motors, and sensors. This library simplifies programming the robot’s movement and actions.[11]

Gpiozero: The gpiozero library is used to work with the GPIO pins on Raspberry Pi devices. It makes it easy to connect and control sensors, buttons, and other hardware components in the smart room.[10]
Picamera: Picamera is a library for controlling the Raspberry Pi Camera Module. It enables capturing images and videos programmatically, which can be used for vision tasks in the smart room.[9]
Pigpio: Pigpio is an advanced library for working with GPIO pins. It supports features like control of motors and sensors.[12]2.3.4 Hardware Platforms:

2.3.4.1. CrowPi (Raspberry Pi):
The CrowPi is an educational kit that integrates a Raspberry Pi computer with various sensors and modules, all housed in a compact laptop-like case. It's designed to facilitate learning and experimentation with electronics and programming.

main Features:

- Integrated Display: Features a built-in screen for direct interaction and monitoring.
- Multiple Sensors: Includes sensors for temperature, humidity, light, and more, allowing for comprehensive environmental monitoring.
- GPIO Access: Provides access to the Raspberry Pi's General-Purpose Input/Output GPIO have pins for connecting additional hardware components.
- Supports multiple programming languages, including Python.

Role in the Project: The CrowPi serves as the central hub for the smart room, running software that manages and monitors various sensors. It collects data from various sensors, processes this data, and communicates with the MasterPi robot.

2.3.4.2. MasterPi Hiwonder AI Vision Robot Arm:
The MasterPi is an AI-powered robot arm mounted on a Mecanum wheel chassis, enabling omnidirectional movement. Equipped with a high-definition camera and powered by a Raspberry Pi, It supports Python programming, making it suitable for various AI and vision-based tasks.
Main Features:

- 5 Degrees of Freedom (DOF) Arm allows for flexible movement.
- HD Camera: Provides real-time video feed for object detection and tracking.
- Mecanum Wheels: Enable 360° movement, allowing the robot to navigate complex environments.
- It supports Python programming.
- More capabilities, such as utilizing OpenCV for image processing, enable functions like colour recognition, line following, and target tracking.

2.3.5. Communication Protocols:

The system uses a Wi-Fi client-server communication model, where the Server is the central unit that manages data, communication, and alerts. The smart room (CrowPi) collects room and patient data through sensors (such as room sound, room light, and temperature) and sends it to the server. The mobile robot (MasterPi) also sends close-up sensor readings when it reaches the patient.
The server processes this data, sends alerts if any anomalies are found, and saves the information in a database. It also updates a web application in real time, allowing hospital staff to monitor the situation from one place. This setup supports centralized, real-time monitoring, as shown in Figure 2.1.

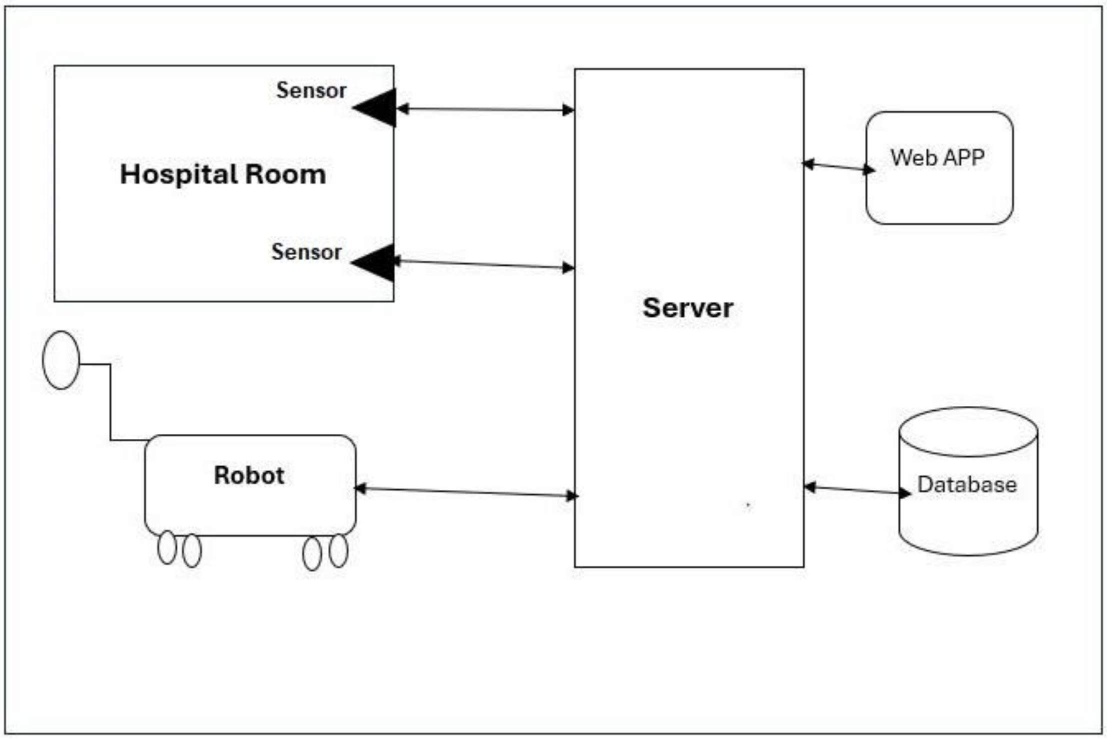


Figure 2.1: Communication System Architecture

Figure 2.1 shows how different parts of the system work together. The smart hospital room and the mobile robot each collect sensor data. The hospital room checks the environment and patient health conditions using fixed sensors, while the robot moves closer to the patient for more detailed readings. Both send data to the central server.

The server, running on a Windows machine, processes this data. It checks for abnormal readings using filtering and anomaly detection methods. If any problems are found, the server saves the data in a PostgreSQL database and sends alerts to the web app, where hospital staff can see the information in real time.

This setup ensures fast communication between the room, robot, server, and staff, helping monitor patients better and faster while reducing the chance of missing important health signs.

2.3.6. Sensors:

Temperature Sensors:
Two temperature sensors for different purposes. The first sensor is placed in the hospital room to measure the room temperature. It helps track environmental conditions, making sure the room stays comfortable and safe for the patient.
The second sensor is used to measure the body temperature of the patient. This sensor is attached to the robot and collects real-time health data when the robot gets close to the patient. Both sensors are connected to the Raspberry Pi, allowing continuous monitoring. If the temperature goes too high or too low, the system will automatically send an alert to hospital staff.
Pulse Oximeter (SpO_2) Sensor:
This sensor measures two important things: the oxygen level in the blood (SpO_2) and the heart rate. It is useful for checking if a patient has low oxygen levels or breathing problems. This information helps hospital staff respond quickly to serious health issues like shortness of breath or low blood oxygen, which may require immediate care. The sensor is connected to the robot, which collects this data when it is near the patient.

- Camera:
  The camera is used for vision-based tasks like detecting the patient's face color or monitoring their breathing. It can also assist in object detection and navigation for the robot.

2.3.7. Integration and Alerts:

2.3.7.1. Algorithms:

This project successfully focuses on comparing two powerful algorithms, Kalman Filter and Isolation Forest, for detecting abnormal sounds in a hospital environment. These methods were tested using real sensor data to improve how the system filters out background hospital sounds and identifies health-related noise. This comparison helped improve the accuracy of patient monitoring and reduce false alerts, which supports faster and better decisions by hospital staff.

2.3.7.2. Navigation and Movement:

Options:

- A*: A* is a popular algorithm for pathfinding because it finds the shortest path between the start and goal efficiently. It uses a heuristic to guide the search, making it suitable for hospital rooms.[13]
- Dynamic Window Approach (DWA): DWA is great for real-time navigation because it calculates the best speed for the robot while avoiding obstacles. This makes it useful for environments with moving objects or tight spaces.[14]
- RRT (Rapidly exploring Random Tree): RRT is ideal for finding paths in complex spaces. It works by sampling random points and connecting them to create a path. It's especially helpful in areas with a lot of obstacles, but it may not always find the shortest path.[15]

Better Option: A* could be a better option because it guarantees finding the shortest and most optimal path, which is important for a hospital room.
But, in dynamic environments, DWA works better where objects might move.

2.3.7.3. Vision and Object Detection:

- YOLO (You Only Look Once): YOLO is a real-time object detection algorithm that processes an entire image in a single pass, detecting multiple objects with bounding boxes. It's suitable for applications requiring quick detection.[16], [17]

- Mask R-CNN: Mask R-CNN is a deep learning algorithm that detects objects and segments them by creating pixel-wise masks. It's beneficial for tasks needing detailed segmentation, such as medical imaging.[18]

- SSD (Single Shot MultiBox Detector): SD detects objects in images using a single deep neural network, balancing speed and accuracy.
  SSD is faster than Mask R-CNN and simpler to implement, but it may not be as accurate as YOLO for small object detection.[19]

Better Option: YOLO could be the better choice because it’s faster than the other two and still accurate enough for my project.[20]

2.3.7.4. Data Collection:

- **Kalman Filter:** The Kalman Filter is a mathematical method used to estimate the true state of a system from noisy measurements. It is widely used in robotics and healthcare applications for tracking sensor data over time.
  How It Works: The Kalman Filter uses the previous data to predict the current value. Then, it compares this prediction with the latest measurement, adjusting the estimate based on how accurate each part is. This process runs continuously, producing smooth and reliable data.
- This method is useful for filtering temperature, heart rate, or sound data, as it reduces noise while preserving accurate trends.[21], [22], [23]
- **Isolation Forest:** The Isolation Forest is a machine learning method used to detect anomalies (also called outliers) in a dataset. It is fast, efficient, and works well with large data streams, making it ideal for healthcare applications.
- How It Works: Randomly selects a feature, randomly picks a value within that feature to split the data. repeats this process, building a tree-like structure to separate each point, and points that are isolated faster (with fewer splits) are likely anomalies.[24], [25]
- **Moving Average Filter:** A Moving Average Filter smooths data by averaging it over a specific time window.
- The Moving Average Filter is easy to implement on a Raspberry Pi and works well for smoothing slower-changing signals, like temperature.[26]
- **Anomaly Detection Models**: use to identify unusual patterns in the data, such as a sudden rise in temperature or irregular heartbeats by identifying data points that deviate significantly from the normal. These models like Hierarchical Temporal Memory (HTM) can alert when anomalies occur, which is useful in applications like monitoring sensor data.[27].
- In addition to technical research, this project also considered safety and healthcare standards to improve trust and usability.

2.4. Additional Research:

2.4.1 ISO Standards and Their Application to the Project:

When designing and implementing a medical robot for a hospital environment, it is important to follow international standards to ensure safety, usability, and compliance with healthcare regulations. Below are the relevant standards and their applications in this project:

Category	ISO Standard	Application in Project	References
Safety Standards	ISO 15066	- Robot will include collision detection and safe-stop mechanisms.	[28]
		- Emergency stop button installed for quick halts.	
		- Robot speed limited near humans	
Risk Management	ISO 14971	- Risk assessments identify hazards, evaluate their impact, and establish control measures.	[29]
Communication and Data Security	ISO 27001	- Implement secure login credentials for accessing the robot. - Add monitoring to detect unauthorized access attempts.	[29]
		- Secure data storage complies with healthcare data protection laws, including GDPR.	
Usability and Design	ISO 62366	Web app interface designed to be simple and intuitive for hospital staff and patients.	[30]

Table 2.1: ISO Standards and Their Application to the Project

Following ISO standards makes the system safer for patients and builds trust with hospital staff and management.

2.5. Existing Final Year Projects

2.5.1. MasterPi Intelligent Robot (Final Project by Gaston DUAULT 2024):

The MasterPi Intelligent Robot is built on a Raspberry Pi platform and is equipped with advanced features, including a mecanum chassis, a robotic arm, and a high-definition camera. It supports functionalities such as colour sorting, object tracking, line following, and intelligent transport. Additionally, the robot integrates an RGB glowing ultrasonic sensor for obstacle avoidance and light colour control, making it versatile for various robotics applications.
Limitations: The robot's movement relies on basic scripts that set specific velocities and durations for actions like moving forward and stopping. These scripts lack advanced navigation algorithms, which means the robot may struggle with complex tasks such as obstacle avoidance and dynamic path planning. Also, the project

employs simple colour-based detection methods for tasks like line following and object tracking.
This project improves on the Baggie-Robots system by using more advanced algorithms for navigation and object detection. Instead of basic movement and simple colour-based tracking, this project integrates better obstacle avoidance and object recognition methods, making the robot more reliable in real-world hospital settings.[31]

2.6. Conclusions

While many healthcare systems use robots or IoT sensors, few combine them into one full system. Most existing projects focus on either home care, vision monitoring, or simple mobile robots. They do not support continuous room monitoring, and advanced data filtering at the same time.

This project fills those gaps by combining a smart room and a robot with real-time communication, navigation, and filtering techniques. It also compares Kalman Filter and Isolation Forest to reduce noise and find real health problems more reliably. The research into programming tools, hardware, and ISO safety standards ensures the project is safe, practical, and useful in hospital settings. By improving both data quality and response time, this project aims to be more effective than existing systems and supports hospital staff in real-time patient monitoring.

3. System Design

3.1. Introduction

This section describes how the smart room and mobile robot are designed to work together as one system. It explains the roles of software, hardware, and communication to ensure smooth interaction between both parts. The design focuses on creating a reliable and flexible system to monitor patient vitals, navigate safely, and send alerts in real time.

3.2 Requirements Gathering:

Before designing the system, it was important to understand what the system should do and how it should work. The features below were identified from aspects of the existing systems considered in the literature. In particular, the use of sound was considered as it is not included in many of the systems reviewed. The requirements were divided into functional and non-functional requirements. These helped guide the design and development of the smart hospital system.

3.2.1 Functional Requirements:

- The smart room should collect sensor data such as temperature, light, and sound.
- The robot should move to the patient when an alert is received.
- The robot should collect patient data such as body temperature, heart rate, and oxygen level.
- The server should store all readings in the database.
- The system should send alerts when abnormal values are detected.
- The web dashboard should display live data and alerts to hospital staff.

3.2.2 Non-Functional Requirements:

- The system should send and receive data in real time.
- Communication between components should be fast and reliable.
- The interface should be simple and easy to use.

3.3. Software Methodology: Incremental Development

This project uses the Incremental Development methodology, which is a good choice for building systems that include both hardware and software. It helps manage complexity by dividing the project into smaller, more manageable steps called increments. Each increment focuses on a single function and is completed and tested before moving on to the next.

The project will be divided into smaller steps (increments). Each increment focuses on a specific function, like setting up sensors in the smart room, programming the robot’s navigation, or developing the communication system. After finishing each increment, it will be tested to ensure it works correctly before moving to the next part.

3.3.1. Overview of System:

The system was designed using the Incremental Development methodology. Each module was built, tested, and integrated step by step. The main modules are:

Data Collection: Sensors on the CrowPi collect room temperature, light, and sound data. Robot sensors collect patient data such as temperature, heart rate, and oxygen level. All data is sent to the server and saved in the PostgreSQL database.

Anomaly Detection: The system uses two methods (Kalman Filter and Isolation Forest) to detect unusual values in both room and patient data. These methods help reduce noise and find health problems early.

Robot Movement: The robot receives commands from the server, moves to the patient's location, collects sound and other readings, and returns.

Communication: MQTT protocol is used for sending commands and sensor data between the smart room, robot, and server.

Web Dashboard: Hospital staff can see live readings, alerts, and charts on a web interface. This allows quick responses to abnormal health conditions.
As shown in System Architecture Diagram in Figure 2:

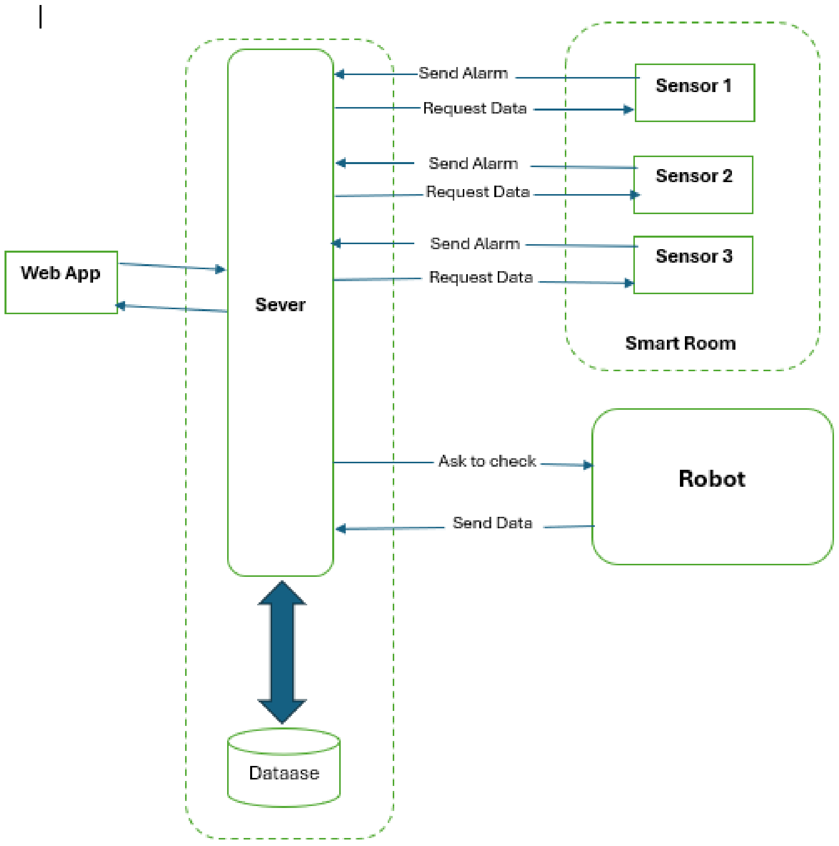


Figure 3.1: System Architecture Diagram

Figure 3.1 show the system integrates the following main components:
The system integrates three main components that work together: the Smart Room, the Mobile Robot, and the Server.

Smart Room (CrowPi): Uses sensors to monitor room conditions such as temperature, light level, and sound. Sends the collected data to the server.

Mobile Robot (MasterPi):

- ➢ Equipped with a robotic arm, vision system, and Mecanum wheels for movement in all directions.
- ➢ Uses navigation algorithms (e.g., A*) to safely move around the room.
- ➢ Performs close-up checks such as measuring patient sound, breathing rates or temperature.

Server: The server is the central hub that manages communication and data management in the system. It connects the smart room, robot, web app, and PostgreSQL database.
The server is responsible for:

- Data Handling: Receiving data from the smart room sensors and storing it in the PostgreSQL database.
- Data Filtering and Anomaly Detection: Applies filtering methods such as the Kalman Filter and Isolation Forest to reduce noise in the sensor data. It also detects unusual or abnormal readings and decides when an alert should be triggered.
- Command Execution: Sending commands to the robot for navigation, vision tasks, and patient monitoring based on alerts from the smart room.
- Real-Time Updates: Sharing real-time updates and alerts with the web app interface for hospital staff to take necessary actions.
- API Implementation: The system uses MQTT (Message Queuing Telemetry Transport) for communication between the smart room, robot, and server.
  Why MQTT?
  (Message Queuing Telemetry Transport) MQTT is a lightweight, publish-subscribe messaging protocol ideal for real-time data transmission in IoT systems. It ensures low-latency updates and efficient data sharing between components.

3.3.2. Communication:

During development, MQTT was chosen instead because it is more suitable for real-time communication in IoT systems. MQTT offers better performance for continuous sensor data transfer and is easier to use for event-based messaging between the robot and smart room.
In this project:
The smart room (CrowPi) and the robot (MasterPi) act as clients.
The server acts as the broker and manages data flow and commands between components.
MQTT is used to:

- Collect and transfer sensor data from the smart room to the server.
- Send commands from the server to the robot (e.g., move to patient, return).
- Send alerts and updates to the web dashboard for hospital staff in real time.

3.3.3. Web App Interface:
Allows hospital staff to view patient data, monitor alerts, and control the robot.
Built using React.js for the frontend and Flask/Django for the backend.

3.3.4. PostgreSQL Database:
Stores patient vitals, robot logs, and alert histories. Ensures data reliability and scalability for future expansion.

3.4 Design Decisions

The table 1 below shows the design options considered for different parts of the system. Each option's strengths and weaknesses are evaluated, and the final choice is explained based on what fits the project best.

Component	Alternative Option	Advantages	Disadvantages	Final Choice & Reason
Navigation	A*	Shortest path, works well in simple spaces	Not good for moving obstacles	Not implemented: Navigation algorithms were planned but not used.
	DWA	Good for dynamic environments	Higher processing needs	Not needed: robot uses simple movement.
	RRT	Works in complex spaces	May not find best path	Not used: complex planning not required.
Vision	YOLO	Fast and works in real-time	Less detailed for small objects	Not implemented: Vision was planned but not used.
	Mask R-CNN	Very accurate with detailed segmentation	Slow and heavy processing	Not used: Too complex for this project.
	SSD	Fast and simple	Less accurate than YOLO	Not used: No vision algorithm used yet.
Data Filtering	Kalman Filter with Anomaly Detection	Smooths noisy data in real-time; useful for detecting unexpected changes	Slightly harder to set up	Picked: Used to reduce noise in sensor data and detect anomalies in live readings.

	Isolation Forest	Detects outliers automatically; works well with large data streams	Needs training and tuning	Picked: Used to compare with Kalman Filter and detect anomalies in sound and vitals.
	Moving Average Filter	Very easy to implement; works for slow signals	Not accurate for fast or complex changes	Not used: Too simple for detecting real-time anomalies in patient monitoring.
Communication	REST API	Simple to understand and set up	Slower for real-time messaging	Planned: Originally chosen but replaced later.
	MQTT	Fast, lightweight, great for IoT devices	Needs setup and testing	Picked: Used for real-time communication between components.

Table 3.1: Design options

3.4.1. Why These Two Filtering Methods:

Kalman Filter and Isolation Forest were chosen because they represent two different ways to detect abnormal data.

Kalman Filter is a mathematical method that removes noise and gives smooth, accurate readings over time. It is useful for tracking changes in sensor values like temperature or heart rate.

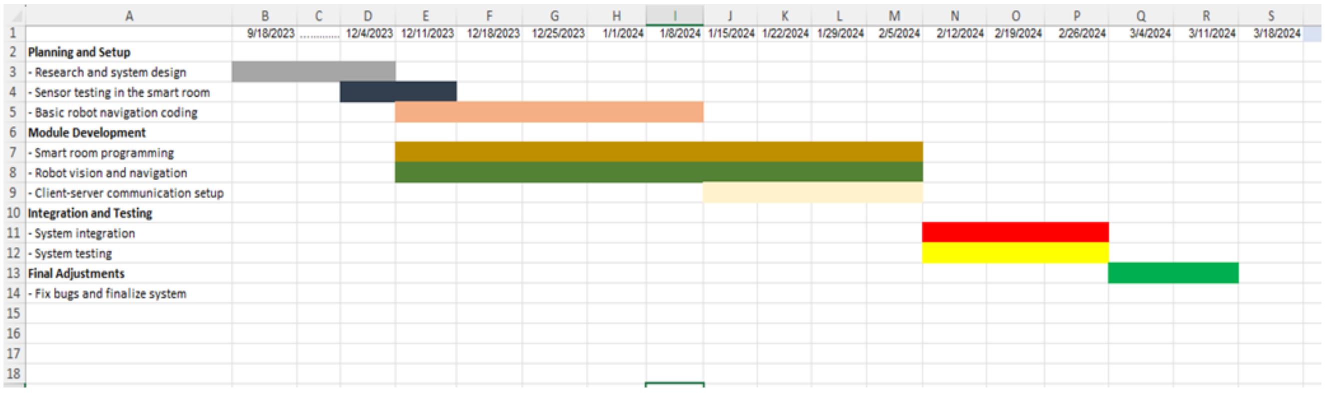
Isolation Forest is an AI-based method that automatically finds unusual patterns or outliers in large sets of data, such as sudden spikes in sound.

Using both methods allows the system to compare how each one works in real-time monitoring. This helps improve the accuracy of detecting problems and reduces false alerts.

3.5. GANTT Chart

Figure 3.2: GANTT Chart

This chart shows the main development stages of the project. Each stage includes one or more steps (increments).



3.6. Database Design:

3.6.1. PostgreSQL database

The system uses a PostgreSQL database to store all sensor readings, patient data, robot logs, and alerts. The database design supports both real-time monitoring and later analysis.

The key tables in the database include:

- smart_room_readings: stores room sensor data (temperature, light, sound)
- robot_readings: stores patient health vitals (temperature, heart rate, oxygen level)
- anomalies: logs abnormal values detected by Kalman Filter or Isolation Forest
- alerts: keeps messages sent to staff, with severity level
- robot_actions: tracks what the robot did (move, scan, etc.) anomaly_comparison: used for testing and comparing both methods
- Other tables include patients, beds, and staff for managing room assignments and access.

3.6.2. Entity-Relationship Diagram (ERD):

The ERD shows how the main entities in the system are connected. It includes tables for storing patient data, smart room readings, robot readings, anomalies, alerts, and actions.

This design allows for:

- Linking sensor readings to specific patients
- Storing detected anomalies from Kalman or Isolation Forest
- Triggering alerts based on abnormal readings
- Tracking robot movements and system actions

The ERD also supports testing and comparison between anomaly detection methods using the anomaly_comparison table.
or full table structure and SQL code, see **Appendix A**.

(See Figure 4 below)

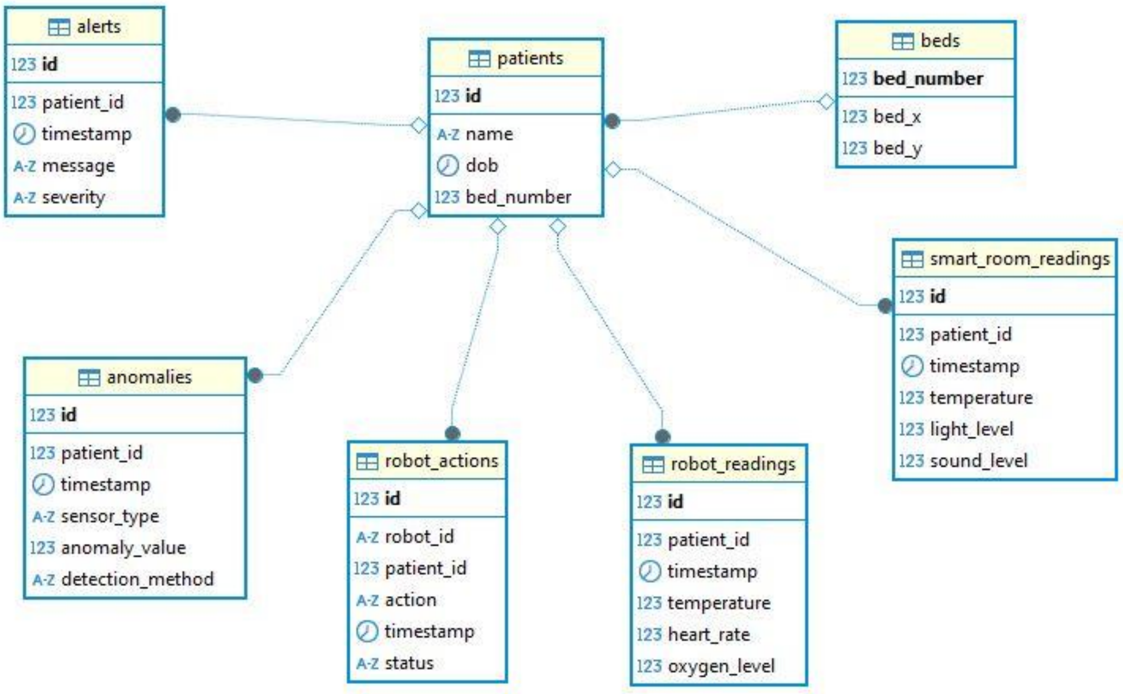


Figure 3.3: Entity-Relationship Diagram of Smart Hospital Monitoring System

3.7. Conclusions

The system is designed to support real-time patient monitoring in a smart hospital room using simple and low-cost components. Each part of the design, from the smart room to the mobile robot and central server, was planned to improve how data is collected, filtered, and shared. The project uses MQTT for fast communication, PostgreSQL for organized data storage, and a web dashboard for easy access by hospital staff. Two filtering methods, Kalman Filter and Isolation Forest, were used to reduce noise and find real health problems quickly.

This helps improve the quality of the sensor data and allows the system to respond faster with fewer false alerts. The design uses an incremental development approach, making each module simple to test and update. This ensures that the system is flexible, works in real-world hospital conditions, and supports the main project goal—detecting vital health changes clearly and early.

4. Software Development

4.1. Introduction:

This part of the project shows how the full system was built to monitor patients in a smart hospital room. The main goal was to filter noise in real-time and detect abnormal sensor readings that might show a health problem. The system uses sensor data from CrowPi and controls a robot that moves near the patient. The sensor data is sent using MQTT and stored in a PostgreSQL database. A Python script checks the data using two methods, Kalman Filter and Isolation Forest, to find abnormal values. The system also includes a web dashboard that shows real-time data and gives alerts when needed. Another web page displays live charts to help staff see how sound features change over time.

4.2. Software Development:

4.2.1 Comparison Between Anomaly Detection Methods

In this section, two different methods compared for filtering noise and detecting anomalies in real-time sensor data:

- Kalman Filter with Anomaly Detection rule
- Isolation Forest (machine learning model)
-

Both methods were tested using real-time data collected from the smart room and robot sensors. However, in this part, the focus is on the sound sensor, because it is more affected by background noise and sudden events (like someone shouting).
The aim was to find out:

- Which method removes normal noise better
- Which one detects real abnormal values more accurately
- Which is more useful for real-time healthcare systems.

In addition to testing the two methods, this section also compares how using a robot can improve the quality of sensor readings. The robot moves closer to the patient, which helps reduce background noise and allows for more accurate sensor data. This comparison helps to evaluate whether robot-assisted data collection produces better results than relying only on sensors placed in the smart room.

4.2.3.1. Code developed:

4.2.3.1.1. A Python script:

Python script (data_filtering.py) was created to run both methods at the same time using live sensor data. Each time a new value is recorded in the database, the script:

- Loads recent data for training (Isolation Forest)
- Runs both methods on the newest reading
- Logs the result and saves any anomaly to the anomalies table,

The full code is available in **Appendix B.1**

4.2.3.1.2. Why dB Was Not Enough:
In the beginning, sound was measured in decibels (dB), using the RMS energy of the signal. However, during testing, it was discovered that dB could not help distinguishing between different types of sounds.

For example, a loud TV and a cough produced similar dB values, even though one is normal background noise and the other could be a sign of a health problem. Because of this, more features were added to better describe each sound.

4.2.3.1.5. Improved Feature Set for Sound:
To improve the detection accuracy, two more sound features were added:

Feature	Description	Why It Helps
RMS (Root Mean Square)	Measures the energy of the signal	Helps estimate sound intensity reliably
ZCR (Zero Crossing Rate)	Counts how often the signal changes direction	Helps detect sharp, short sounds like coughs or claps

Table 4.1: Sound Features

RMS (Root Mean Square): RMS calculates the square root of the average of the squares of the signal's amplitude values. It provides a measure of the signal's power or energy over time. It accounts for both the magnitude and duration of the signal, making it effective in distinguishing between types of sound. [32]

ZCR (Zero Crossing Rate): ZCR measures how frequently the audio signal crosses the zero-amplitude axis, indicating changes in the signal's polarity. ZCR useful in distinguishing between different types of sounds, such as separating speech from background noise.[33]

dB (Decibel): The decibel scale quantifies sound intensity logarithmically, reflecting how humans perceive changes in loudness. The dB scale is a standard unit of measurement in audio engineering.[34]

These three features, dB, RMS, and ZCR, were used together to test each new sound reading.
Now, both Kalman Filter and Isolation Forest can use these values to:

- Better identify sudden, sharp changes in sound (e.g., cough, scream)
- Ignore steady loud background noise (e.g., fan, TV).

4.2.3.1.5 System Updates for System Improvements
To support this work, several parts of the system were updated:

- Python script data_filtering.py updated for 3 features:
  The full code in **Appendix B.2** [35], [36]

- sensors.js updated to store db, rms, and zcr in PostgreSQL:

The full code in **Appendix B.3**

- CrowPi sound sensor Python script updated to send 3 values via MQTT: The full code in **Appendix B.4**

- PostgreSQL schema updated to include new sound feature columns:

```
ALTER TABLE smart_room_readings
   ADD COLUMN db FLOAT,
   ADD COLUMN rms FLOAT,
   ADD COLUMN zcr FLOAT;

-- Optional: remove old column if not needed
ALTER TABLE smart_room_readings DROP COLUMN sound_level;
```

4.2.3.1.6 Data Requirements for Each Method:
Both detection methods use sensor data collected in real time from the smart room. The requirements for each method are different:

Method	Needs Training?	Input Features	Best For
Kalman Filter	No	dB, RMS, ZCR (live values)	Real-time tracking
Isolation Forest	Yes (100+ normal readings)	dB, RMS, ZCR	Detecting rare outliers

Table 4.2: Data Requirements for Each Method

The implementation of both methods, including database access, model setup, and anomaly insertion logic, is already described in previews sections.
The Kalman Filter and Isolation Forest methods were already developed and tested in earlier sections using the light sensor. For this section, the same methods are applied to the sound sensor to compare their performance. No changes to the data requirements were needed, as both methods support real-time readings from any sensor type.

4.2.3.1.7 Data Collection and Simulation:
In this section, I collected and prepared sound data for training the Isolation Forest model. Since live smart room data was limited, I searched online and selected one hospital-related noise dataset. The goal was to simulate different types of normal background sounds like walking, phone ringing, or TV, and convert them into features suitable for machine learning.

To train the Isolation Forest method, real hospital noise data was needed. After searching online, one suitable dataset was selected. This dataset includes common hospital background sounds such as people talking, walking, TV noise, phone ringing, and other environmental sounds.[35], [36] However, the dataset only contained audio recordings in .wav format without any features. To fix this, a Python script was created to extract important sound features using the librosa library.

The script does the following steps:

Loads each .wav file from the dataset.
Calculates three features:

- ZCR (Zero Crossing Rate) – how often the sound signal changes direction.
- RMS (Root Mean Square) – shows how strong the sound is.
- dB (Decibels) – calculated from RMS and shifted by +100 to stay in a positive range.

Skips files that are too quiet or unreadable.
Saves the results in a CSV file called training_noise_data_audio_features.csv.[37], [38]

The Isolation Forest model was trained only on noise samples labelled as "normal". Abnormal sounds such as coughing were kept separate and used for testing to evaluate if the model can correctly detect these as anomalies. This approach helps ensure the model does not learn from abnormal data during training and improves generalisation to new, unexpected events.

4.2.3.1.8 Development Improvements During Testing:
Several changes were made during testing to improve the system and reduce false detections. These updates show real development and tuning effort:

4.2.3.1.8.1. Control When Detection Runs:
At first, the system used a child process in server.js to run data_filtering.py. This script was running multiple times, even when there was no new data.

To fix this, the logic was moved to sensors.js. Now, the system checks if the new MQTT message has a different timestamp. It runs the detection script only when new sensor data arrives:

```
 if (timestamp !== lastProcessedTimestamp) {
     lastProcessedTimestamp = timestamp;

     const scriptPath = path.join(__dirname, '../data_filtering.py');
     const py = spawn('python', ['-u', scriptPath]);
     ...
 }
```

Benefit:
This change stopped repeated checks and ensured that both methods only run when new readings are available.

4.2.3.1.8.2. Improve Kalman Filter Accuracy Using Data Filtering:
Even though the Kalman Filter is not trained like Isolation Forest, its results depend on clean starting values. During testing, the Kalman method was detecting some false alerts.

To improve this, the feature ranges were used to select only clean noise data from the database. This gave the Kalman filter stable values to work with.

Example:
Kalman now uses starting values based on realistic safe ranges:

- db: 58 to 62
- rms: 0.005 to 0.015
- zcr: 0 to 0.02

If the difference (called residual) between the new value and the predicted one is too high, it triggers an alert. Using filtered data helped reduce the number of false alerts and improved Kalman results during live tests.

4.2.3.1.8.3. Make Training Data Match Real Hospital Sounds:

At first, the system marked normal background noise (like people talking, walking, or a TV) as abnormal. This was happening because the training data had only very quiet readings.

To fix this, the training dataset was improved by:

- Adding real examples of safe hospital noise
- Including sounds like talking, walking, and light background activity

Increasing the normal ranges for rms and zcr based on live patient tests.

4.2.3.1.8. Removed db Feature from Detection
During testing, it was found that db values caused confusion and many false alarms.

db often overlapped in noise, normal, and abnormal ranges

Both Isolation Forest and Kalman Filter performed better after removing db
Final model used only rms and zcr for sound analysis.

.
4.2.3.1.9. Created Evaluation Script to Test Models:
A new script (testdata.py) was developed to evaluate the detection accuracy using the same logic as real-time monitoring.

4.2.3.1.10. Use of Controlled Timestamps to Avoid Reprocessing:

To avoid repeating the same readings, the system checks if the current MQTT message has a new timestamp before running the anomaly detection script. This ensures:

- The script only runs when new live data arrives.
- No duplicated analysis happens, which improves system efficiency and output clarity.

Full code of final python script for both methods in **Appendix B.4**

4.2.2. Creating Anomaly Detection methods for Smart Room

4.2.2.1. Overview:

This part of the project focuses on building and testing two anomaly detection methods: Kalman Filter and Isolation Forest. These methods were developed and tested first using light sensor data from the smart room. Light data was selected in the beginning because it is easier to control and test. The goal here was not only to detect abnormal readings, but also to develop the full detection pipeline, including data collection, processing, and filtering.

4.2.2.2. Code developed:

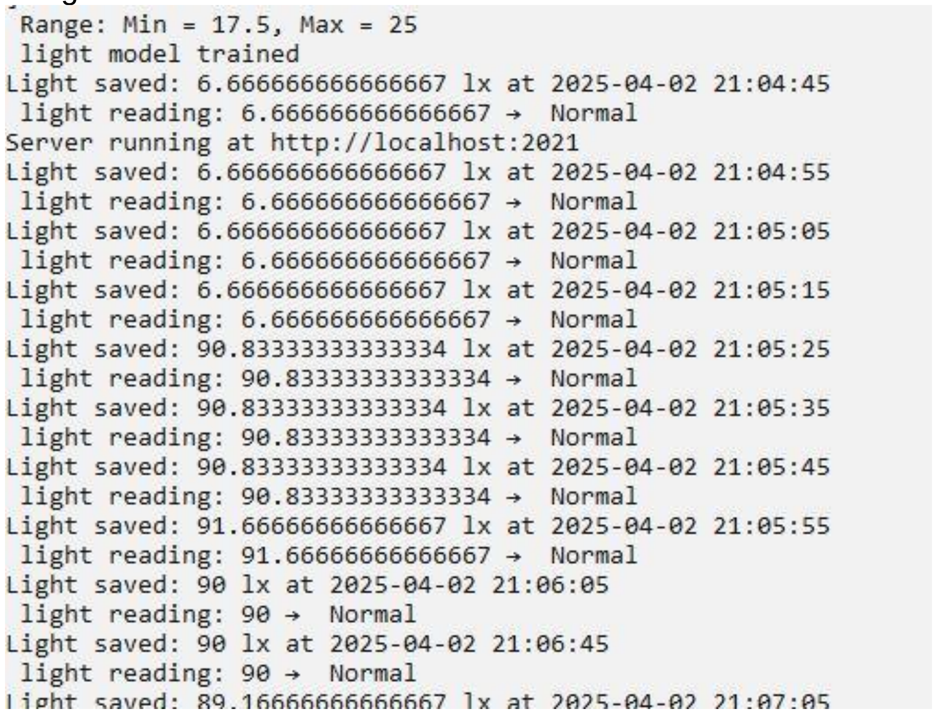
4.2.2.2.1. Isolation Forest:

The first attempt used a JavaScript file (anomaly.js) to apply the Isolation Forest algorithm on light sensor values. However, this version didn't work correctly, it always marked the readings as normal, even when values were too high or too low. anomaly.js full code :

```
const { IsolationForest } = require('isolation-forest');

let models = {}; // Store models for each sensor type

// Train model using normal data from database
async function trainFromDatabase(pool) {
  const sensors = [
    { type: 'light', column: 'light_level', table: 'smart_room_readings', min: 5, max:25}
  ];

  for (const sensor of sensors) {
    try {
      const result = await pool.query(`
        SELECT ${sensor.column} FROM ${sensor.table}
        WHERE ${sensor.column} IS NOT NULL
        AND ${sensor.column} BETWEEN $1 AND $2
        LIMIT 200
      `, [sensor.min, sensor.max]);

      const data = result.rows.map(row => row[sensor.column]);
      console.log(` ${sensor.type} training data:`);
```

```
            console.log(data);
            console.log(` Range: Min = ${Math.min(...data)}, Max = ${Math.max(...data)}`);


            if (data.length > 5) {
                const iso = new IsolationForest({ contamination: 0.1 });
                iso.fit(data.map(x => [x]));
                models[sensor.type] = iso;
                console.log(` ${sensor.type} model trained`);
            } else {
                console.log(` Not enough data to train ${sensor.type}`);
            }

        } catch (err) {
            console.error(` Error training ${sensor.type}:`, err.message);
        }
    }
}

// Check if a new value is anomaly
function isAnomaly(sensorType, value) {
    const model = models[sensorType];

    if (!model) {
        console.log(`No model found for ${sensorType}`);
        return false;
    }

    const prediction = model.predict([[value]]);
    const isOutlier = prediction[0] === -1;

    // Show helpful log
    console.log(` ${sensorType} reading: ${value} → ${isOutlier ? ' Anomaly' : ' Normal'}`);

    return isOutlier;
}



// Export both functions
module.exports = {
    trainFromDatabase,
    isAnomaly
};
```

as shown in figure 4. Below



```
 Range: Min = 17.5, Max = 25
 light model trained
Light saved: 6.66666666666667 lx at 2025-04-02 21:04:45
 light reading: 6.66666666666667 →  Normal
Server running at http://localhost:2021
Light saved: 6.66666666666667 lx at 2025-04-02 21:04:55
 light reading: 6.66666666666667 →  Normal
Light saved: 6.66666666666667 lx at 2025-04-02 21:05:05
 light reading: 6.66666666666667 →  Normal
Light saved: 6.66666666666667 lx at 2025-04-02 21:05:15
 light reading: 6.66666666666667 →  Normal
Light saved: 90.83333333333334 lx at 2025-04-02 21:05:25
 light reading: 90.83333333333334 →  Normal
Light saved: 90.83333333333334 lx at 2025-04-02 21:05:35
 light reading: 90.83333333333334 →  Normal
Light saved: 90.83333333333334 lx at 2025-04-02 21:05:45
 light reading: 90.83333333333334 →  Normal
Light saved: 91.6666666666667 lx at 2025-04-02 21:05:55
 light reading: 91.6666666666667 →  Normal
Light saved: 90 lx at 2025-04-02 21:06:05
 light reading: 90 →  Normal
Light saved: 90 lx at 2025-04-02 21:06:45
 light reading: 90 →  Normal
Light saved: 89.1666666666667 lx at 2025-04-02 21:07:05
```

Figure 4. : IsolationForest method not working with anomaly.js

To fix this, a Python version of Isolation Forest was implemented inside the data_filtering.py file. This version worked successfully.
Steps in the Isolation Forest method:

- Connect to the PostgreSQL database.
- Load the last 100 normal light values (e.g. between 10–25 lx).
- Train the model using these values.
- Get the latest sensor reading and check if it's very different.
- If yes, mark it as an anomaly and save it in the anomalies table.

4.2.2.2.2. Kalman Filter with Anomaly Detection:
In the same Python script (which was later renamed to data_filtering.py), a Kalman Filter class was created to work alongside the Isolation Forest model. This filter is a math-based method that predicts sensor values based on previous data and helps track how the readings change over time. It is useful for identifying unusual spikes by comparing expected vs actual values in real time.

How the Kalman Filter works:
- A class called KalmanFilter created.
- It uses math equations to predict the next value based on previous readings.
- It compares the predicted value to the actual reading from the sensor.
- The difference between them is called the residual.
- If the residual is too large, that means the sensor value is not expected → maybe an anomaly.

- If the residual is small, it's considered normal noise.

The residual threshold set to 3 at first, this means if the value is more than 3 units away from what the Kalman filter expects, it will be marked as an anomaly. Sometimes, small changes happen in the room that are not problems, for example someone turns on their phone screen, the TV light reflects on the sensor, light from another room enters. Therefore, a setting called noise_tolerance was added in the sensor config to help reduce false alarms. The noise_tolerance setting was used to. say:
"If the difference is small, it's OK. Don't show it as an anomaly."
This makes the system smarter and helps it know the difference between noise and real problems, like if a patient suddenly turns the main light on (which may mean they need help).

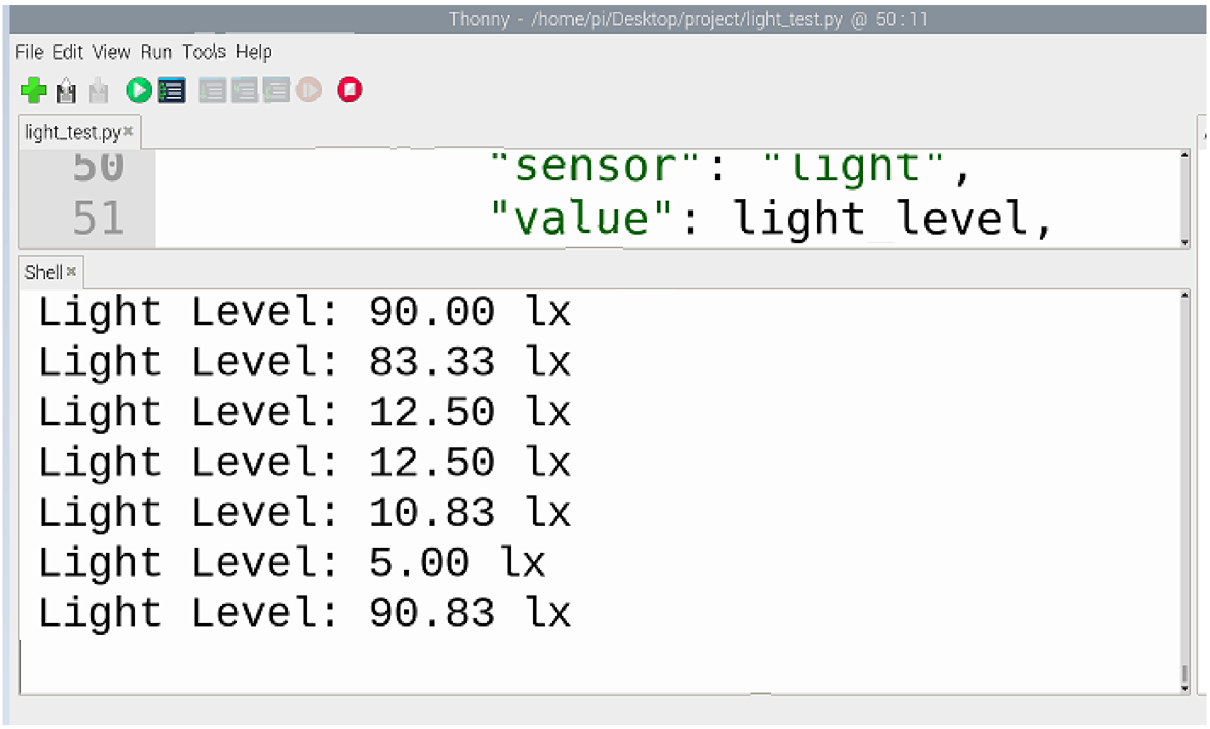
Full code of both method in **Appendix B.6**

4.2.2.3. How it was tested:

4.2.2.3.1. Isolation Forest:

To test the Isolation Forest method for detecting anomalies in light sensor data.

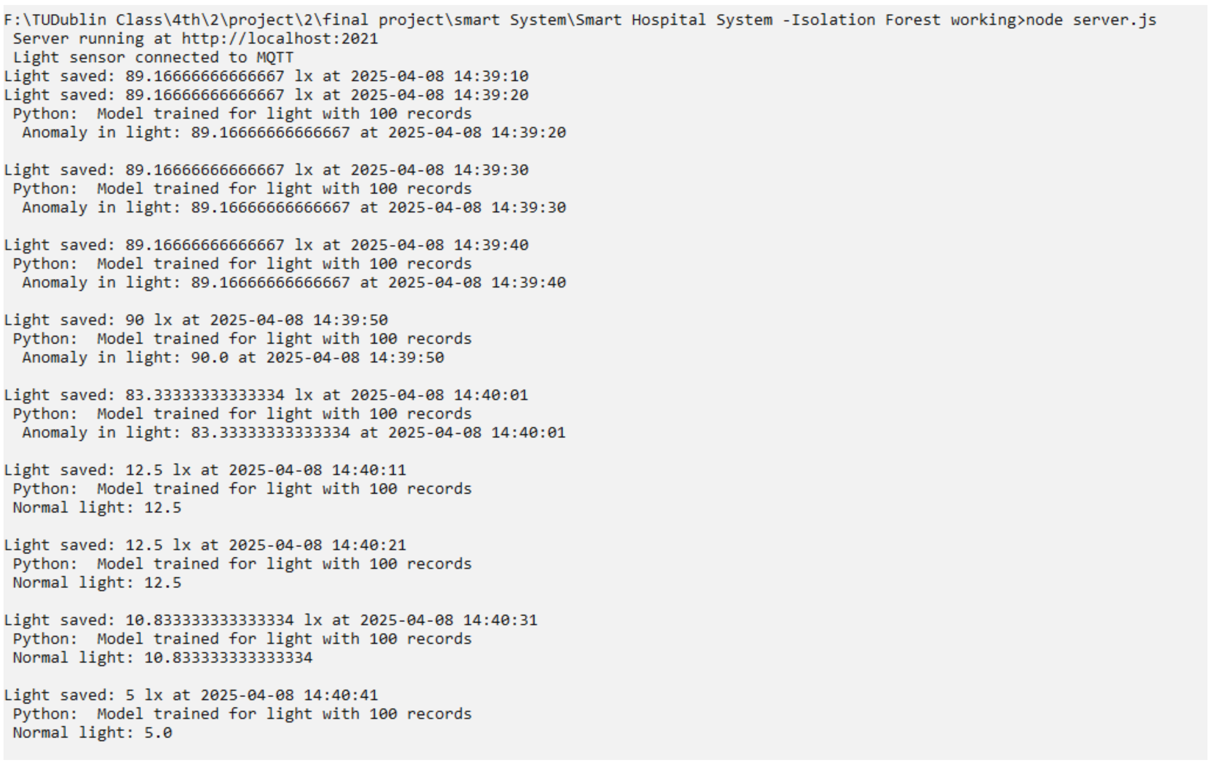
Smart room read and send Light reading to server as shown in image below


Thonny - /home/pi/Desktop/project/light_test.py @ 50 : 11
File Edit View Run Tools Help
light_test.py
Shell

```
                    "sensor": "light",
                    "value": light_level,
```

```
Light Level: 90.00 lx
Light Level: 83.33 lx
Light Level: 12.50 lx
Light Level: 12.50 lx
Light Level: 10.83 lx
Light Level: 5.00 lx
Light Level: 90.83 lx
```

Server: Receives light value, saves it in smart_room_readings, and calls the isolation_forest.py script to check if the value is normal or an anomaly:
Each time a new reading is received, the Python script loads the last 100 records from the database to train the Isolation Forest model.



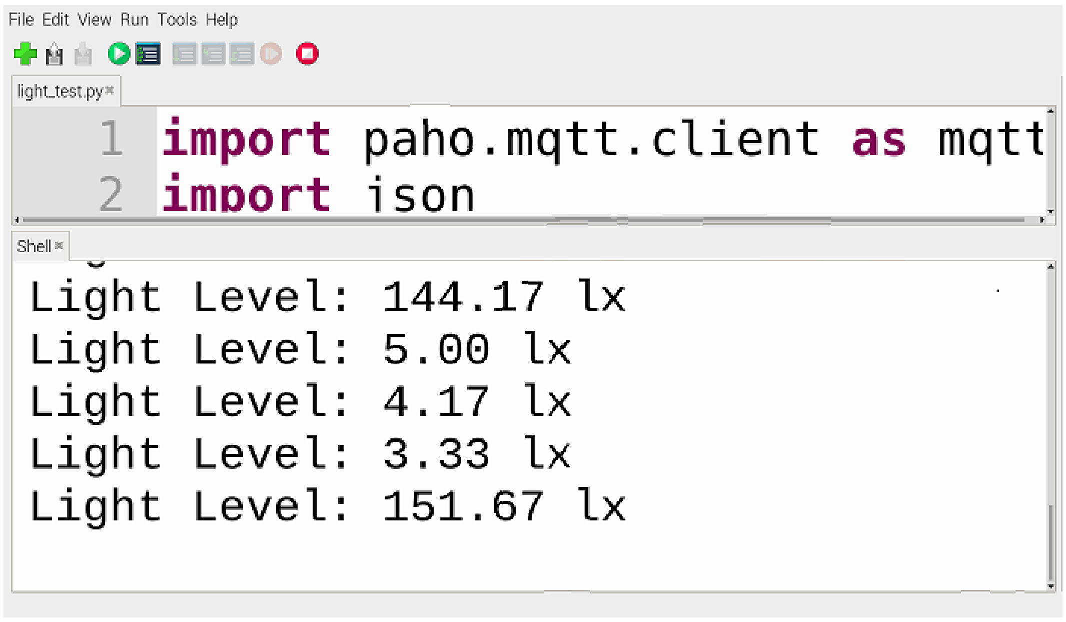
```
F:\TUDublin Class\4th\2\project\2\final project\smart System\Smart Hospital System -Isolation Forest working>node server.js
 Server running at http://localhost:2021
 Light sensor connected to MQTT
Light saved: 89.16666666666667 lx at 2025-04-08 14:39:10
Light saved: 89.16666666666667 lx at 2025-04-08 14:39:20
 Python:  Model trained for light with 100 records
  Anomaly in light: 89.16666666666667 at 2025-04-08 14:39:20

Light saved: 89.16666666666667 lx at 2025-04-08 14:39:30
 Python:  Model trained for light with 100 records
  Anomaly in light: 89.16666666666667 at 2025-04-08 14:39:30

Light saved: 89.16666666666667 lx at 2025-04-08 14:39:40
 Python:  Model trained for light with 100 records
  Anomaly in light: 89.16666666666667 at 2025-04-08 14:39:40

Light saved: 90 lx at 2025-04-08 14:39:50
 Python:  Model trained for light with 100 records
  Anomaly in light: 90.0 at 2025-04-08 14:39:50

Light saved: 83.33333333333334 lx at 2025-04-08 14:40:01
 Python:  Model trained for light with 100 records
  Anomaly in light: 83.33333333333334 at 2025-04-08 14:40:01

Light saved: 12.5 lx at 2025-04-08 14:40:11
 Python:  Model trained for light with 100 records
 Normal light: 12.5

Light saved: 12.5 lx at 2025-04-08 14:40:21
 Python:  Model trained for light with 100 records
 Normal light: 12.5

Light saved: 10.833333333333334 lx at 2025-04-08 14:40:31
 Python:  Model trained for light with 100 records
 Normal light: 10.833333333333334

Light saved: 5 lx at 2025-04-08 14:40:41
 Python:  Model trained for light with 100 records
 Normal light: 5.0
```

Results**:** (night time : under 20 lx, light turn on : above 40 lx)
- When the light level was high (e.g., 90 lx, 89 lx, 83 lx): The system detected them as anomalies.

- When the light level dropped to more normal indoor values, the system classified them as normal.

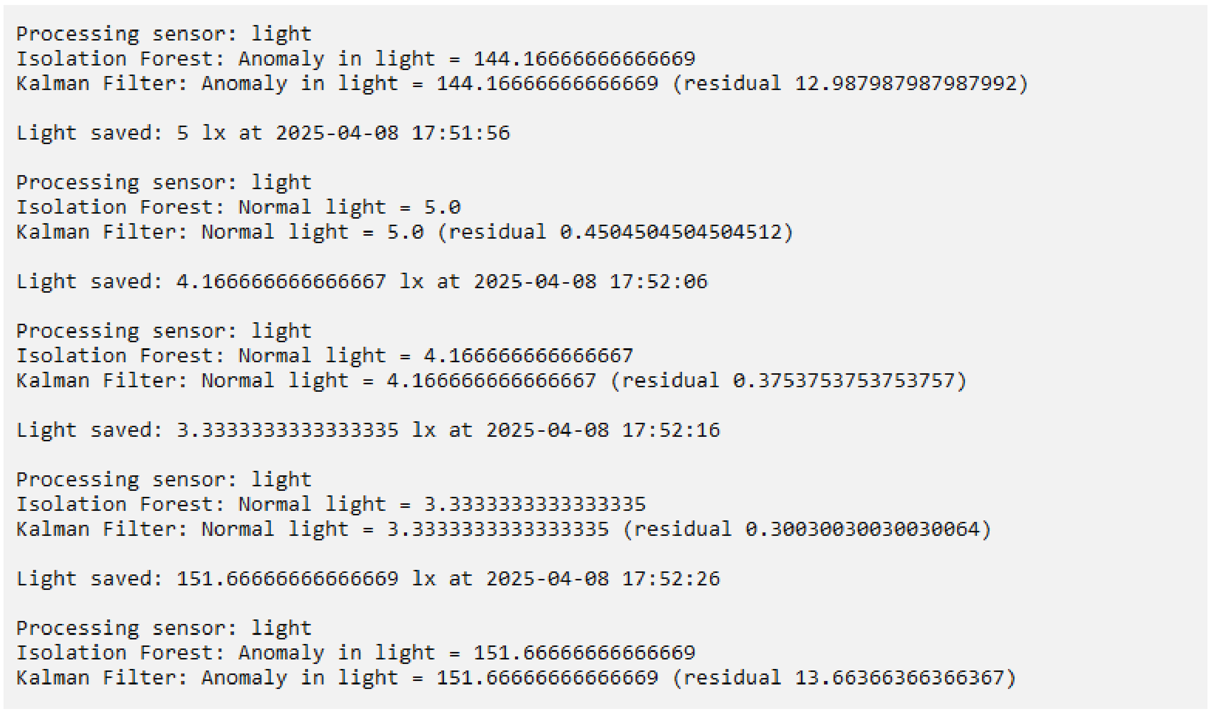
4.2.2.3.2. Kalman Filter with Anomaly Detection:

After testing Isolation Forest, The Kalman Filter was tested separately to see how well it could detect abnormal light sensor readings. The goal was to see if it could spot when the light level suddenly changes and decide if the change is just noise or a possible real anomaly.



```
import paho.mqtt.client as mqtt
import json
```

```
Light Level: 144.17 lx
Light Level: 5.00 lx
Light Level: 4.17 lx
Light Level: 3.33 lx
Light Level: 151.67 lx
```


```
Processing sensor: light
Isolation Forest: Anomaly in light = 144.16666666666669
Kalman Filter: Anomaly in light = 144.16666666666669 (residual 12.987987987987992)

Light saved: 5 lx at 2025-04-08 17:51:56

Processing sensor: light
Isolation Forest: Normal light = 5.0
Kalman Filter: Normal light = 5.0 (residual 0.4504504504504512)

Light saved: 4.166666666666667 lx at 2025-04-08 17:52:06

Processing sensor: light
Isolation Forest: Normal light = 4.166666666666667
Kalman Filter: Normal light = 4.166666666666667 (residual 0.3753753753753757)

Light saved: 3.3333333333333335 lx at 2025-04-08 17:52:16

Processing sensor: light
Isolation Forest: Normal light = 3.3333333333333335
Kalman Filter: Normal light = 3.3333333333333335 (residual 0.30030030030030064)

Light saved: 151.66666666666669 lx at 2025-04-08 17:52:26

Processing sensor: light
Isolation Forest: Anomaly in light = 151.66666666666669
Kalman Filter: Anomaly in light = 151.66666666666669 (residual 13.66366366366367)
```

The Kalman Filter was set with a **residual threshold of 3** to decide if a reading is abnormal.

The Kalman Filter correctly predicted and detected large sudden changes as anomalies.

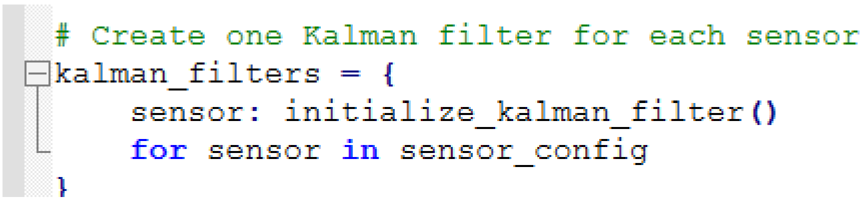
It used residual values to compare predictions vs. real readings:
- Small residual: Normal (likely noise)
- Large residual: Anomaly (unexpected change)

4.2.2.4. Improvements made after testing:

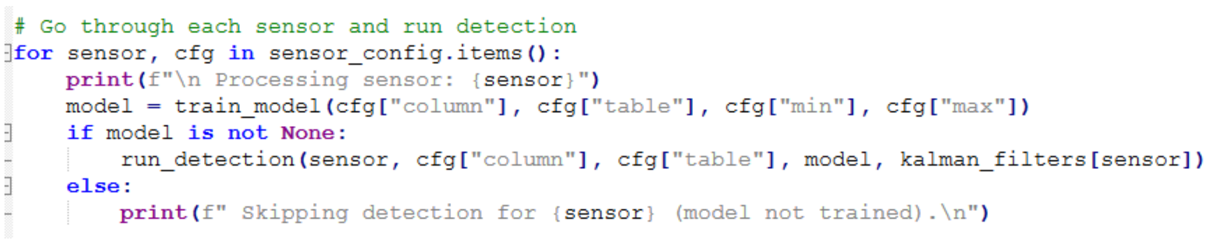
During testing, it was observed that the Kalman Filter gave correct results, but it re-calculated everything from the beginning every time a new reading came in. This meant:
- It didn't "learn" from past values.
- It acted more like a basic filter, not a predictive tracker.

Replace with: "While testing different light values from the smart room, it was observed that :
Even when sending many readings close together (like 5.0 lx, 4.1 lx, 3.3 lx...), the Kalman filter treated each one as if it were the first reading.
It recalculated the prediction and error from scratch every time.

The Kalman Filter was updated to reuse the same instance per sensor and reused it in each call. This way, the filter keeps learning from past values and doesn't reset on each reading:

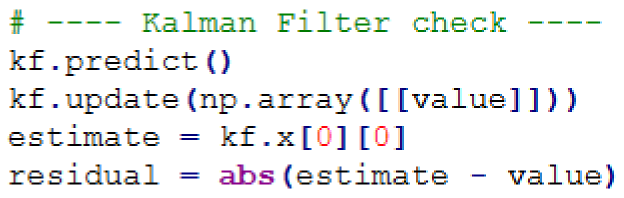

```
# Create one Kalman filter for each sensor
kalman_filters = {
    sensor: initialize_kalman_filter()
    for sensor in sensor_config
}
```

This creates a Kalman filter object per sensor, it is only initialized once, not every time.
The Kalman filter is passed into the function



```
# Go through each sensor and run detection
for sensor, cfg in sensor_config.items():
    print(f"\n Processing sensor: {sensor}")
    model = train_model(cfg["column"], cfg["table"], cfg["min"], cfg["max"])
    if model is not None:
        run_detection(sensor, cfg["column"], cfg["table"], model, kalman_filters[sensor])
    else:
        print(f" Skipping detection for {sensor} (model not trained).\n")
```

Then predict and update the filter over time inside the function



```
# ---- Kalman Filter check ----
kf.predict()
kf.update(np.array([[value]]))
estimate = kf.x[0][0]
residual = abs(estimate - value)
```

Kf is Kalman filter instance, it is now persistent, the predictions get better as more values come in.

4.2.2.5. Challenges faced:

- JavaScript version failed to detect anomalies, so it was replaced with Python.
- Kalman Filter had to be updated to maintain state across readings.
- High and low light changes were simulated manually to test the system properly.

4.2.2.6. Limitations or what was not finished:

- Only light sensor was used at this stage to develop and confirm the methods.
- Other sections include testing with sound sensor

4.2.3 Smart Room Data Collection

4.2.3.1. Overview

While developing the anomaly detection system, the data collection setup was created at the same time. The light sensor was used during testing because it is simple and helps confirm that the full data pipeline—from smart room to server to database—was working correctly. The system was designed to collect sensor data using MQTT and store it in a PostgreSQL database for later processing.

4.2.3.2. What was developed:

4.2.3.2.1. Server Setup

A file called server.js was created to handle incoming data. It performs these tasks:

Listens for sensor data (e.g. light) from the smart room using MQTT.

Saves the data to the smart_room_readings table in the PostgreSQL database.

Displays the latest reading on the web interface.

To connect safely to the database, a .env file was used. This file stores important settings like the database username, password, and IP address.

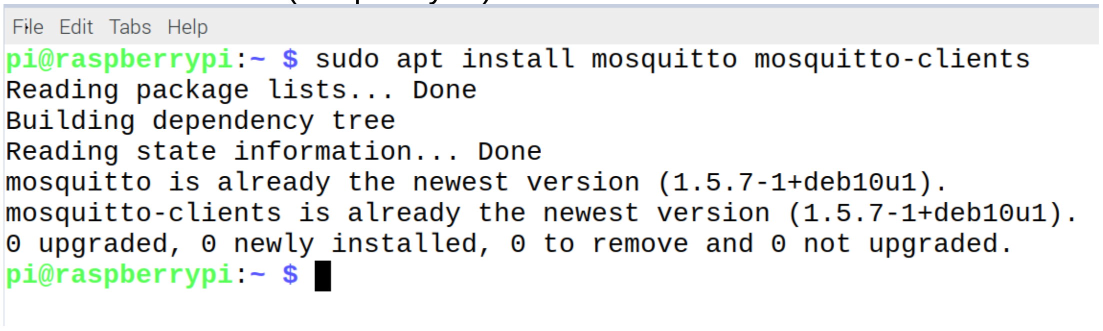
Why .env file is used:

- Keeps passwords and private info out of the main code.
- Makes it easier to move or share the project to another device.

4.2.3.2.2. MQTT Communication Setup

To send data from the smart room to the server, MQTT was used as the communication method.
Mosquitto MQTT broker was installed on both:

- The smart room (Raspberry Pi)



```
pi@raspberrypi:~ $ sudo apt install mosquitto mosquitto-clients
Reading package lists... Done
Building dependency tree
Reading state information... Done
mosquitto is already the newest version (1.5.7-1+deb10u1).
mosquitto-clients is already the newest version (1.5.7-1+deb10u1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
pi@raspberrypi:~ $ 
```

- The Windows laptop (server)

MQTT Configuration:

- The Windows server (IP: 192.168.1.13) acts as the MQTT broker.
- The Raspberry Pi acts as the client, publishing sensor values to this broker.

So when the smart room sends a message to smartroom/light, the server receives that message and saves it into the PostgreSQL database.

4.2.3.2.2. Sensors Management:

After that, a new file named sensors.js was created to manage all the smart room sensors in one place. This makes it easier to read sensor data and send it to the server for processing and anomaly detection.

4.2.3.2.3. Sound Sensor Setup:

A Python script was also developed in the smart room to collect both light and sound data. The light values were taken from the light sensor, and the sound levels were measured using a microphone connected to the CrowPi (not the built-in one). These readings were sent to the server and stored in the database. This data was later used to compare the two filtering methods for detecting anomalies.

After the Python script in the smart room was tested, an issue appeared with the sound values — many of them were negative. This happened because the sound was recorded as a small number (RMS), and then converted to decibels using the formula:

db = 20 * log10(rms + 1e-6).

4.2.3.2.3. 1.Fixing Sound Data

When the sound was quiet, the RMS value was close to zero, so the log result became a negative number.
To solve this, the code was updated by adding +90 dB to shift the values into a more realistic range. For example, a quiet room that showed -60 dB now shows 30 dB, and a loud sound like shouting shows above 85 dB, which matches typical real-world sound levels.

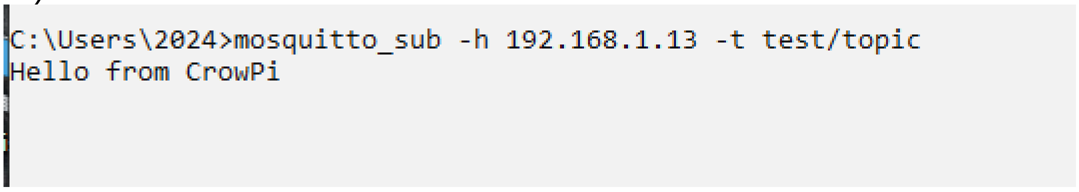
This adjustment helped make the sound values easier to understand and better for testing the anomaly detection methods.

For testing sound-based anomaly detection, I used the CrowPi sound sensor to simulate both smart room readings and close-up patient readings, because I had no sound sensor on the robot.

4.2.3.3. How it was tested:

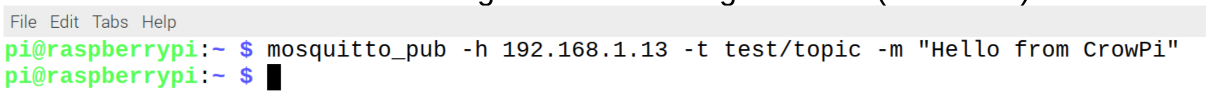
4.2.3.3.1. MQTT Connection:

Raspberry Pi sent test data using Python and MQTT.
Server received it using server.js.
Console printed “Hello from CrowPi” → confirms success.
Test server.js: Server wait for message from smart room using broker IP (Server IP):



```
C:\Users\2024>mosquitto_sub -h 192.168.1.13 -t test/topic
Hello from CrowPi
```

Test smart room : it sends message to Server using broker IP(Server IP):

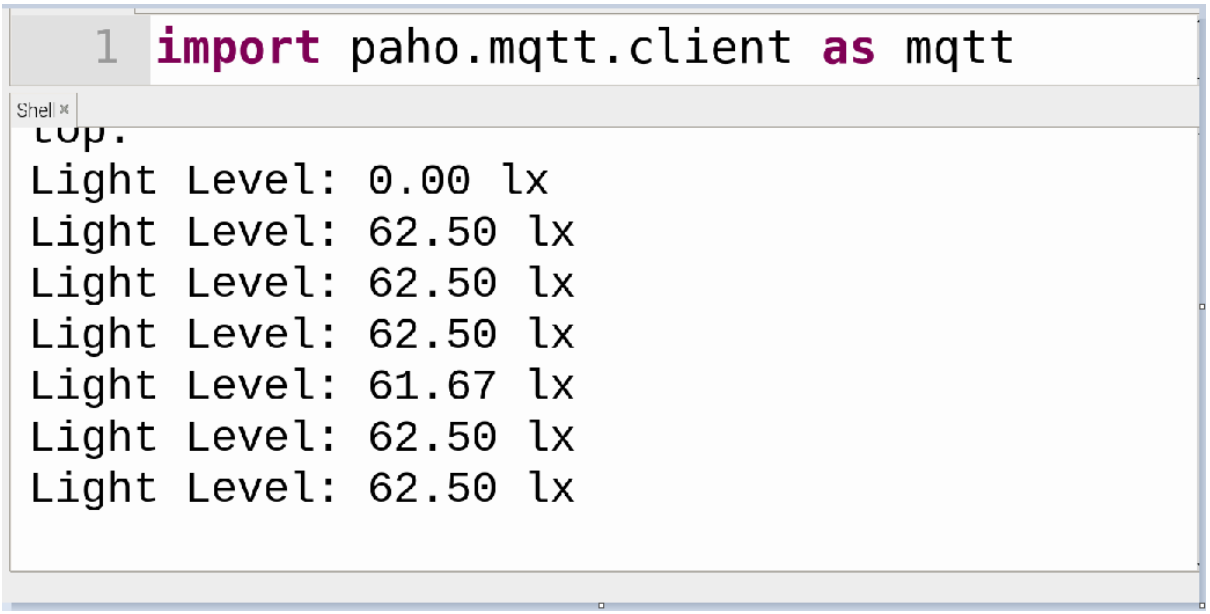


```
pi@raspberrypi:~ $ mosquitto_pub -h 192.168.1.13 -t test/topic -m "Hello from CrowPi"
pi@raspberrypi:~ $ 
```

In Server, the message “Hello from CrowPi” show that means connection worked.

Sensor Data collected

- Smart room collect Data and Send it to Server:

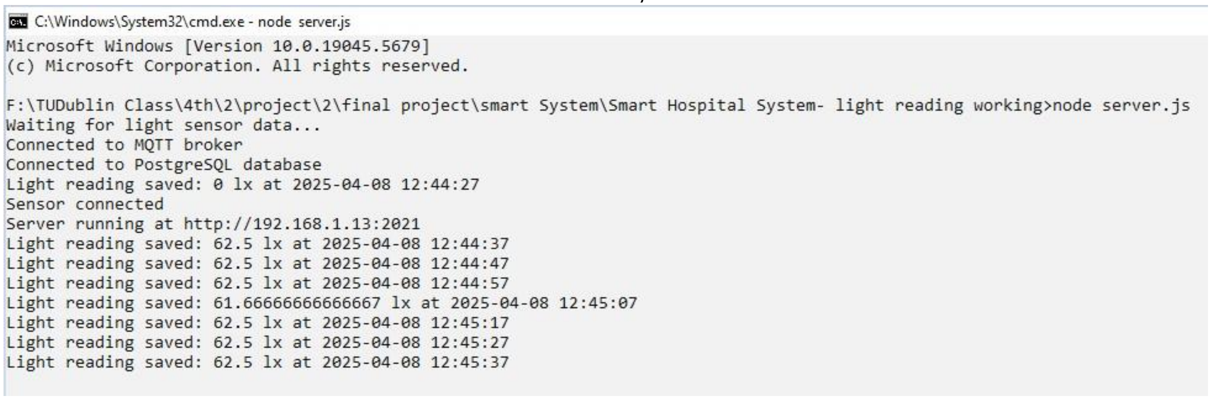


```
import paho.mqtt.client as mqtt
```

```
top.
Light Level: 0.00 lx
Light Level: 62.50 lx
Light Level: 62.50 lx
Light Level: 62.50 lx
Light Level: 61.67 lx
Light Level: 62.50 lx
Light Level: 62.50 lx
```

lx: light intensity

Server Part: Received Data from smart room, and saved in dataset:



```
C:\Windows\System32\cmd.exe - node server.js
Microsoft Windows [Version 10.0.19045.5679]
(c) Microsoft Corporation. All rights reserved.

F:\TUDublin Class\4th\2\project\2\final project\smart System\Smart Hospital System- light reading working>node server.js
Waiting for light sensor data...
Connected to MQTT broker
Connected to PostgreSQL database
Light reading saved: 0 lx at 2025-04-08 12:44:27
Sensor connected
Server running at http://192.168.1.13:2021
Light reading saved: 62.5 lx at 2025-04-08 12:44:37
Light reading saved: 62.5 lx at 2025-04-08 12:44:47
Light reading saved: 62.5 lx at 2025-04-08 12:44:57
Light reading saved: 61.666666666666667 lx at 2025-04-08 12:45:07
Light reading saved: 62.5 lx at 2025-04-08 12:45:17
Light reading saved: 62.5 lx at 2025-04-08 12:45:27
Light reading saved: 62.5 lx at 2025-04-08 12:45:37
```

Challenges faced:
At first, both devices used localhost (127.0.0.1) by mistake. This meant they couldn’t connect to each other.
The issue was fixed by using the server’s real IP address on both devices so they could exchange messages over the network.
Limitations or what was not finished:
Only light and sound sensors were included in this stage. Other sensors (like temperature or SpO_2) were not connected yet.

Data collection was working but needed more tuning before adding the robot and real-time alert handling.

4.2.4 Robot Movement & Vision

4.2.4.1What was developed:

To enable robot navigation, the MasterPi robot's operating system was upgraded from the original 32-bit Ubuntu version, which caused compatibility issues with required libraries. After installing a new 64-bit OS:

- All missing system packages and servers were installed to support robot control.
- Motor speed problems were fixed by adjusting the configuration in the Board.py file.
- The functionality of all motors and servos was tested to ensure smooth movement.
- These setup steps and screenshots are included in Appendix C: Setting Up the MasterPi Robot.

A Python script was then developed to control the robot's movement and obstacle avoidance. Using the Hiwonder SDK and ultrasonic sensor, the robot can:

- Move forward and sideways in small steps,
- Rotate 90° in either direction,
- Detect obstacles in front using distance readings,
- Shift sideways to avoid objects when needed,
- Move toward a target coordinate in X-Y space,
- Apply correction to its path after avoidance.

The robot receives commands via MQTT from the server, which includes the target patient's coordinates (x, y).
Once a patient alert is received:

- The robot waits for an MQTT message containing the patient's location.
- It navigates to that (x, y) coordinate, avoiding obstacles on the way.

- When it arrives, it moves slightly near the sound source to collect clearer readings and reduce environmental noise.

- After collecting the sound data (RMS, dB, ZCR), it returns to its original location using the stored initial_x and initial_y values.

This was achieved without external mapping or vision input. The full code is included in **Appendix C**.

4.2.4.3. How it was tested:

To test the robot movement and obstacle avoidance, a target location was set in one direction along the Y-axis. The robot was manually placed at the initial position and started after receiving a simulated MQTT message from the server with patient coordinates.
During testing:

- The robot successfully moved forward toward the target point.

- If an obstacle was placed in its path, the robot detected it using the ultrasonic sensor.

- It then moved sideways to avoid the obstacle and continued toward the target.

- After reaching the patient's position, it moved slightly near the sound source to simulate collecting better sound data (to improve signal clarity and reduce background noise).

This test confirmed that the robot could use location from MQTT messages, move, avoid obstacles, and perform a near-patient reading action as planned.

4.2.4.4. Challenges faced:

I originally planned to use A* for navigation and YOLO for object detection. However, I faced several technical problems:

- The YOLO library required by the robot's camera could not be installed on the original 32-bit OS version. Many machine learning and vision libraries now need a 64-bit system, and dependencies were missing.

- After updating the OS to 64-bit, I had to reconfigure the robot from scratch, which took time.

- Installing OpenCV with GPU support for YOLO was very complex and not fully supported on the MasterPi hardware.

- For A*, I needed real-time mapping and obstacle data, but the robot setup didn’t include LiDAR or accurate positioning sensors. This made it hard to implement A* pathfinding correctly.

4.2.4.5. Limitations or what was not finished:

- I did not implement A* or any full navigation algorithm. The robot currently moves using simple, pre-defined directions (like move forward or rotate).

- I also did not use YOLO for object or face detection because of OS and hardware limitations.

4.2.5 Web Dashboard & Alerts
4.2.5. What was developed:
Two web pages were created:

4.2.5.1 Main Dashboard Page (index.html):
Shows real-time light and sound readings from the smart room.
Displays a warning message if both Kalman Filter and Isolation Forest detect an abnormal value.
Plays a sound alert and logs the alert in a list for staff to review.

4.2.5.2. Charts Page (charts.html):
Displays 4 live line charts:

- Kalman RMS
- Kalman ZCR
- Isolation Forest RMS
- Isolation Forest ZCR

These show how sound features change over time.
Charts auto-update every 10 seconds using JavaScript.

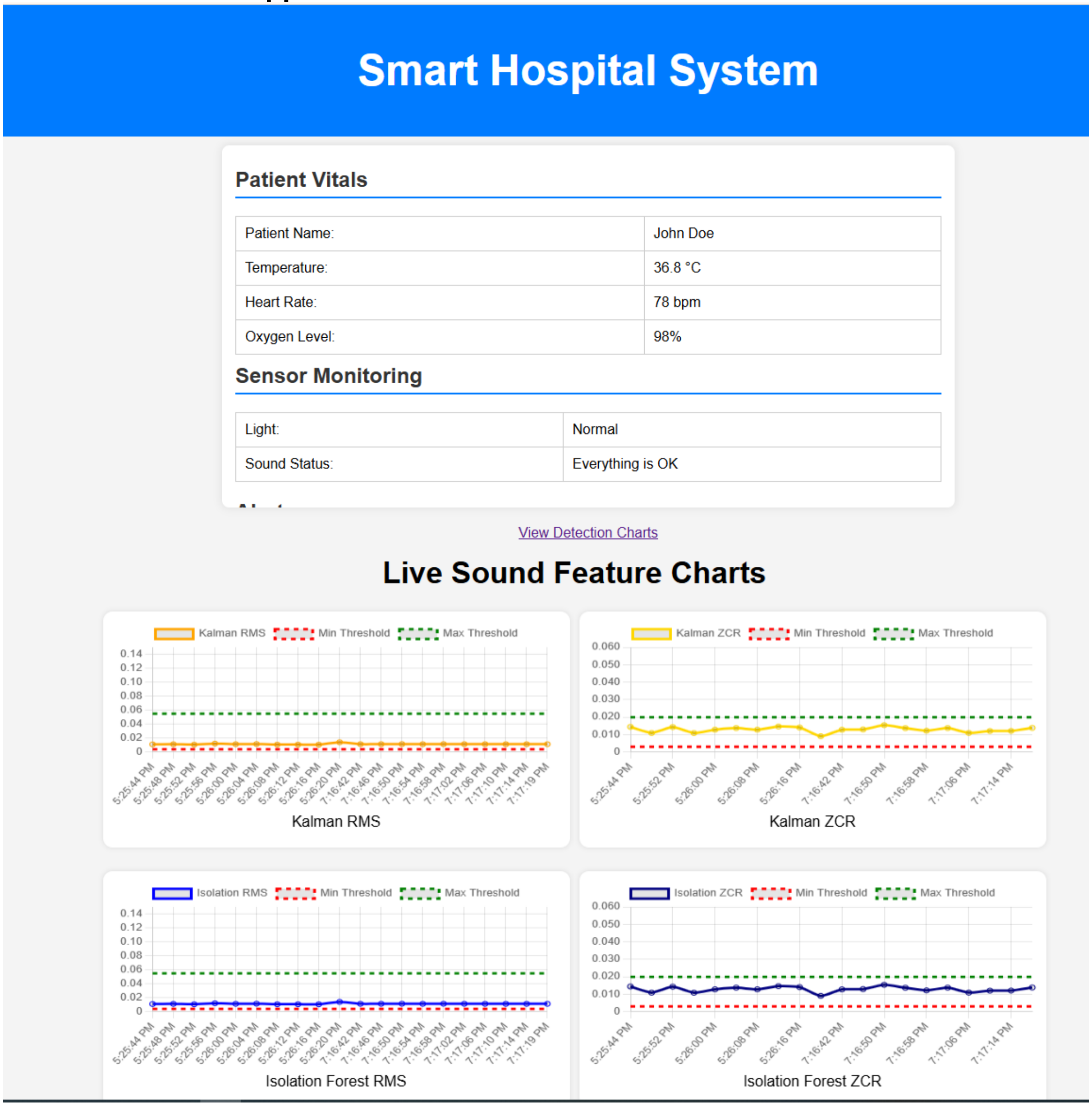
Challenges faced:

The charts were not updating with live data.
This issue was solved by setting a 10-second timer using JavaScript to fetch new data from the server and update the charts on the screen.

Limitations or what was not finished:

The design is simple and could be improved to look more professional in the future.

Final view of web application:



Smart Hospital System

Patient Vitals

Patient Name:	John Doe
Temperature:	36.8 °C
Heart Rate:	78 bpm
Oxygen Level:	98%

Sensor Monitoring

Light:	Normal
Sound Status:	Everything is OK

View Detection Charts

Live Sound Feature Charts

4.4. Conclusions:

This project successfully developed a smart hospital system that collects and filters real-time sensor data to detect abnormal light and sound conditions. Two detection methods, Kalman Filter and Isolation Forest, were tested and compared using real data.

Key achievements:

- A working system was built that connects CrowPi sensors and the MasterPi robot using MQTT.
- Both filtering methods were applied on real sensor data (light and sound).
- The system showed alerts in real-time and used a robot to improve sound data collection by reducing background noise.
- An improved feature set (RMS and ZCR) gave better results than using dB alone.
- A dashboard with live charts and alerts helps staff monitor patients more easily.

What was learned:

- Kalman Filter works well for fast, continuous data tracking.
- Isolation Forest is better for detecting sudden abnormal values using machine learning.
- Combining both methods gives stronger results and reduces false alerts.
- Using a robot to move closer to the patient improves sound detection.

This project shows how combining data filtering, machine learning, and robotics can make real-time patient monitoring smarter, more accurate, and more helpful in hospital environments.

5. Testing and Evaluation

5.1. Introduction

This section presents how the two anomaly detection methods (Kalman Filter and Isolation Forest) were tested and evaluated. To understand their accuracy and performance, several tests were conducted using pre-recorded audio feature datasets (not real-time data). The aim was to identify which method performs better in detecting abnormal sounds, such as coughing or shouting, and how the settings of each method can affect results.

The testing process included both real-time sensor readings and offline evaluation using collected datasets. Evaluation metrics such as accuracy, precision, and recall were used to compare the performance of both methods.

5.2 Evaluation Parameters:

To evaluate and compare the methods, the following metrics were used:

- True Positive (TP): Abnormal sounds correctly detected as abnormal.
- True Negative (TN): Normal sounds correctly detected as normal.
- False Positive (FP): Normal sounds incorrectly detected as abnormal.
- False Negative (FN): Abnormal sounds incorrectly detected as normal.

From these, the following scores were calculated:

- Accuracy: Proportion of correct predictions.

  Formula: (TP + TN) / (TP + TN + FP + FN)

- Precision: Proportion of correctly predicted abnormal sounds out of all predicted abnormal.

  Formula: TP / (TP + FP)

- Recall (Sensitivity): Proportion of correctly detected abnormal sounds out of all actual abnormal.

  Formula: TP / (TP + FN)

These metrics help in understanding the overall performance and reliability of each method.

5.3 Testing Using Collected Feature Datasets:

The tests were done using two datasets that were collected from online sources:

- Noise Dataset (Normal): Includes common hospital background noise such as TV, fan, walking, and phone sounds.

- Abnormal Dataset: Includes different types of health-related sounds such as coughs and vocal signs of pain (e.g., "o", "e", "a").

The references for these datasets are listed in the References section.[35], [36]

Both models were tested using both datasets noise data as normal data, and abnormal data. Results were saved and evaluated to see how well each method detects anomalies.
The Isolation Forest model was trained using the normal dataset, and both models were tested using both datasets. Results were saved and evaluated to see how well each method detects anomalies.

5.3.1 Initial Results:

This table shows the baseline results for both methods before tuning any parameters:

Method	Dataset	TP	TN	FP	FN	Accuracy	Recall	Precision
Isolation Forest	Abnormal	108	0	0	95	0.53	0.53	1.00
	Normal	0	491	55	0	0.90	-	0.00
Kalman Filter	Abnormal	137	0	0	66	0.67	0.67	1.00
	Normal	0	336	210	0	0.62	-	0.00

Table 5.1: first test result

This table shows the initial performance of both anomaly detection methods. The Isolation Forest model performed well in detecting abnormal sounds (precision = 1.00) but missed many actual abnormal cases (recall = 0.53). The Kalman Filter had slightly better recall (0.67), meaning it was more sensitive to changes, but it also produced more false positives in the normal dataset, leading to lower accuracy. These results highlight the need for fine-tuning to find the right balance between sensitivity and false alarms.

5.3.2 Parameter Tuning for Anomaly Detection Methods:

To improve the performance of the Isolation Forest model, different contamination values were tested. The contamination parameter controls the expected proportion of anomalies in the training data. A higher value makes the model more sensitive to unusual patterns but may cause more false positives. A lower value reduces false alarms but may miss real anomalies. The purpose of this tuning was to find the best value that gives a good balance between accuracy, precision, and recall. The table below shows the results for different contamination settings.

Contamination	Dataset	TP	TN	FP	FN	Accuracy	Recall	Precision
0.05	Abnormal	99	-	-	104	0.49	0.49	1.00
	Normal	-	518	28	-	0.95	-	0.00
0.10	Abnormal	108	-	-	95	0.53	0.53	1.00
	Normal	-	491	55	-	0.90	-	0.00

0.15	Abnormal	106	-	-	97	0.52	0.52	1.00
	Normal	-	491	55	-	0.90	-	0.00

Table 5.2: proportion of anomalies when contamination levels changed

Table 5.2 shows that best Setting when Contamination = 0.10 offers a balance between high recall and acceptable accuracy on normal data.
A contamination value of 0.10 allows the model to detect more true anomalies without increasing false positives too much. It improves recall while maintaining high accuracy for the normal dataset, making it a reliable setting for healthcare environments.

5.3.3. Kalman Filter Threshold Tuning:

Different residual thresholds were tested to control how sensitive the Kalman Filter is to changes.
Lower thresholds make the filter more sensitive but increase false positives. Higher thresholds reduce false alarms but may miss real anomalies. Tuning helps find a balance between early detection and reliability.

Threshold	Dataset	TP	TN	FP	FN	Accuracy	Recall	Precision
2.0	Abnormal	157	-	-	46	0.77	0.77	1.00
	Normal	-	227	319	-	0.42	-	0.00
3.0	Abnormal	137	-	-	66	0.67	0.67	1.00
	Normal	-	336	210	-	0.62	-	0.00
4.0	Abnormal	120	-	-	83	0.59	0.59	1.00
	Normal	-	408	138	-	0.75	-	0.00

Table 5.3: proportion of anomalies when Threshold changed

Table 5.3 show that best Setting when Threshold = 3.0 offers good recall and a fair balance of false positives.
This threshold was chosen because it detects most real anomalies while reducing the number of false alarms compared to lower thresholds.
The next section will begin testing both methods with real live sensor data, followed by evaluating how the robot's movement can improve the accuracy of both methods.

In both tuning sections, the aim was to find the best balance between catching true anomalies and avoiding false alarms.

5.3.4. Using Feature Ranges to Improve Detection:

To improve both filtering methods, I analyzed the training dataset to find the typical range of values for noise sounds. I used the dataset that contains normal noise such as TV and people talking. From that dataset, I calculated the most common values for the three sound features:

- db was usually greater than 70
- rms was often greater than 0.05

- zcr was between 0.06 and 0.1

I applied this filter in the code using the line:

```
(train_df["db"] > 70) & (train_df["rms"] > 0.05) & (train_df["zcr"] > 0.06) & (train_df["zcr"] <
0.1)
```

This filter helped clean the training data by keeping only strong and clear noise samples. I used this filtered data to train the Isolation Forest model. Kalman Filter does not learn from data, but it also used the same range for consistency.

After testing with abnormal and noise data, I found that Isolation Forest became much more accurate. The recall increased and the model was better at ignoring false alarms. Kalman Filter performance stayed the same.

5.3.4. 1. Results Before and After Filtering:

Kalman Filter results did not change because it does not use training data, but this experiment showed that filtering feature ranges can improve accuracy and recall for data-driven models like Isolation Forest.

Isolation Forest model:

Method	**Dataset**	**TP**	**TN**	**FP**	**FN**	**Accuracy**	**Recall**	**Precision**
Before Update	Abnormal	108	0	0	95	0.53	0.53	1.00
After Update	Abnormal	173	0	0	30	0.85	0.85	1.00
Before Update	Normal	0	491	55	0	0.90	0.90	0.00
After Update	Normal	180	0	0	23	0.89	0.89	1.00

Table 5.4: Results Before and After Filtering

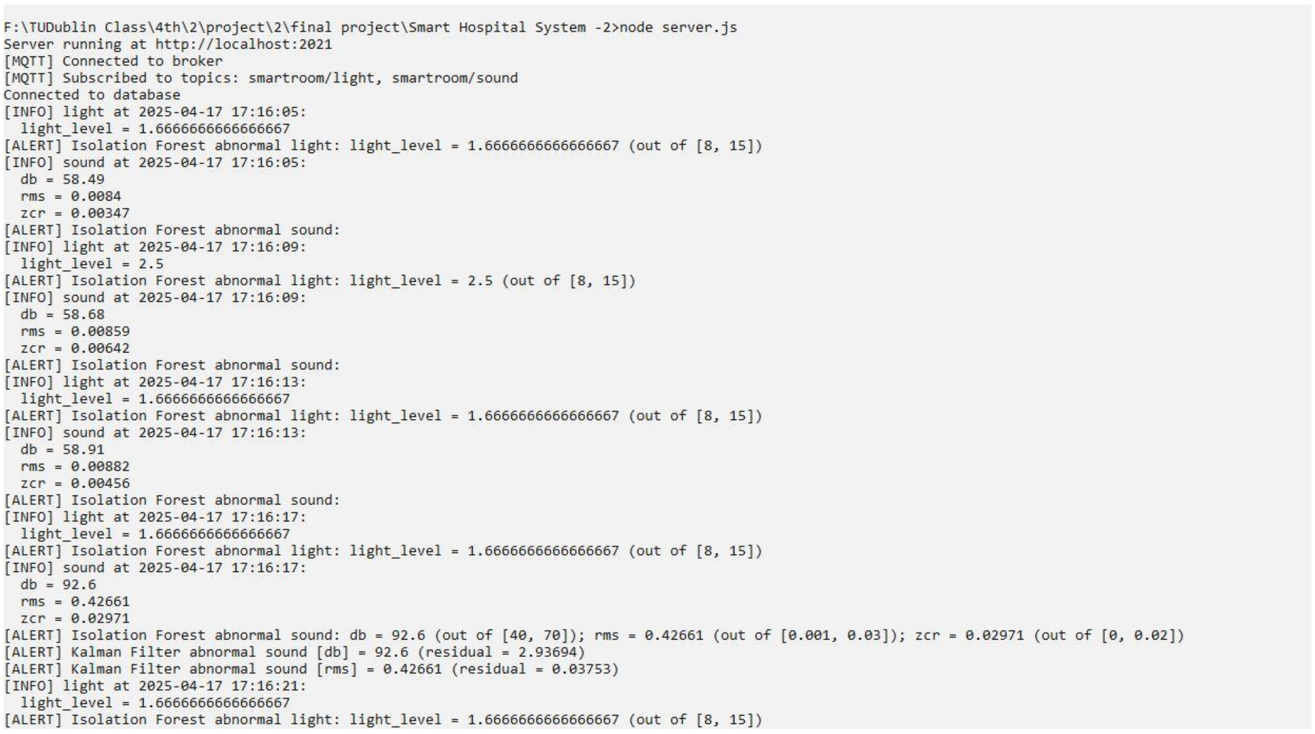
Although the Kalman Filter does not use training data like Isolation Forest, I applied feature ranges (db, rms, and zcr) in the PostgreSQL query in code to control which sensor readings were selected for analysis.

This helped the Kalman Filter work with more stable and clean noise data. As a result, the predictions became more consistent and reliable when detecting changes. This shows that filtering input data can also improve the Kalman Filter's performance during live monitoring.

5.4. Real-Time Testing with Two Patients

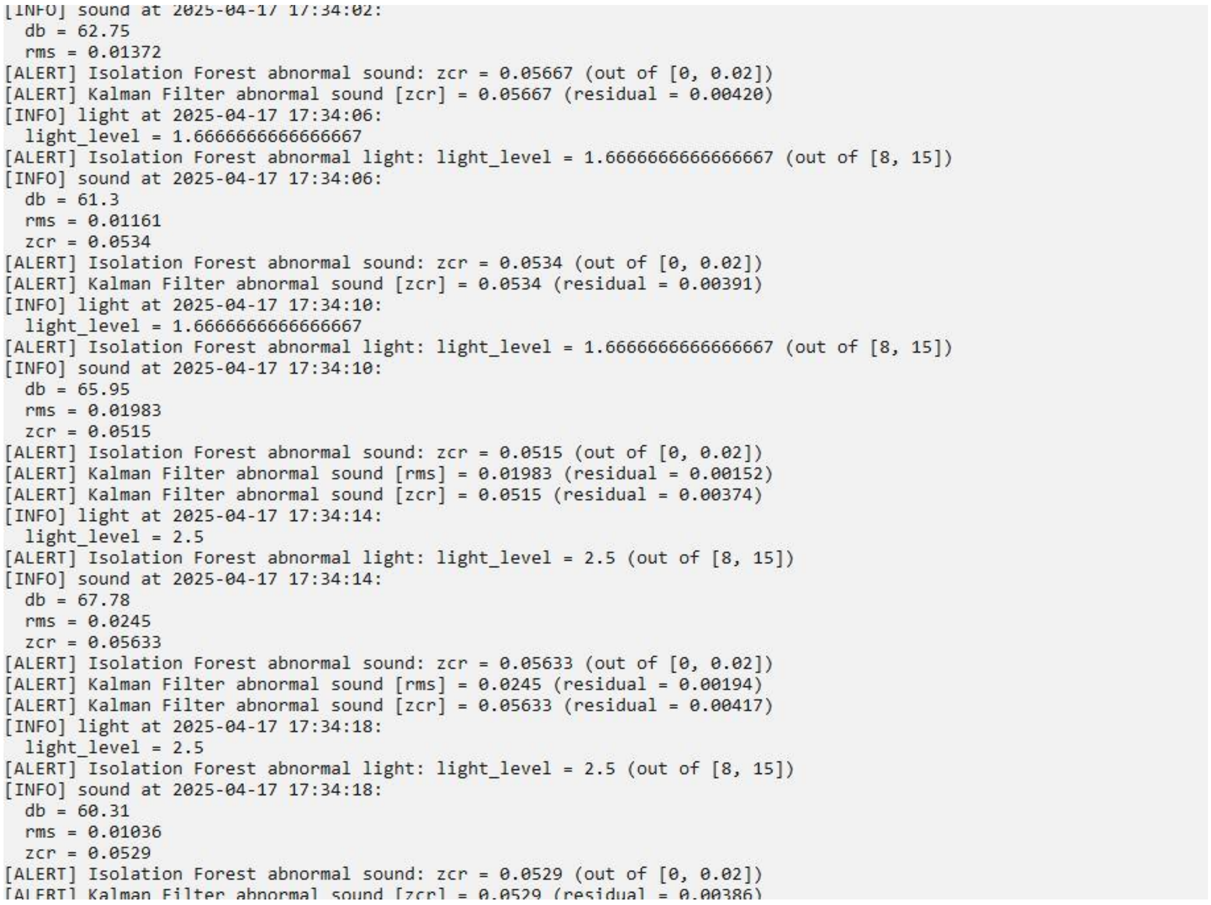
The final system was tested using real live data from two different patients. One patient (Patient 6) had abnormal sound readings, and the other (Patient 5) had normal sound levels with some background noise.

The goal was to check if the system could correctly detect only real abnormal data and not raise false alarms for noisy but safe values.



```
F:\TUDublin Class\4th\2\project\2\final project\Smart Hospital System -2>node server.js
Server running at http://localhost:2021
[MQTT] Connected to broker
[MQTT] Subscribed to topics: smartroom/light, smartroom/sound
Connected to database
[INFO] light at 2025-04-17 17:16:05:
  light_level = 1.6666666666666667
[ALERT] Isolation Forest abnormal light: light_level = 1.6666666666666667 (out of [8, 15])
[INFO] sound at 2025-04-17 17:16:05:
  db = 58.49
  rms = 0.0084
  zcr = 0.00347
[ALERT] Isolation Forest abnormal sound:
[INFO] light at 2025-04-17 17:16:09:
  light_level = 2.5
[ALERT] Isolation Forest abnormal light: light_level = 2.5 (out of [8, 15])
[INFO] sound at 2025-04-17 17:16:09:
  db = 58.68
  rms = 0.00859
  zcr = 0.00642
[ALERT] Isolation Forest abnormal sound:
[INFO] light at 2025-04-17 17:16:13:
  light_level = 1.6666666666666667
[ALERT] Isolation Forest abnormal light: light_level = 1.6666666666666667 (out of [8, 15])
[INFO] sound at 2025-04-17 17:16:13:
  db = 58.91
  rms = 0.00882
  zcr = 0.00456
[ALERT] Isolation Forest abnormal sound:
[INFO] light at 2025-04-17 17:16:17:
  light_level = 1.6666666666666667
[ALERT] Isolation Forest abnormal light: light_level = 1.6666666666666667 (out of [8, 15])
[INFO] sound at 2025-04-17 17:16:17:
  db = 92.6
  rms = 0.42661
  zcr = 0.02971
[ALERT] Isolation Forest abnormal sound: db = 92.6 (out of [40, 70]); rms = 0.42661 (out of [0.001, 0.03]); zcr = 0.02971 (out of [0, 0.02])
[ALERT] Kalman Filter abnormal sound [db] = 92.6 (residual = 2.93694)
[ALERT] Kalman Filter abnormal sound [rms] = 0.42661 (residual = 0.03753)
[INFO] light at 2025-04-17 17:16:21:
  light_level = 1.6666666666666667
[ALERT] Isolation Forest abnormal light: light_level = 1.6666666666666667 (out of [8, 15])
```

Figure 5.1: first sound test-A



```
[INFO] sound at 2025-04-17 17:34:02:
  db = 62.75
  rms = 0.01372
[ALERT] Isolation Forest abnormal sound: zcr = 0.05667 (out of [0, 0.02])
[ALERT] Kalman Filter abnormal sound [zcr] = 0.05667 (residual = 0.00420)
[INFO] light at 2025-04-17 17:34:06:
  light_level = 1.6666666666666667
[ALERT] Isolation Forest abnormal light: light_level = 1.6666666666666667 (out of [8, 15])
[INFO] sound at 2025-04-17 17:34:06:
  db = 61.3
  rms = 0.01161
  zcr = 0.0534
[ALERT] Isolation Forest abnormal sound: zcr = 0.0534 (out of [0, 0.02])
[ALERT] Kalman Filter abnormal sound [zcr] = 0.0534 (residual = 0.00391)
[INFO] light at 2025-04-17 17:34:10:
  light_level = 1.6666666666666667
[ALERT] Isolation Forest abnormal light: light_level = 1.6666666666666667 (out of [8, 15])
[INFO] sound at 2025-04-17 17:34:10:
  db = 65.95
  rms = 0.01983
  zcr = 0.0515
[ALERT] Isolation Forest abnormal sound: zcr = 0.0515 (out of [0, 0.02])
[ALERT] Kalman Filter abnormal sound [rms] = 0.01983 (residual = 0.00152)
[ALERT] Kalman Filter abnormal sound [zcr] = 0.0515 (residual = 0.00374)
[INFO] light at 2025-04-17 17:34:14:
  light_level = 2.5
[ALERT] Isolation Forest abnormal light: light_level = 2.5 (out of [8, 15])
[INFO] sound at 2025-04-17 17:34:14:
  db = 67.78
  rms = 0.0245
  zcr = 0.05633
[ALERT] Isolation Forest abnormal sound: zcr = 0.05633 (out of [0, 0.02])
[ALERT] Kalman Filter abnormal sound [rms] = 0.0245 (residual = 0.00194)
[ALERT] Kalman Filter abnormal sound [zcr] = 0.05633 (residual = 0.00417)
[INFO] light at 2025-04-17 17:34:18:
  light_level = 2.5
[ALERT] Isolation Forest abnormal light: light_level = 2.5 (out of [8, 15])
[INFO] sound at 2025-04-17 17:34:18:
  db = 60.31
  rms = 0.01036
  zcr = 0.0529
[ALERT] Isolation Forest abnormal sound: zcr = 0.0529 (out of [0, 0.02])
[ALERT] Kalman Filter abnormal sound [zcr] = 0.0529 (residual = 0.00386)
```

Figure 5.2: first sound test-B

5.4.1Feature Analysis and Live Data Ranges

Data was collected live from both patients and grouped into three categories. The following tables show the **minimum and maximum values** for each sound feature in each group. Shown in below Tables:

Table 5.5 show Real Abnormal Sound Data (Patient 6):

Feature	**Min**	**Max**
db	61.20	92.26
rms	0.01148	0.41041
zcr	0.00417	0.08073

Table 5.5: Real Abnormal Sound Data (Patient 6)

Table 5.6 show Fake Abnormal (Noise Mistaken as Abnormal by System):

Feature	Min	Max
db	68.24	82.27
rms	0.02583	0.12981
zcr	0.00467	0.05558

Table 5.6: Fake Abnormal (Noise Mistaken as Abnormal by System)

Table 5.7 show Real Normal and Noise Data (Not Marked as Abnormal):

Feature	Min	Max
db	68.28	69.49
rms	0.02594	0.02983
zcr	0.00592	0.00651

Table 5.6: Real Normal and Noise Data (Not Marked as Abnormal)

5.4.1.1 Feature Similarities:

Some ranges were shared between all groups, which caused confusion for the system:

- db from 68.28 to 69.49 appears in all three types.
- rms between 0.026 and 0.030 is seen in real abnormal, noise, and normal.
- zcr around 0.005 to 0.006 is also found in every group.

5.4.1.2. Feature Differences:

- Db: Real abnormal values go much lower and higher than others. But there is also overlap with safe ranges.
- Rms: Real abnormal values go up to 0.41041, which is much higher than noise.
- Zcr: Real abnormal values can be very high (up to 0.08073), not found in safe or noisy readings.

5.4.1.3. Best Feature for Detection:

- Db: Overlaps with both real and fake abnormal data. Not useful.
- Rms: Better. Values above 0.12981 clearly show abnormal cases.
- Zcr: Also better. Values above 0.00651 are only seen in real abnormal readings.

The best combination for detecting real abnormal sound is using rms + zcr and removing db.

5.5. Improvements for Both Methods:

To increase the accuracy and reduce false alarms:

- Remove db from both Isolation Forest and Kalman Filter.
- Use only rms and zcr.
- Update the sensor configuration in the code:

```
"sound": {
   "columns": ["rms", "zcr"],
   "csv_file": "trainingData.csv",
   "training_source": "csv",
   "min_values": [0.02594, 0.00592],
   "max_values": [0.02983, 0.00651]
}
```

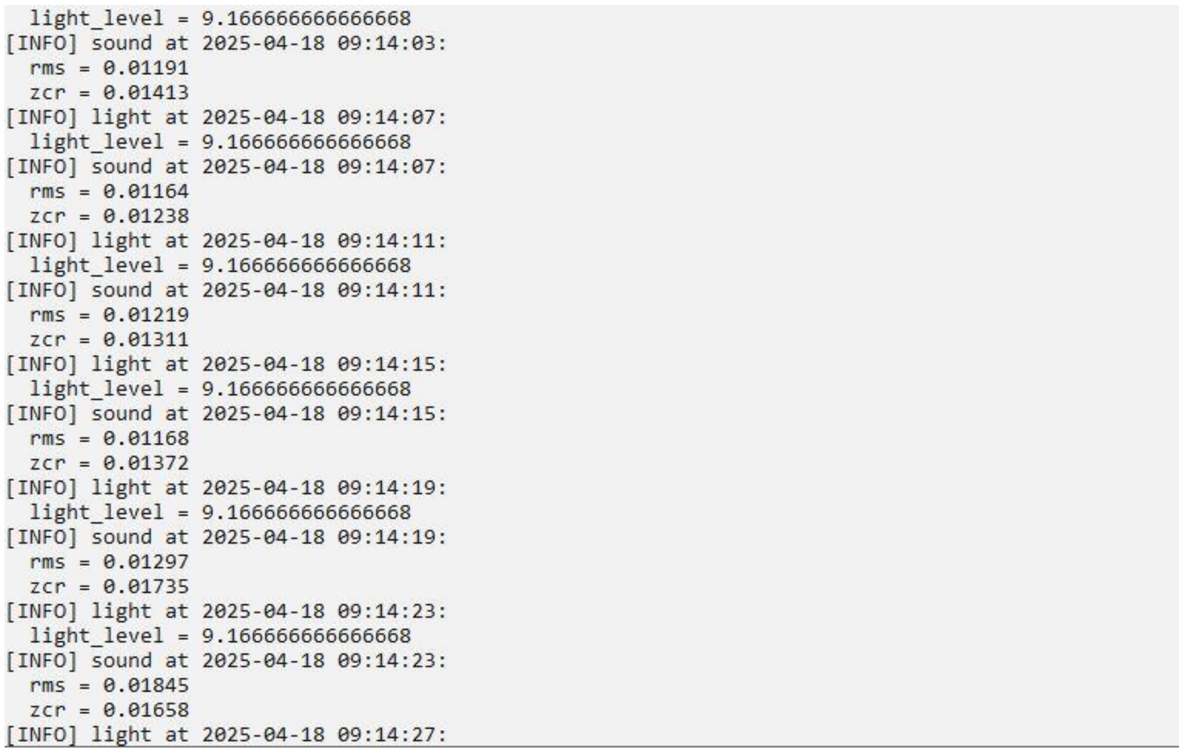
- Use stricter Kalman thresholds:
    - rms residual > 0.0015
    - zcr residual > 0.001

5.6. Update Training Data and Re-Test:
The training data was updated to match the real safe and abnormal ranges. A new CSV file was created with:

- 100 records labelled as abnormal (based on real Patient 6)
- 50 records of normal and 50 of noise labelled as normal

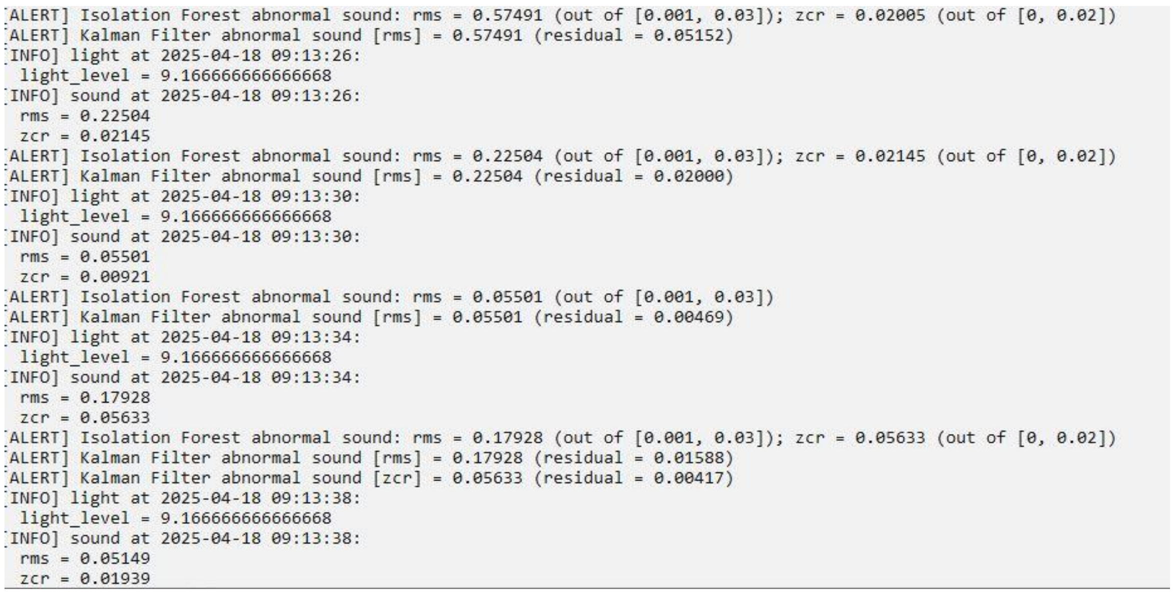
This updated data was used for training. The system was then re-tested using this data as real input to simulate live readings. Both Isolation Forest and Kalman Filter gave strong results, correctly detecting real abnormal values and ignoring normal/noise.

Below figures show results of test after improvement:



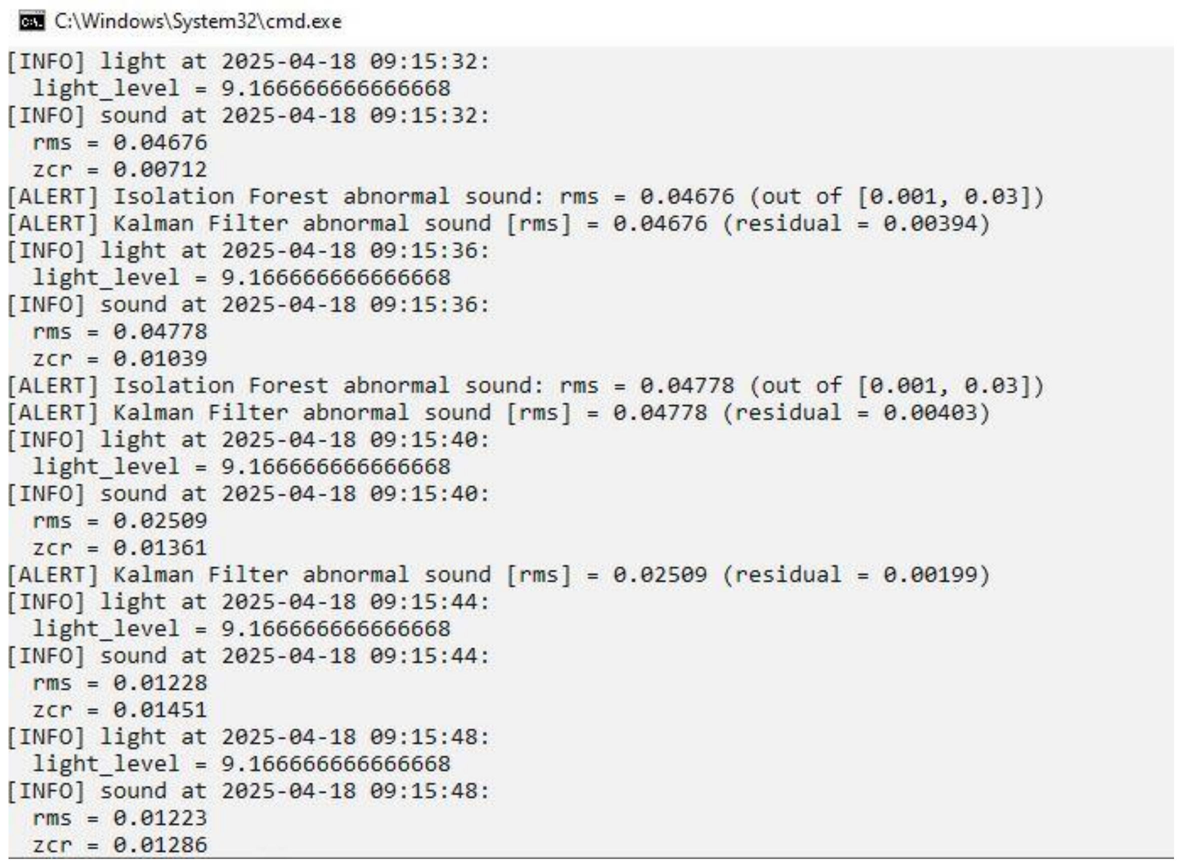
```
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:14:03:
  rms = 0.01191
  zcr = 0.01413
[INFO] light at 2025-04-18 09:14:07:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:14:07:
  rms = 0.01164
  zcr = 0.01238
[INFO] light at 2025-04-18 09:14:11:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:14:11:
  rms = 0.01219
  zcr = 0.01311
[INFO] light at 2025-04-18 09:14:15:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:14:15:
  rms = 0.01168
  zcr = 0.01372
[INFO] light at 2025-04-18 09:14:19:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:14:19:
  rms = 0.01297
  zcr = 0.01735
[INFO] light at 2025-04-18 09:14:23:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:14:23:
  rms = 0.01845
  zcr = 0.01658
[INFO] light at 2025-04-18 09:14:27:
```

Figure 5.3: 2nd live test normal data



```
ALERT] Isolation Forest abnormal sound: rms = 0.57491 (out of [0.001, 0.03]); zcr = 0.02005 (out of [0, 0.02])
ALERT] Kalman Filter abnormal sound [rms] = 0.57491 (residual = 0.05152)
INFO] light at 2025-04-18 09:13:26:
 light_level = 9.16666666666668
INFO] sound at 2025-04-18 09:13:26:
 rms = 0.22504
 zcr = 0.02145
ALERT] Isolation Forest abnormal sound: rms = 0.22504 (out of [0.001, 0.03]); zcr = 0.02145 (out of [0, 0.02])
ALERT] Kalman Filter abnormal sound [rms] = 0.22504 (residual = 0.02000)
INFO] light at 2025-04-18 09:13:30:
 light_level = 9.16666666666668
INFO] sound at 2025-04-18 09:13:30:
 rms = 0.05501
 zcr = 0.00921
ALERT] Isolation Forest abnormal sound: rms = 0.05501 (out of [0.001, 0.03])
ALERT] Kalman Filter abnormal sound [rms] = 0.05501 (residual = 0.00469)
INFO] light at 2025-04-18 09:13:34:
 light_level = 9.16666666666668
INFO] sound at 2025-04-18 09:13:34:
 rms = 0.17928
 zcr = 0.05633
ALERT] Isolation Forest abnormal sound: rms = 0.17928 (out of [0.001, 0.03]); zcr = 0.05633 (out of [0, 0.02])
ALERT] Kalman Filter abnormal sound [rms] = 0.17928 (residual = 0.01588)
ALERT] Kalman Filter abnormal sound [zcr] = 0.05633 (residual = 0.00417)
INFO] light at 2025-04-18 09:13:38:
 light_level = 9.16666666666668
INFO] sound at 2025-04-18 09:13:38:
 rms = 0.05149
 zcr = 0.01939
```

Figure 5.4: 2nd live test false abnormal data



```
C:\Windows\System32\cmd.exe
[INFO] light at 2025-04-18 09:15:32:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:15:32:
  rms = 0.04676
  zcr = 0.00712
[ALERT] Isolation Forest abnormal sound: rms = 0.04676 (out of [0.001, 0.03])
[ALERT] Kalman Filter abnormal sound [rms] = 0.04676 (residual = 0.00394)
[INFO] light at 2025-04-18 09:15:36:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:15:36:
  rms = 0.04778
  zcr = 0.01039
[ALERT] Isolation Forest abnormal sound: rms = 0.04778 (out of [0.001, 0.03])
[ALERT] Kalman Filter abnormal sound [rms] = 0.04778 (residual = 0.00403)
[INFO] light at 2025-04-18 09:15:40:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:15:40:
  rms = 0.02509
  zcr = 0.01361
[ALERT] Kalman Filter abnormal sound [rms] = 0.02509 (residual = 0.00199)
[INFO] light at 2025-04-18 09:15:44:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:15:44:
  rms = 0.01228
  zcr = 0.01451
[INFO] light at 2025-04-18 09:15:48:
  light_level = 9.16666666666668
[INFO] sound at 2025-04-18 09:15:48:
  rms = 0.01223
  zcr = 0.01286
```

Figure 5.5: 2nd live test abnormal data

Alos there is video show how whole system work to gether.

5.7. Adjustment to Improve Detection (ZCR and Light Level):

During testing, Isolation Forest flagged several readings as abnormal based on ZCR values, even though those values were inside the expected noise range. For example:

ZCR = 0.01937 (threshold was [0.003, 0.015])

To fix this issue, the maximum ZCR value was increased to 0.02 in the system:

- The sensor_config in data_filtering.py was updated
- The trainingData.csv file was also updated to include safe ZCR values up to 0.02
- The model was trained again, and the system was re-tested.

In addition, some light sensor readings were also marked as abnormal by Isolation Forest, even though they were considered safe. To fix this, the maximum light level in the configuration was increased to 25 instead of the previous value of 15. This change helped prevent false detection of normal light values as abnormal.

5.9. Retesting with Collected Test Data:

To simulate a realistic hospital environment, I created a new test dataset using collected data that included three types of sound cases:

- Normal and Noise: Common hospital sounds like walking, fan, TV.
- Abnormal: Coughs, pain sounds, and other health-related alerts.
- Uncertain: Borderline cases that could confuse the system, such as medium-level RMS or ZCR values close to the normal/abnormal threshold.

This helped test whether the models can handle difficult situations, not just clear cases.

5.9.1. Final Evaluation Results:

Metric	**Isolation Forest**	**Kalman Filter**
Accuracy	**84%**	70%
Precision	78%	**100%**
Recall	**94%**	41%
TP	94	41
FP	26	**0**
FN	6	59
TN	74	**100**

Table 5.7: Final Evaluation Results

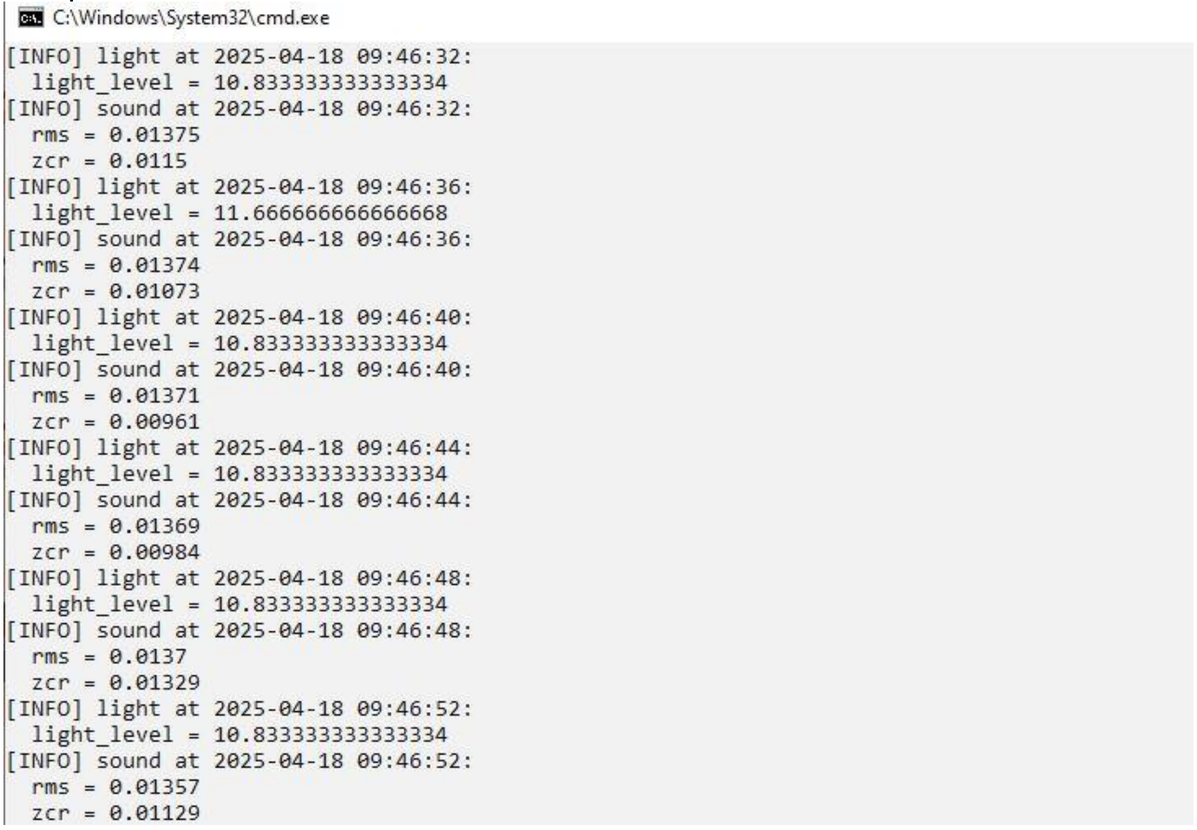
5.10. Why I Did Not Calculate Accuracy, Precision, or Recall for Live Data:

Accuracy, precision, and recall were not calculated for live patient data because the true labels (ground truth) for each reading were not available. In a real hospital, this would require doctors to manually label each sound as abnormal or normal, which is not possible during regular testing. Instead, I used feature ranges and expert analysis to observe how well the system responded to live sensor inputs.

5.11. Final Observation of System Behaviour:

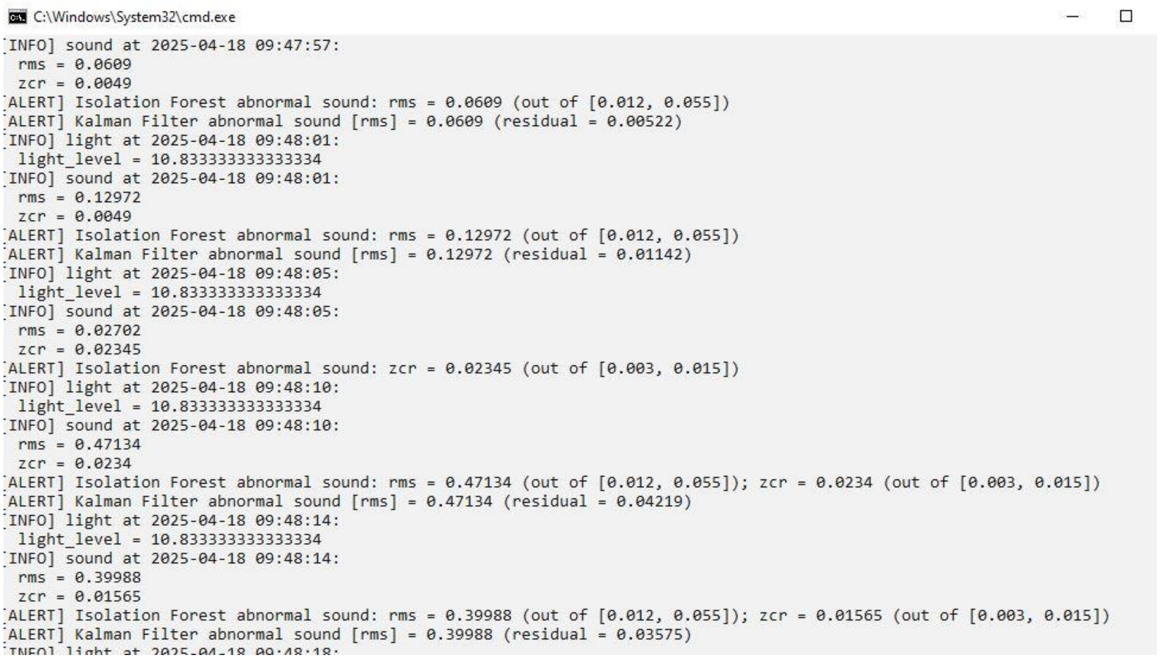
The final version of the system worked almost correctly during real-time monitoring. It was able to detect abnormal sounds at the right moments and ignored most normal background noise. The server received sound data, analysed it, and triggered alerts from both Kalman Filter and Isolation Forest when necessary.

To improve sound accuracy, the robot was used to move closer to the sound source. By getting near the patient, the microphone could pick up clearer sound signals and reduce background noise from the room. This helped the system make better decisions and improved the overall detection results.The figures below show examples of this behaviour:



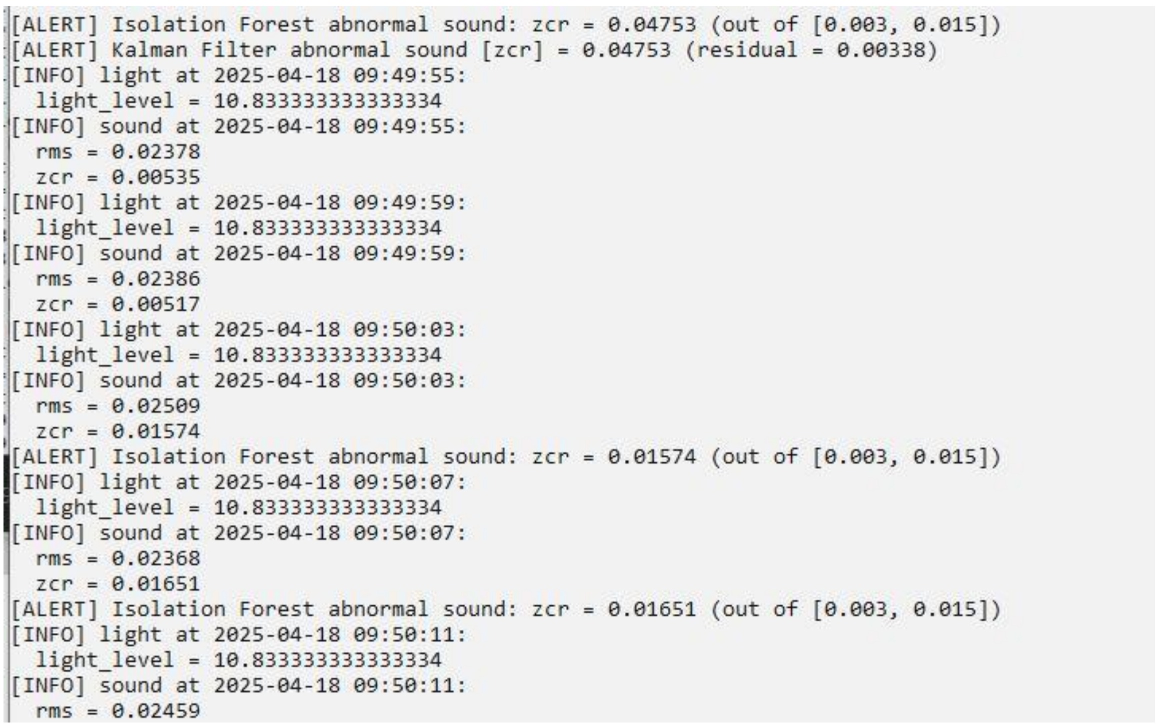
```
C:\Windows\System32\cmd.exe
[INFO] light at 2025-04-18 09:46:32:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:46:32:
  rms = 0.01375
  zcr = 0.0115
[INFO] light at 2025-04-18 09:46:36:
  light_level = 11.666666666666668
[INFO] sound at 2025-04-18 09:46:36:
  rms = 0.01374
  zcr = 0.01073
[INFO] light at 2025-04-18 09:46:40:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:46:40:
  rms = 0.01371
  zcr = 0.00961
[INFO] light at 2025-04-18 09:46:44:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:46:44:
  rms = 0.01369
  zcr = 0.00984
[INFO] light at 2025-04-18 09:46:48:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:46:48:
  rms = 0.0137
  zcr = 0.01329
[INFO] light at 2025-04-18 09:46:52:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:46:52:
  rms = 0.01357
  zcr = 0.01129
```

Figure 5.6: 3rd live test normal data



```
C:\Windows\System32\cmd.exe
INFO] sound at 2025-04-18 09:47:57:
 rms = 0.0609
 zcr = 0.0049
ALERT] Isolation Forest abnormal sound: rms = 0.0609 (out of [0.012, 0.055])
ALERT] Kalman Filter abnormal sound [rms] = 0.0609 (residual = 0.00522)
INFO] light at 2025-04-18 09:48:01:
 light_level = 10.833333333333334
INFO] sound at 2025-04-18 09:48:01:
 rms = 0.12972
 zcr = 0.0049
ALERT] Isolation Forest abnormal sound: rms = 0.12972 (out of [0.012, 0.055])
ALERT] Kalman Filter abnormal sound [rms] = 0.12972 (residual = 0.01142)
INFO] light at 2025-04-18 09:48:05:
 light_level = 10.833333333333334
INFO] sound at 2025-04-18 09:48:05:
 rms = 0.02702
 zcr = 0.02345
ALERT] Isolation Forest abnormal sound: zcr = 0.02345 (out of [0.003, 0.015])
INFO] light at 2025-04-18 09:48:10:
 light_level = 10.833333333333334
INFO] sound at 2025-04-18 09:48:10:
 rms = 0.47134
 zcr = 0.0234
ALERT] Isolation Forest abnormal sound: rms = 0.47134 (out of [0.012, 0.055]); zcr = 0.0234 (out of [0.003, 0.015])
ALERT] Kalman Filter abnormal sound [rms] = 0.47134 (residual = 0.04219)
INFO] light at 2025-04-18 09:48:14:
 light_level = 10.833333333333334
INFO] sound at 2025-04-18 09:48:14:
 rms = 0.39988
 zcr = 0.01565
ALERT] Isolation Forest abnormal sound: rms = 0.39988 (out of [0.012, 0.055]); zcr = 0.01565 (out of [0.003, 0.015])
ALERT] Kalman Filter abnormal sound [rms] = 0.39988 (residual = 0.03575)
INFO] light at 2025-04-18 09:48:18:
```

Figure 5.7: 3rd live test false abnormal data



```
[ALERT] Isolation Forest abnormal sound: zcr = 0.04753 (out of [0.003, 0.015])
[ALERT] Kalman Filter abnormal sound [zcr] = 0.04753 (residual = 0.00338)
[INFO] light at 2025-04-18 09:49:55:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:49:55:
  rms = 0.02378
  zcr = 0.00535
[INFO] light at 2025-04-18 09:49:59:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:49:59:
  rms = 0.02386
  zcr = 0.00517
[INFO] light at 2025-04-18 09:50:03:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:50:03:
  rms = 0.02509
  zcr = 0.01574
[ALERT] Isolation Forest abnormal sound: zcr = 0.01574 (out of [0.003, 0.015])
[INFO] light at 2025-04-18 09:50:07:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:50:07:
  rms = 0.02368
  zcr = 0.01651
[ALERT] Isolation Forest abnormal sound: zcr = 0.01651 (out of [0.003, 0.015])
[INFO] light at 2025-04-18 09:50:11:
  light_level = 10.833333333333334
[INFO] sound at 2025-04-18 09:50:11:
  rms = 0.02459
```

Figure 5.8: 3rd live test abnormal data

A short video was also created to demonstrate the full system running live. The video explains how data is collected from sensors, processed by the Python filtering code, sent through MQTT to the Node.js server, stored in PostgreSQL, and finally displayed as alerts on the dashboard.

5.4. Conclusions:

The final system was successfully tested using offline datasets, parameter tuning, feature range filtering, and live patient monitoring. The best results were achieved when using only RMS and ZCR features. Removing dB helped reduce confusion and false alarms.

During testing with collected data, the Isolation Forest model gave better results than Kalman Filter. It had higher accuracy and recall because it was trained using many normal examples and could clearly spot strange or abnormal sounds.

However, in real-time monitoring, the Kalman Filter worked better. It reacted faster to new values and was more stable with live sensor data. Kalman does not need training and updates its prediction every time, which makes it better for live hospital environments.

Using both methods together gave more reliable results and helped detect real problems while reducing false alerts.

6. Conclusions and Future Work

6.1. Conclusions

This project successfully built and tested a real-time system to monitor patients in a smart hospital room using sensor data, machine learning, and a mobile robot. The goal was to detect abnormal conditions like sudden sounds or light changes that may mean a patient needs help. The system collects data using Smart Room sensors, sends it to a server using MQTT, stores it in PostgreSQL, and filters the data using two methods: Kalman Filter and Isolation Forest.

The results showed that using only RMS and ZCR sound features gave the best performance. The decibel (dB) value caused confusion and was removed. Isolation Forest worked better than Kalman Filter when tested on collected data, because it was trained to detect outliers. However, during real-time testing, Kalman Filter performed better, as it reacted faster and more smoothly to live data without needing training. Both methods together gave strong results and helped reduce false alarms.

The project also included a web dashboard with two pages. The first page shows real-time sensor values and alerts when an abnormal condition is detected. The second page displays live charts for sound features (RMS and ZCR) filtered by both Kalman Filter and Isolation Forest. These features help hospital staff quickly understand what is happening in the room.

The robot was programmed to move near the patient and collect better sound data. This reduced background noise and improved the quality of readings. The robot also avoided obstacles using distance sensors. All movement was done using simple commands like move forward, rotate, and shift sideways.

6.2. Future Work

This project showed that it is possible to monitor patients using sound sensors, filtering methods, and a mobile robot. However, there are many areas that can be improved or added in the future:

Add vision features: The robot camera can be used to detect face colour, movement, or breathing rate. This would help monitor patients without needing extra sensors.

Use YOLO for object or face detection: In the future, object detection using YOLO can be added once all camera libraries are working on the robot.

Improve robot navigation: Right now, the robot uses simple movements. Advanced algorithms like A* or DWA can help it move more smartly in hospital rooms and avoid obstacles

Bibliography

[1] L. Tarassenko, M. Villarroel, A. Guazzi, J. Jorge, D. A. Clifton, and C. Pugh, “Non-contact video-based vital sign monitoring using ambient light and auto-regressive models,” *Physiol. Meas.*, vol. 35, no. 5, pp. 807–831, May 2014, doi: 10.1088/0967-3334/35/5/807.

[2] K. S. Naik and E. Sudarshan, “SMART HEALTHCARE MONITORING SYSTEM USING RASPBERRY Pi ON IoT PLATFORM,” vol. 14, no. 4, 2019.

[3] H.-W. Huang *et al.*, “Agile mobile robotic platform for contactless vital signs monitoring”, Accessed: Nov. 29, 2024. [Online]. Available: https://www.authorea.com/doi/full/10.36227/techrxiv.12811982.v1?commit=5b04299d96c6f4489a4d6b28eaa28e792bd86a97

[4] H.-W. Huang *et al.*, “Agile mobile robotic platform for contactless vital signs monitoring”, Accessed: Nov. 29, 2024. [Online]. Available: https://www.authorea.com/doi/full/10.36227/techrxiv.12811982.v1?commit=5b04299d96c6f4489a4d6b28eaa28e792bd86a97

[5] C. Mireles, M. Sanchez, D. Cruz-Ortiz, I. Salgado, and I. Chairez, “Home-care nursing controlled mobile robot with vital signal monitoring,” *Med. Biol. Eng. Comput.*, vol. 61, no. 2, pp. 399–420, Feb. 2023, doi: 10.1007/s11517-022-02712-y.

[6] L. Tarassenko, M. Villarroel, A. Guazzi, J. Jorge, D. A. Clifton, and C. Pugh, “Non-contact video-based vital sign monitoring using ambient light and auto-regressive models,” *Physiol. Meas.*, vol. 35, no. 5, p. 807, Mar. 2014, doi: 10.1088/0967-3334/35/5/807.

[7] K. S. Naik and E. Sudarshan, “SMART HEALTHCARE MONITORING SYSTEM USING RASPBERRY Pi ON IoT PLATFORM,” vol. 14, no. 4, 2019.

[8] S. L. Rohit and B. V. Tank, “Iot Based Health Monitoring System Using Raspberry PI - Review,” in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, Coimbatore: IEEE, Apr. 2018, pp. 997–1002. doi: 10.1109/ICICCT.2018.8472957.

[9] “Raspberry Pi OS - Raspberry Pi Documentation.” Accessed: Nov. 30, 2024. [Online]. Available: https://www.raspberrypi.com/documentation/computers/os.html

[10] “Using motors | Physical Computing with Python | Python | Coding projects for kids and teens.” Accessed: Nov. 30, 2024. [Online]. Available: https://projects.raspberrypi.org/en/projects/physical-computing/14

[11] Hiwonder, *Hiwonder-docs/MasterPi*. (Nov. 06, 2024). Python. Accessed: Nov. 30, 2024. [Online]. Available: https://github.com/Hiwonder-docs/MasterPi

[12] “pigpio library.” Accessed: Nov. 30, 2024. [Online]. Available: https://abyz.me.uk/rpi/pigpio/

[13] “A* Search Algorithm,” GeeksforGeeks. Accessed: Dec. 07, 2024. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/

[14] D. Fox, W. Burgard, and S. Thrun, “The Dynamic Window Approach to Collision Avoidance,” *Robot. Autom. Mag. IEEE*, vol. 4, pp. 23–33, Apr. 1997, doi: 10.1109/100.580977.

[15] J. Yao, “RRT algorithm learning and optimization,” *Appl. Comput. Eng.*, vol. 53, pp. 296–302, Mar. 2024, doi: 10.54254/2755-2721/53/20241614.

[16] “OBJECT DETECTION: YOLO VS FASTER R-CNN,” *Int. Res. J. Mod. Eng. Technol. Sci.*.

[17] “YOLOv8 vs Mask R-CNN: In-depth Analysis and Comparison,” Labelvisor. Accessed: Dec. 07, 2024. [Online]. Available: https://www.labelvisor.com/yolov8-vs-mask-r-cnn-in-depth-analysis-and-comparison/

[18] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” Jan. 24, 2018, *arXiv*: arXiv:1703.06870. doi: 10.48550/arXiv.1703.06870.

[19] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” vol. 9905, 2016, pp. 21–37. doi: 10.1007/978-3-319-46448-0_2.

[20] R. Sapkota, D. Ahmed, and M. Karkee, “Comparing YOLOv8 and Mask R-CNN for instance segmentation in complex orchard environments,” *Artif. Intell. Agric.*, vol. 13, pp. 84–99, Sep. 2024, doi: 10.1016/j.aiia.2024.07.001.

[21] L. Kleeman, “Understanding and Applying Kalman Filtering”.

[22] M. Laaraiedh, “Implementation of Kalman Filter with Python Language”.

[23] M. Sun, R. Hodgskin-Brown, M. E. Davies, I. K. Proudler, and J. R. Hopgood, “Adaptive Kernel Kalman Filter for Magnetic Anomaly Detection-based Metallic Target Tracking,” in *2023 Sensor Signal Processing for Defence Conference (SSPD)*, Edinburgh, United Kingdom: IEEE, Sep. 2023, pp. 1–5. doi: 10.1109/SSPD57945.2023.10256968.

[24] “IsolationForest,” scikit-learn. Accessed: Apr. 05, 2025. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[25] “Isolation Forest Guide: Explanation and Python Implementation.” Accessed: Apr. 05, 2025. [Online]. Available: https://www.datacamp.com/tutorial/isolation-forest

[26] “https://sail.usc.edu/~lgoldste/Ling582/Week%203/Moving%20Average%20Filters.pdf.” Accessed: Dec. 07, 2024. [Online]. Available: https://sail.usc.edu/~lgoldste/Ling582/Week%203/Moving%20Average%20Filters.pdf

[27] PonDad, *PonDad/RaspberryPi4-Unsupervised-Real-Time-Anomaly-Detection*. (Aug. 15, 2024). Python. Accessed: Dec. 07, 2024. [Online]. Available: https://github.com/PonDad/RaspberryPi4-Unsupervised-Real-Time-Anomaly-Detection

[28] “Safety Standards in Healthcare Robotics,” UL Solutions. Accessed: Dec. 07, 2024. [Online]. Available: https://www.ul.com/insights/safety-standards-healthcare-robotics

[29] T. A. Wilbon, “Application of Risk Management Principles for Medical Devices”.

[30] G.-Z. Yang *et al.*, “Medical robotics—Regulatory, ethical, and legal considerations for increasing levels of autonomy,” *Sci. Robot.*, vol. 2, no. 4, p. eaam8638, Mar. 2017, doi: 10.1126/scirobotics.aam8638.

[31] Gaston, *gastonduault/Baggie-Robots*. (Nov. 18, 2024). Python. Accessed: Dec. 07, 2024. [Online]. Available: https://github.com/gastonduault/Baggie-Robots

[32] “RMS (root mean square) in audio,” Ampedstudio. Accessed: Apr. 16, 2025. [Online]. Available: https://ampedstudio.com/rms-in-audio/

[33] “Zero crossing rate in python - DataSpoof.” Accessed: Apr. 16, 2025. [Online]. Available: https://www.dataspoof.info/post/zero-crossing-rate-in-python/

[34] “Root mean square normalization in Python,” 🧠 SuperKogito. Accessed: Apr. 16, 2025. [Online]. Available: https://superkogito.github.io/blog/2020/04/30/rms_normalization.html

[35] “Hospital Ambient Noise Dataset.” Accessed: Apr. 14, 2025. [Online]. Available: https://www.kaggle.com/datasets/nafin59/hospital-ambient-noise
[36] “https://github.com/iiscleap/Coswara-Data/tree/master.” Accessed: Apr. 14, 2025. [Online]. Available: https://github.com/iiscleap/Coswara-Data/tree/master

Appendices
Appendix A: SQL Schema and Database Structure

Appendix A.1: SQL Schema of Datasets

```sql
-- Drop existing tables to avoid conflicts
DROP TABLE IF EXISTS staff_actions, robot_actions, alerts, anomalies,
anomaly_comparison, robot_readings, smart_room_readings, staff, patients,
beds, users CASCADE;

-- Beds Table: Stores fixed bed locations
CREATE TABLE beds (
    bed_number SERIAL PRIMARY KEY,
    bed_x INT NOT NULL,
    bed_y INT NOT NULL
);

-- Patients Table: Stores patient details and links to a bed
CREATE TABLE patients (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    DOB DATE NOT NULL,
    bed_number INT UNIQUE REFERENCES beds(bed_number) ON DELETE
SET NULL
);

-- Staff Table: Stores hospital staff information
CREATE TABLE staff (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    role VARCHAR(50) CHECK (role IN ('Doctor', 'Nurse', 'Admin')) NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password TEXT NOT NULL
);

-- Smart Room Readings Table: Monitors room environmental conditions
CREATE TABLE smart_room_readings (
    id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(id) ON DELETE CASCADE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    temperature FLOAT,
    light_level FLOAT,
    sound_level FLOAT
);

-- Robot Sensor Readings Table: Collects patient health vitals
CREATE TABLE robot_readings (
    id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(id) ON DELETE CASCADE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    temperature FLOAT,
    heart_rate INT,
    oxygen_level INT
);

-- Anomalies Table: Tracks environmental anomalies
CREATE TABLE anomalies (
    id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(id) ON DELETE CASCADE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    sensor_type VARCHAR(50) CHECK (sensor_type IN ('temperature', 'light',
'sound', 'heart_rate', 'oxygen_level')) NOT NULL,
    anomaly_value FLOAT NOT NULL,
    detection_method VARCHAR(50) CHECK (detection_method IN ('Kalman
Filter', 'Isolation Forest')) NOT NULL
);

-- Alerts Table: Stores alert messages
CREATE TABLE alerts (
    id SERIAL PRIMARY KEY,
    patient_id INT REFERENCES patients(id) ON DELETE CASCADE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    message TEXT NOT NULL,
    severity VARCHAR(20) CHECK (severity IN ('low', 'medium', 'high')) NOT
NULL
);

-- Robot Actions Table: Logs robot movements and statuses
CREATE TABLE robot_actions (
    id SERIAL PRIMARY KEY,
    robot_id VARCHAR(50),
    patient_id INT REFERENCES patients(id),
    action VARCHAR(100),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(50)
);

-- Anomaly Comparison Table: For comparing sound readings and detection
methods
CREATE TABLE anomaly_comparison (
    id SERIAL PRIMARY KEY,
    source VARCHAR(50) CHECK (source IN ('CrowPi', 'Robot_No_AI',
'Robot_Kalman', 'Robot_IForest')),
    sound_level FLOAT,
    is_anomaly BOOLEAN,
    detection_method VARCHAR(50) CHECK (detection_method IN ('None',
'Kalman Filter', 'Isolation Forest')),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Appendix B: Code Listings

Appendix B.1: data_filtering.py(work with just db sound level)

```
# data_filtering.py
import os
import time
import psycopg2
import numpy as np
from datetime import datetime
from sklearn.ensemble import IsolationForest
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

# Connect to PostgreSQL database
conn = psycopg2.connect(
    dbname=os.getenv("DB_DATABASE"),
    user=os.getenv("DB_USER"),
    password=os.getenv("DB_PASSWORD"),
    host=os.getenv("DB_HOST"),
    port=os.getenv("DB_PORT")
)
cursor = conn.cursor()

# -------------------------------
# Kalman Filter Class
# -------------------------------
class KalmanFilter:
    def __init__(self, F, H, Q, R, P, x0):
        self.F = F  # State transition
        self.H = H  # Measurement matrix
        self.Q = Q  # Process noise
        self.R = R  # Measurement noise
        self.P = P  # Error covariance
        self.x = x0  # Initial state

    def predict(self):
        # Predict next state
        self.x = np.dot(self.F, self.x)
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q

    def update(self, z):
        # Update with new measurement
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
        y = z - np.dot(self.H, self.x)
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.F.shape[1])
        self.P = np.dot(I - np.dot(K, self.H), self.P)

# Function to create one Kalman filter
```

```
def initialize_kalman_filter():
    F = np.array([[1]])
    H = np.array([[1]])
    Q = np.array([[0.01]])
    R = np.array([[0.1]])
    P = np.array([[1]])
    x0 = np.array([[0]])
    return KalmanFilter(F, H, Q, R, P, x0)

# ------------------------------
# Sensor Setup
# ------------------------------
sensor_config = {
    "light": {
        "column": "light_level",
        "table": "smart_room_readings",
        "min": 10,
        "max": 25
    }
    # You can add more sensors here if needed
}

# ------------------------------
# Train Isolation Forest
# ------------------------------
def train_model(column, table, min_val, max_val):
    # Get latest data from database
    cursor.execute(f"""
        SELECT {column} FROM {table}
        WHERE {column} IS NOT NULL AND {column} BETWEEN %s AND %s
        ORDER BY timestamp DESC LIMIT 100
    """, (min_val, max_val))
    rows = cursor.fetchall()
    values = np.array([r[0] for r in rows]).reshape(-1, 1)

    if len(values) >= 5:
        model = IsolationForest(contamination=0.1)
        model.fit(values)
        print(f" Model trained with {len(values)} values.")
        return model
    else:
        print(f" Not enough data to train model for column '{column}'. Only {len(values)} values
found.")
        return None

# ------------------------------
# Run Detection
# ------------------------------
def run_detection(sensor_type, column, table, model, kf):
    # Get latest one value from the sensor table
    cursor.execute(f"""
        SELECT id, {column}, timestamp FROM {table}
        WHERE {column} IS NOT NULL
        ORDER BY timestamp DESC LIMIT 1
    """)
    row = cursor.fetchone()

    if row:
        record_id, value, timestamp = row
        value = float(value)
        # ---- Isolation Forest check ----
        pred = model.predict([[value]])
        if pred[0] == -1:
            print(f" Isolation Forest: Anomaly in {sensor_type} = {value}")
            cursor.execute("""
                INSERT INTO anomalies(sensor_type, anomaly_value, detection_method, timestamp)
                VALUES (%s, %s, %s, %s)
            """, (sensor_type, value, "Isolation Forest", timestamp))
            conn.commit()
        else:
            print(f" Isolation Forest: Normal {sensor_type} = {value}")

        # ---- Kalman Filter check ----
        kf.predict()
        kf.update(np.array([[value]]))
        estimate = kf.x[0][0]
        residual = abs(estimate - value)

        if residual > 3:
            print(f" Kalman Filter: Anomaly in {sensor_type} = {value} (residual {residual})")
            cursor.execute("""
                INSERT INTO anomalies(sensor_type, anomaly_value, detection_method, timestamp)
                VALUES (%s, %s, %s, %s)
            """, (sensor_type, value, "Kalman Filter", timestamp))
            conn.commit()
        else:
            print(f" Kalman Filter: Normal {sensor_type} = {value} (residual {residual})")

# --------------------------------
# Main Code
# --------------------------------

# Create one Kalman filter for each sensor
kalman_filters = {
    sensor: initialize_kalman_filter()
    for sensor in sensor_config
}

# Go through each sensor and run detection
print("Real-time sensor monitoring started (updates every 5 seconds)...\n")

try:
    while True:
        for sensor, cfg in sensor_config.items():
            print(f"\n Checking sensor: {sensor}")
            model = train_model(cfg["column"], cfg["table"], cfg["min"], cfg["max"])
            if model is not None:
                run_detection(sensor, cfg["column"], cfg["table"], model, kalman_filters[sensor])
            else:
                print(f" Not enough data to train model for {sensor}. Skipping.\n")

        time.sleep(5)  # Wait 5 seconds before next reading

except KeyboardInterrupt:
    print("\n Monitoring stopped by user.")

finally:
    # Close database connection
    cursor.close()
    conn.close()
```

Appendix B.2

```
import os
import time
import psycopg2
import numpy as np
import pandas as pd
from datetime import datetime
from sklearn.ensemble import IsolationForest
from dotenv import load_dotenv

load_dotenv()
print("Running data_filtering.py")

# Connect to PostgreSQL database
try:
    conn = psycopg2.connect(
        dbname=os.getenv("DB_DATABASE"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"),
        host=os.getenv("DB_HOST"),
        port=os.getenv("DB_PORT")
    )
    cursor = conn.cursor()
except Exception as e:
    print("Database connection error:", e)
    exit(1)

# Kalman Filter Class
class KalmanFilter:
    def __init__(self, F, H, Q, R, P, x0):
        self.F = F
        self.H = H
        self.Q = Q
        self.R = R
        self.P = P
        self.x = x0

    def predict(self):
        self.x = np.dot(self.F, self.x)
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q

    def update(self, z):
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
        y = z - np.dot(self.H, self.x)
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.F.shape[1])
        self.P = np.dot(I - np.dot(K, self.H), self.P)

def initialize_kalman_filter():
    F = np.array([[1]])
    H = np.array([[1]])
    Q = np.array([[0.01]])
    R = np.array([[0.1]])
    P = np.array([[1]])
    x0 = np.array([[0]])
    return KalmanFilter(F, H, Q, R, P, x0)
```

```python
# Sensor configuration
sensor_config = {
    "light": {
        "columns": ["light_level"],
        "table": "smart_room_readings",
        "min_values": [10],
        "max_values": [25],
        "training_source": "postgres"
    },
    "sound": {
        "columns": ["db", "rms", "zcr"],
        "csv_file": "training_noise_data_audio_features.csv",
        "training_source": "csv"
    }
}

def train_model(sensor_name, cfg):
    try:
        if cfg["training_source"] == "csv":
            df = pd.read_csv(cfg["csv_file"])
            df = df[df['label'] == 'normal']
            values = df[cfg["columns"]].values
        else:
            columns = cfg["columns"]
            table = cfg["table"]
            min_vals = cfg["min_values"]
            max_vals = cfg["max_values"]
            conditions = " AND ".join([
                f"{col} BETWEEN {min_val} AND {max_val}"
                for col, min_val, max_val in zip(columns, min_vals, max_vals)
            ])
            query = f"""
                SELECT {', '.join(columns)} FROM {table}
                WHERE {conditions}
                ORDER BY timestamp DESC LIMIT 100
            """
            cursor.execute(query)
            rows = cursor.fetchall()
            values = np.array(rows)

        if len(values) >= 5:
            model = IsolationForest(contamination=0.1)
            model.fit(values)
            print(f"[{sensor_name}] Model trained with {len(values)} samples.")
            return model
        else:
            print(f"[{sensor_name}] Not enough training data.")
            return None
    except Exception as e:
        print(f"[{sensor_name}] Training error:", e)
        return None

def run_detection(sensor_type, columns, table, model, kf_list):
    try:
        cursor.execute(f"""
            SELECT id, {', '.join(columns)}, timestamp FROM {table}
            WHERE {columns[0]} IS NOT NULL
            ORDER BY timestamp DESC LIMIT 1
        """)
        row = cursor.fetchone()
        if not row:
            return

        record_id, *values, timestamp = row
        values = [float(v) for v in values]

        # Isolation Forest
        pred = model.predict([values])
        if pred[0] == -1:
            print(f"Isolation Forest: Anomaly in {sensor_type} = {values}")
            cursor.execute("""
                INSERT INTO anomalies(sensor_type, anomaly_value, detection_method, timestamp)
                VALUES (%s, %s, %s, %s)
            """, (sensor_type, str(values), "Isolation Forest", timestamp))
            conn.commit()

        # Kalman Filter for each feature
        for i, val in enumerate(values):
            kf_list[i].predict()
            kf_list[i].update(np.array([[val]]))
            estimate = kf_list[i].x[0][0]
            residual = abs(estimate - val)

            if residual > 3:
                print(f"Kalman Filter: Anomaly in {sensor_type} [{columns[i]}] = {val} (residual =
{residual})")
                cursor.execute("""
                    INSERT INTO anomalies(sensor_type, anomaly_value, detection_method, timestamp)
                    VALUES (%s, %s, %s, %s)
                """, (f"{sensor_type}_{columns[i]}", val, "Kalman Filter", timestamp))
                conn.commit()
            else:
                print(f"Kalman Filter: Normal {sensor_type} [{columns[i]}] = {val} (residual = {residual})")

    except Exception as e:
        print("Error during detection:", e)

# Initialize Kalman Filters
kalman_filters = {sensor: [initialize_kalman_filter() for _ in cfg["columns"]] for sensor, cfg in
sensor_config.items()}

print("Real-time sensor monitoring started (updates every 10 seconds)...\n")

try:
    while True:
        for sensor, cfg in sensor_config.items():
            print(f"\nChecking sensor: {sensor}")
            model = train_model(sensor, cfg)
            if model:
                run_detection(sensor, cfg["columns"], cfg.get("table", "smart_room_readings"), model,
kalman_filters[sensor])
        time.sleep(10)

except KeyboardInterrupt:
    print("\nMonitoring stopped by user.")

except Exception as e:
    print("Unexpected error:", e)
```

```
finally:
    cursor.close()
    conn.close()
    print("Database connection closed.")
```

Appendix B.3: sensors.js

```
// sensors.js
// This file connects to MQTT, listens to sensor topics, and saves data to the database

const mqtt = require('mqtt');

module.exports = (pool) => {
    const brokerUrl = process.env.MQTT_BROKER || 'mqtt://192.168.1.26';
    const mqttClient = mqtt.connect(brokerUrl);

    const topics = ['smartroom/light', 'smartroom/sound'];

    mqttClient.on('connect', () => {
        console.log('Connected to MQTT broker');
        mqttClient.subscribe(topics, (err) => {
            if (err) {
                console.error('Error subscribing to topics:', err.message);
            } else {
                console.log('Subscribed to sensor topics');
            }
        });
    });

    mqttClient.on('message', async (topic, message) => {
        console.log(`Received MQTT message: ${topic} -> ${message.toString()}`);
        try {
            const data = JSON.parse(message.toString());

            if (!data.timestamp) {
                console.warn(`Missing timestamp in topic ${topic}`);
                return;
            }

            await handleSensorData(topic, data, pool);
        } catch (err) {
            console.error('Error processing MQTT message:', err.message);
        }
    });
};

// Save sensor data to the database
async function handleSensorData(topic, data, pool) {
    const { timestamp } = data;

    try {
        if (topic === 'smartroom/light') {
            const { value } = data;
            if (value === undefined) {
                console.warn(`Missing light value`);
                return;
            }

            await pool.query(
                `INSERT INTO smart_room_readings(light_level, timestamp) VALUES($1, $2)`,
                [value, timestamp]
            );
            console.log(`light_level saved: ${value} at ${timestamp}`);

        } else if (topic === 'smartroom/sound') {
            const { db, rms, zcr } = data;
            if (db === undefined || rms === undefined || zcr === undefined) {
                console.warn(`Missing sound features: db=${db}, rms=${rms}, zcr=${zcr}`);
                return;
            }

            await pool.query(
                `INSERT INTO smart_room_readings(db, rms, zcr, timestamp) VALUES($1, $2, $3, $4)`,
                [db, rms, zcr, timestamp]
            );
            console.log(`Sound features saved: db=${db}, rms=${rms}, zcr=${zcr} at ${timestamp}`);
        } else {
            console.warn(`Unknown topic received: ${topic}`);
        }
    } catch (err) {
        console.error(`Database insert error for ${topic}:`, err.message);
    }
}
```

Appendix B.4:

```
# This script reads both light and sound sensors and sends them via MQTT every 10 seconds

import paho.mqtt.client as mqtt
import json
import time
from time import strftime
import numpy as np
import sounddevice as sd
import smbus

# --- Light Sensor Class ---
bus = smbus.SMBus(1)
class LightSensor:
    def __init__(self):
        self.DEVICE = 0x5c
        self.ONE_TIME_HIGH_RES_MODE_1 = 0x20

    def convertToNumber(self, data):
        return ((data[1] + (256 * data[0])) / 1.2)

    def readLight(self):
        data = bus.read_i2c_block_data(self.DEVICE, self.ONE_TIME_HIGH_RES_MODE_1)
        return self.convertToNumber(data)

# --- Sound Feature Extraction ---
def extract_sound_features(duration=1, fs=44100):
    recording = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='float64')
    sd.wait()

    # Flatten the array
    signal = recording.flatten()

    # RMS and dB
    rms = np.sqrt(np.mean(signal**2))
    db = 20 * np.log10(rms + 1e-6) + 100

    # Zero Crossing Rate
    zcr = ((signal[:-1] * signal[1:]) < 0).sum() / len(signal)

    return round(db, 2), round(rms, 5), round(zcr, 5)

# --- MQTT Setup ---
MQTT_BROKER = "192.168.1.26"
MQTT_PORT = 1883
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.connect(MQTT_BROKER, MQTT_PORT, 60)
print("Connected to MQTT server at", MQTT_BROKER)

light_sensor = LightSensor()

# --- Main loop ---
try:
    while True:
        timestamp = strftime("%Y-%m-%d %H:%M:%S")

        # Read sensors
        light_value = light_sensor.readLight()
        db, rms, zcr = extract_sound_features()

        # Publish light
        light_msg = {
            "sensor": "light",
            "value": light_value,
            "timestamp": timestamp
        }
        client.publish("smartroom/light", json.dumps(light_msg))
        print(f"Light sent: {light_value:.2f} lx")

        # Publish sound with all features
        sound_msg = {
            "sensor": "sound",
            "db": db,
            "rms": rms,
            "zcr": zcr,
            "timestamp": timestamp
        }
        client.publish("smartroom/sound", json.dumps(sound_msg))
        print(f"Sound sent: db={db:.2f}, rms={rms:.5f}, zcr={zcr:.5f}")

        time.sleep(10)

except KeyboardInterrupt:
    print("Stopped sending sensor data.")
```

Appendix B.5:

```
import os
import time
import psycopg2
import numpy as np
import pandas as pd
from datetime import datetime
from sklearn.ensemble import IsolationForest
from dotenv import load_dotenv

load_dotenv()
print("data_filtering started")
patient_id = 10

# Connect to database
try:
    conn = psycopg2.connect(
        dbname=os.getenv("DB_DATABASE"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"),
        host=os.getenv("DB_HOST"),
        port=os.getenv("DB_PORT")
    )
    cursor = conn.cursor()
except Exception as e:
    print("Database connection error:", e)
    exit(1)

# Kalman Filter class
class KalmanFilter:
    def __init__(self, F, H, Q, R, P, x0):
        self.F = F
        self.H = H
        self.Q = Q
        self.R = R
        self.P = P
        self.x = x0

    def predict(self):
        self.x = np.dot(self.F, self.x)
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q

    def update(self, z):
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
        y = z - np.dot(self.H, self.x)
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.F.shape[1])
        self.P = np.dot(I - np.dot(K, self.H), self.P)

# Initialize Kalman Filter
def initialize_kalman_filter(sensor_type, feature):
    if sensor_type == "sound":
        if feature == "rms":
            x0 = np.array([[0.003]])
        elif feature == "zcr":
            x0 = np.array([[0.01]])
        else:
            x0 = np.array([[0]])
    elif sensor_type == "light" and feature == "light_level":
        x0 = np.array([[12]])
    else:
        x0 = np.array([[0]])

    return KalmanFilter(
        F=np.array([[1]]),
        H=np.array([[1]]),
        Q=np.array([[0.01]]),
        R=np.array([[0.1]]),
        P=np.array([[1]]),
        x0=x0
    )

# Sensor configuration
sensor_config = {
    "light": {
        "columns": ["light_level"],
        "csv_file": "trainingData.csv",
        "training_source": "csv",
        "min_values": [0],
        "max_values": [25]
    },
    "sound": {
        "columns": ["rms", "zcr"],
        "csv_file": "trainingData.csv",
        "training_source": "csv",
        "min_values": [0.01, 0.003],
        "max_values": [0.055, 0.02]
    }

}

# Train Isolation Forest
def train_model(sensor_name, cfg):
    try:
        if cfg["training_source"] == "csv":
            df = pd.read_csv(cfg["csv_file"])
            df = df[df['label'] == 'normal']

            if sensor_name == "sound":
                df = df[
                    (df["rms"] >= 0.001) & (df["rms"] <= 0.03) &
                    (df["zcr"] >= 0) & (df["zcr"] <= 0.02)
                ]
            elif sensor_name == "light":
                df = df[(df["light_level"] >= 0) & (df["light_level"] <= 25)]

            values = df[cfg["columns"]].values
        else:
            conditions = " AND ".join([
                f"{col} BETWEEN {min_val} AND {max_val}"
                for col, min_val, max_val in zip(cfg["columns"], cfg["min_values"],
cfg["max_values"])
            ])
            query = f"""
                SELECT {', '.join(cfg["columns"])} FROM {cfg["table"]}
                WHERE {conditions}
                ORDER BY timestamp DESC LIMIT 100
            """
            cursor.execute(query)
            values = np.array(cursor.fetchall())

        if len(values) >= 3:
            model = IsolationForest(contamination=0.1)
            model.fit(values)
            return model
        else:
            return None

    except Exception as e:
        print(f"[{sensor_name}] Training error:", e)
        return None

# Run detection
def run_detection(sensor_type, columns, table, model, kf_list):
    try:
        cursor.execute(f"""
            SELECT id, patient_id, {', '.join(columns)}, timestamp FROM {table}
            WHERE {columns[0]} IS NOT NULL
            ORDER BY timestamp DESC LIMIT 1
        """)
        row = cursor.fetchone()
        if not row:
            return

        record_id, patient_id, *values, timestamp = row

        cursor.execute("""
            SELECT COUNT(*) FROM anomalies
            WHERE timestamp = %s AND sensor_type = %s
        """, (timestamp, sensor_type))
        if cursor.fetchone()[0] > 0:
            return

        values = [float(v) for v in values]
        print(f"\n[INFO] {sensor_type} at {timestamp}:")
        for col, val in zip(columns, values):
            print(f"  {col} = {val}")
        # Isolation Forest detection
        pred = model.predict([values])
        if pred[0] == -1:
            abnormal_features = []
            for i, val in enumerate(values):
                min_val = sensor_config[sensor_type]["min_values"][i]
                max_val = sensor_config[sensor_type]["max_values"][i]
                if val < min_val or val > max_val:
                    abnormal_features.append(f"{columns[i]} = {val} (out of [{min_val},
{max_val}])")

            if abnormal_features:
                print(f"[ALERT] Isolation Forest abnormal {sensor_type}: " + ";
".join(abnormal_features))
                cursor.execute("""
                    INSERT INTO anomalies(patient_id, sensor_type, db, rms, zcr,
detection_method, timestamp)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
                """, (patient_id, sensor_type, None,
                      values[0] if len(values) > 0 else None,
                      values[1] if len(values) > 1 else None,
                      "Isolation Forest", timestamp))
                conn.commit()

        # Kalman Filter detection
        for i, val in enumerate(values):
            kf_list[i].predict()
            kf_list[i].update(np.array([[val]]))
            residual = abs(kf_list[i].x[0][0] - val)

            # Set strict thresholds for live detection
            if columns[i] == "rms":
                threshold = 0.004
            elif columns[i] == "zcr":
                threshold = 0.003
            else:
                threshold = 1

            if residual > threshold:
                print(f"[ALERT] Kalman Filter abnormal {sensor_type} [{columns[i]}] =
{val} (residual = {residual:.5f})")
                cursor.execute("""
                    INSERT INTO anomalies(patient_id, sensor_type, db, rms, zcr,
detection_method, timestamp)
                    VALUES (%s, %s, %s, %s, %s, %s, %s)
                """, (patient_id, sensor_type, None,
                      values[0] if len(values) > 0 else None,
                      values[1] if len(values) > 1 else None,
                      "Kalman Filter", timestamp))
            conn.commit()

    except Exception as e:
        print("Detection error:", e)
        conn.rollback()

# Initialize Kalman filters
kalman_filters = {
    sensor: [initialize_kalman_filter(sensor, feature) for feature in cfg["columns"]]
    for sensor, cfg in sensor_config.items()
}

print("Monitoring started...")

try:
    for sensor, cfg in sensor_config.items():
        model = train_model(sensor, cfg)
        if model:
            run_detection(sensor, cfg["columns"], cfg.get("table",
"smart_room_readings"), model, kalman_filters[sensor])
except KeyboardInterrupt:
    print("Stopped by user.")
finally:
    cursor.close()
    conn.close()
    print("Database closed.")
```

Appendix B.7 : full code of server.js:

```
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const { Pool } = require('pg');
const path = require('path');
const app = express();
const port = 2021;

app.use(bodyParser.json());
app.use(express.static('public'));

// Connect to PostgreSQL
const pool = new Pool({
    user: process.env.DB_USER,
    host: process.env.DB_HOST,
    database: process.env.DB_DATABASE,
    password: process.env.DB_PASSWORD,
    port: process.env.DB_PORT
});

pool.connect((err, client, release) => {
    if (err) console.error('Database error:', err.stack);
    else {
        console.log('Connected to database');
        release();
    }
});

// Load sensor handling module
require('./sensors/sensors')(pool);

// === Light Status API ===
app.get('/api/light', async (req, res) => {
    try {
        const result = await pool.query(`
            SELECT light_level, timestamp
            FROM smart_room_readings
            WHERE light_level IS NOT NULL
            ORDER BY timestamp DESC
            LIMIT 1
        `);

        const latest = result.rows[0];
        if (!latest) return res.json({ light_level: null, methods: [] });

        const anomaly = await pool.query(`
            SELECT DISTINCT detection_method
            FROM anomalies
            WHERE sensor_type = 'light' AND timestamp = $1
        `, [latest.timestamp]);

        res.json({
            light_level: latest.light_level,
            methods: anomaly.rows.map(r => r.detection_method)
        });
    } catch (err) {
        console.error('Error /api/light:', err);
        res.status(500).json({ error: 'DB error' });
    }
});

// === Sound Status API ===
app.get('/api/sound', async (req, res) => {
    try {
        const result = await pool.query(`
            SELECT rms, zcr, timestamp
            FROM smart_room_readings
            WHERE rms IS NOT NULL AND zcr IS NOT NULL
            ORDER BY timestamp DESC
            LIMIT 1
        `);
        const latest = result.rows[0];
        if (!latest) return res.json({ rms: null, zcr: null, methods: [] });

        const anomaly = await pool.query(`
            SELECT DISTINCT detection_method
            FROM anomalies
            WHERE sensor_type = 'sound' AND timestamp = $1
        `, [latest.timestamp]);

        res.json({
            rms: latest.rms,
            zcr: latest.zcr,
            methods: anomaly.rows.map(r => r.detection_method)
        });
    } catch (err) {
        console.error('Error /api/sound:', err);
        res.status(500).json({ error: 'DB error' });
    }
});

// === Sound Charts API ===
app.get('/api/sound-charts', async (req, res) => {
  try {
    const result = await pool.query(`
      SELECT timestamp, rms, zcr
      FROM smart_room_readings
      WHERE rms IS NOT NULL AND zcr IS NOT NULL
      ORDER BY timestamp DESC
      LIMIT 20
    `);

    const reversed = result.rows.reverse();

    const kalman_rms = [], kalman_zcr = [], iso_rms = [], iso_zcr = [];

    reversed.forEach((row, index) => {
      const label = new Date(row.timestamp).toLocaleTimeString();
      kalman_rms.push({ label, value: parseFloat(row.rms) });
      kalman_zcr.push({ label, value: parseFloat(row.zcr) });
      iso_rms.push({ label, value: parseFloat(row.rms) });
      iso_zcr.push({ label, value: parseFloat(row.zcr) });
    });

    res.json({ kalman_rms, kalman_zcr, iso_rms, iso_zcr });

  } catch (err) {
    console.error('Error /api/sound-charts:', err);
    res.status(500).json({ error: 'Chart data error' });
  }
```

```
});


// Start server
app.listen(port, () => {
   console.log(`Server running at http://localhost:${port}`);
});
```

Appendix B.8: full code of server.js:

```
const mqtt = require('mqtt');
const path = require('path');
const { spawn } = require('child_process');
const patient_id = 10;

module.exports = (pool) => {
    const brokerUrl = process.env.MQTT_BROKER || 'mqtt://localhost';
    const mqttClient = mqtt.connect(brokerUrl);
    const topics = ['smartroom/light', 'smartroom/sound'];

    let lastProcessedTimestamp = null;

    mqttClient.on('connect', () => {
        console.log('[MQTT] Connected to broker');
        mqttClient.subscribe(topics, () => {
            console.log('[MQTT] Subscribed to topics:', topics.join(',
'));
        });
    });

    mqttClient.on('message', async (topic, message) => {
        try {
            const data = JSON.parse(message.toString());
            if (!data.timestamp) return;

            const { timestamp } = data;

            if (topic === 'smartroom/light') {
                const { value } = data;
                if (value !== undefined) {
                    await pool.query(
                        'INSERT INTO smart_room_readings(patient_id,
light_level, timestamp) VALUES ($1, $2, $3)',
                        [patient_id, value, timestamp]
                    );
                }

            } else if (topic === 'smartroom/sound') {
                const { rms, zcr } = data;
                if (rms !== undefined && zcr !== undefined) {
                    await pool.query(
                        'INSERT INTO smart_room_readings(patient_id, rms,
zcr, timestamp) VALUES ($1, $2, $3, $4)',
                        [patient_id, rms, zcr, timestamp]
                    );
                }
            }

            // Run filtering only for new timestamp
            if (timestamp !== lastProcessedTimestamp) {
                lastProcessedTimestamp = timestamp;

                const scriptPath = path.join(__dirname,
'../data_filtering.py');
                const py = spawn('python', ['-u', scriptPath]);

                py.stdout.on('data', (data) => {
                    const text = data.toString();
                    if (text.includes('[INFO]') ||
text.includes('[ALERT]')) {
                        console.log(text.trim());
                    }
                });

                py.stderr.on('data', (data) => {
                    console.error(`Python Error: ${data}`);
                });
            }

        } catch (err) {
            console.error('MQTT message error:', err.message);
        }
    });
};
```

Appendix B.9:index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Smart Hospital Dashboard</title>
  <link rel="stylesheet" href="style.css" />
  <script src="dashboard.js" defer></script>
</head>
<body>
    <header>
        <h1>Smart Hospital System</h1>
    </header>

    <main>
        <section id="patient-info">
            <h2>Patient Vitals</h2>
            <table>
                <tbody>
                <tr><td>Patient Name:</td><td>John Doe</td></tr>
                <tr><td>Temperature:</td><td>36.8 °C</td></tr>
                <tr><td>Heart Rate:</td><td>78 bpm</td></tr>
                <tr><td>Oxygen Level:</td><td>98%</td></tr>
                </tbody>
            </table>
        </section>

        <section id="room-readings">
            <h2>Sensor Monitoring</h2>
            <table>
            <tbody>
            <tr><td>Light:</td><td id="room-light">--</td></tr>
            <tr><td>Sound Status:</td><td id="sound-status">--</td></tr>
            </tbody>
        </table>
        </section>

        <section id="alerts">
        <h2>Alerts</h2>
        <ul id="alert-list"></ul>
        </section>
    </main>
    <div style="text-align: center; margin-top: 20px;">
        <a href="charts.html" class="charts-button">View Detection
Charts</a>
    </div>


    <div id="popup-alert" class="hidden"></div>
    <audio id="alert-sound" src="alarm.mp3" preload="auto"></audio>
</body>
</html>
```

Appendix B.10: dashboard

```
let lastAlertActive = false;

async function updateDashboard() {
  try {
    // === Get Light Data ===
    const lightRes = await fetch('/api/light');
    const lightData = await lightRes.json();
    const light = parseFloat(lightData.light_level);
    const lightBox = document.getElementById("room-light");
    const lightMethods = lightData.methods || [];

    const lightAbnormal = lightMethods.includes("Kalman Filter") &&
lightMethods.includes("Isolation Forest");
    lightBox.textContent = isNaN(light) ? "--" : (lightAbnormal ? "Check
Light " : "Normal");

    // === Get Sound Data ===
    const soundRes = await fetch('/api/sound');
    const soundData = await soundRes.json();
    const soundMethods = soundData.methods || [];
    const statusBox = document.getElementById("sound-status");

    const soundAbnormal = soundMethods.includes("Kalman Filter") &&
soundMethods.includes("Isolation Forest");
    statusBox.textContent = soundAbnormal ? "Abnormal Sound " :
"Everything is OK";

    // === Show Alert if any sensor is abnormal by both methods ===
    const showAlert = lightAbnormal || soundAbnormal;

    const popup = document.getElementById("popup-alert");
    const audio = document.getElementById("alert-sound");

    if (showAlert) {
      const message = lightAbnormal && soundAbnormal
        ? " Both Light & Sound are abnormal!"
        : lightAbnormal
          ? " Abnormal Light Detected!"
          : " Abnormal Sound Detected!";
      showPopup(message);

      if (!lastAlertActive) {
        audio.play();
        addAlertLog(" " + message + " at " + new
Date().toLocaleTimeString());
      }

      lastAlertActive = true;
    } else {
      hidePopup();
      if (lastAlertActive) {
        audio.pause();
        audio.currentTime = 0;
        addAlertLog(" Back to normal at " + new
Date().toLocaleTimeString());
      }

      lastAlertActive = false;
    }

  } catch (error) {
    console.error("Dashboard update error:", error);
    document.getElementById("room-light").textContent = "Error";
    document.getElementById("sound-status").textContent = "Error";
  }
}

function showPopup(message) {
  const popup = document.getElementById("popup-alert");
  popup.textContent = message;
  popup.classList.remove("hidden");
}

function hidePopup() {
  const popup = document.getElementById("popup-alert");
  popup.classList.add("hidden");
}

function addAlertLog(message) {
  const li = document.createElement("li");
  li.textContent = message;
  document.getElementById("alert-list").appendChild(li);
}

setInterval(updateDashboard, 10000);
updateDashboard();

const eventSource = new EventSource('/events');
eventSource.onmessage = function (event) {
  addAlertLog("" + event.data);
};
```

Appendix B.11: Robot

```
import time
import sys
import signal

# Add robot SDK path
sys.path.append('/home/pi/MasterPi/')
```

```
# Import robot and sensor modules
import HiwonderSDK.mecanum as mecanum
from HiwonderSDK.Ultrasonic import Ultrasonic

# Create robot and sensor objects
chassis = mecanum.MecanumChassis()
ultrasonic_sensor = Ultrasonic()

# Robot settings
ROBOT_WIDTH_CM = 15
OBSTACLE_THRESHOLD = 19
STEP = 3  # step size in cm
SIDE_STEP_CM = 5  # for avoid movement
REAL_SPEED = 25  # speed in cm per second

is_running = True

# Stop the robot when Ctrl+C is pressed
def signal_handler(sig, frame):
    global is_running
    print("Stop signal received. Stopping robot.")
    is_running = False
    chassis.stop()
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)

# Check if something is in front of the robot
def is_obstacle_detected(threshold_cm=OBSTACLE_THRESHOLD):
    try:
        distance = ultrasonic_sensor.getDistance()
    except:
        distance = 999  # if sensor fails
    print(f"Distance from obstacle: {distance} cm")
    return distance < threshold_cm

# Move forward
def move_forward(cm):
    print(f"Robot moves forward {cm} cm")
    duration = cm / REAL_SPEED
    chassis.set_velocity(50, 90, 0)  # forward direction
    time.sleep(duration)
    chassis.stop()

# Move right (sideways)
def move_right(cm):
    print(f"Robot moves right {cm} cm")
    duration = cm / REAL_SPEED
    chassis.set_velocity(50, 0, 0)  # right direction
    time.sleep(duration)
    chassis.stop()

# Move left (sideways)
def move_left(cm):
    print(f"Robot moves left {cm} cm")
    duration = cm / REAL_SPEED
    chassis.set_velocity(50, 180, 0)  # left direction
    time.sleep(duration)
    chassis.stop()
```

```
# Rotate robot 90 degrees
def rotate_90(direction):
    print(f"Robot rotates 90 degrees to the {direction}")
    target_angle = 90 if direction == "right" else -90 if direction ==
"left" else 0
    if target_angle == 0:
        print("Invalid rotation direction")
        return

    speed_deg_per_sec = 90
    correction = 0.57
    duration = abs(target_angle) / speed_deg_per_sec
    duration *= correction
    angular_rate = speed_deg_per_sec if target_angle > 0 else -
speed_deg_per_sec

    chassis.set_velocity(0, 0, angular_rate)
    time.sleep(duration)
    chassis.stop()
    print("Rotation complete")

# Move sideways to avoid obstacle
def avoid_obstacle(step_cm=SIDE_STEP_CM):
    print("Obstacle detected. Robot starts avoiding to the right.")
    moved_cm = 0
    while is_obstacle_detected(OBSTACLE_THRESHOLD + ROBOT_WIDTH_CM) and
is_running:
        move_right(step_cm)
        moved_cm += step_cm
        print("Robot keeps avoiding to the right.")
    print(f"Robot finished avoiding. Total moved right: {moved_cm} cm")
    return moved_cm

# Function to correct X or Y position if needed
def correct_position(axis, current_pos, target_pos):
    print(f"Checking if correction needed for {axis.upper()} axis.")
    while current_pos < target_pos and is_running:
        print(f"{axis.upper()} too small. Moving right.")
        move_right(STEP) if axis == 'x' else move_forward(STEP)
        current_pos += STEP

    while current_pos > target_pos and is_running:
        print(f"{axis.upper()} too large. Moving left.")
        move_left(STEP) if axis == 'x' else move_forward(-STEP)
        current_pos -= STEP

    print(f"{axis.upper()} correction done. Current position:
{current_pos} cm")
    return current_pos

# Main function to move robot to a target point (x, y) in meters
def move_to_target(target_x_meters, target_y_meters):
    x_pos = 0
    y_pos = 0
    x_shift_happened = False
    y_shift_happened = False

    # Convert meters to cm
    target_x = target_x_meters * 100
    target_y = target_y_meters * 100
    print(f"Robot starts moving to target: X={target_x} cm, Y={target_y}
cm")

    # Move in Y direction first
    print("Moving in Y direction...")
    while y_pos < target_y and is_running:
        if is_obstacle_detected(OBSTACLE_THRESHOLD + 9):
            print("Obstacle ahead in Y. Stopping to avoid.")
            chassis.stop()
            time.sleep(2)
            shift_x = avoid_obstacle()
            x_pos += shift_x
            x_shift_happened = True
            move_forward(STEP)
            y_pos += STEP
        else:
            move_forward(STEP)
            y_pos += STEP

    print(f"Finished Y movement. Current Y: {y_pos} cm")

    # Move in X direction if needed
    if target_x > 0:
        print("Now rotating and moving in X direction...")
        rotate_90("right")
        time.sleep(2)

        while x_pos < target_x and is_running:
            if is_obstacle_detected(OBSTACLE_THRESHOLD + 9):
                print("Obstacle ahead in X. Stopping to avoid.")
                chassis.stop()
                time.sleep(2)
                shift_y = avoid_obstacle()
                y_pos += shift_y
                y_shift_happened = True
                move_forward(STEP)
                x_pos += STEP
            else:
                move_forward(STEP)
                x_pos += STEP

        print(f"Finished X movement. Current X: {x_pos} cm")
    else:
        print("X target is zero. No movement in X.")

    # Final correction only if needed
    if x_shift_happened and x_pos != target_x:
        print("X correction needed.")
        x_pos = correct_position('x', x_pos, target_x)

    if y_shift_happened and y_pos != target_y:
        print("Y correction needed.")
        y_pos = correct_position('y', y_pos, target_y)

    print("Robot reached the final target and stopped.")

# Run the robot movement
if __name__ == '__main__':
    time.sleep(3)
    move_to_target(0, 1)  # Example target: x=0m, y=1m
```

Appendix B.12 : Smart room

```
# This script reads both light and sound sensors and sends them via MQTT
every 10 seconds

import paho.mqtt.client as mqtt
import json
import time
from time import strftime
import numpy as np
import sounddevice as sd
import smbus

# --- Light Sensor Class ---
bus = smbus.SMBus(1)
class LightSensor:
    def __init__(self):
        self.DEVICE = 0x5c
        self.ONE_TIME_HIGH_RES_MODE_1 = 0x20

    def convertToNumber(self, data):
        return ((data[1] + (256 * data[0])) / 1.2)

    def readLight(self):
        data = bus.read_i2c_block_data(self.DEVICE,
self.ONE_TIME_HIGH_RES_MODE_1)
        return self.convertToNumber(data)

# --- Sound Feature Extraction ---
def extract_sound_features(duration=1, fs=44100):
    recording = sd.rec(int(duration * fs), samplerate=fs, channels=1,
dtype='float64')
    sd.wait()

    # Flatten the array
    signal = recording.flatten()

    # RMS and dB
    rms = np.sqrt(np.mean(signal**2))
    db = 20 * np.log10(rms + 1e-6) + 100

    # Zero Crossing Rate
    zcr = ((signal[:-1] * signal[1:]) < 0).sum() / len(signal)

    return round(db, 2), round(rms, 5), round(zcr, 5)

# --- MQTT Setup ---
MQTT_BROKER = "192.168.1.26"
MQTT_PORT = 1883
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
client.connect(MQTT_BROKER, MQTT_PORT, 60)
print("Connected to MQTT server at", MQTT_BROKER)

light_sensor = LightSensor()

# --- Main loop ---
try:
    while True:
        timestamp = strftime("%Y-%m-%d %H:%M:%S")

        # Read sensors
        light_value = light_sensor.readLight()
        db, rms, zcr = extract_sound_features()

        # Publish light
        light_msg = {
            "sensor": "light",
            "value": light_value,
            "timestamp": timestamp
        }
        client.publish("smartroom/light", json.dumps(light_msg))
        print(f"Light sent: {light_value:.2f} lx")

        # Publish sound with all features
        sound_msg = {
            "sensor": "sound",
            "db": db,
            "rms": rms,
            "zcr": zcr,
            "timestamp": timestamp
        }
        client.publish("smartroom/sound", json.dumps(sound_msg))
        print(f"Sound sent: db={db:.2f}, rms={rms:.5f}, zcr={zcr:.5f}")

        time.sleep(10)

except KeyboardInterrupt:
    print("Stopped sending sensor data.")
```

Appendix C – Code for Testing and Evaluation

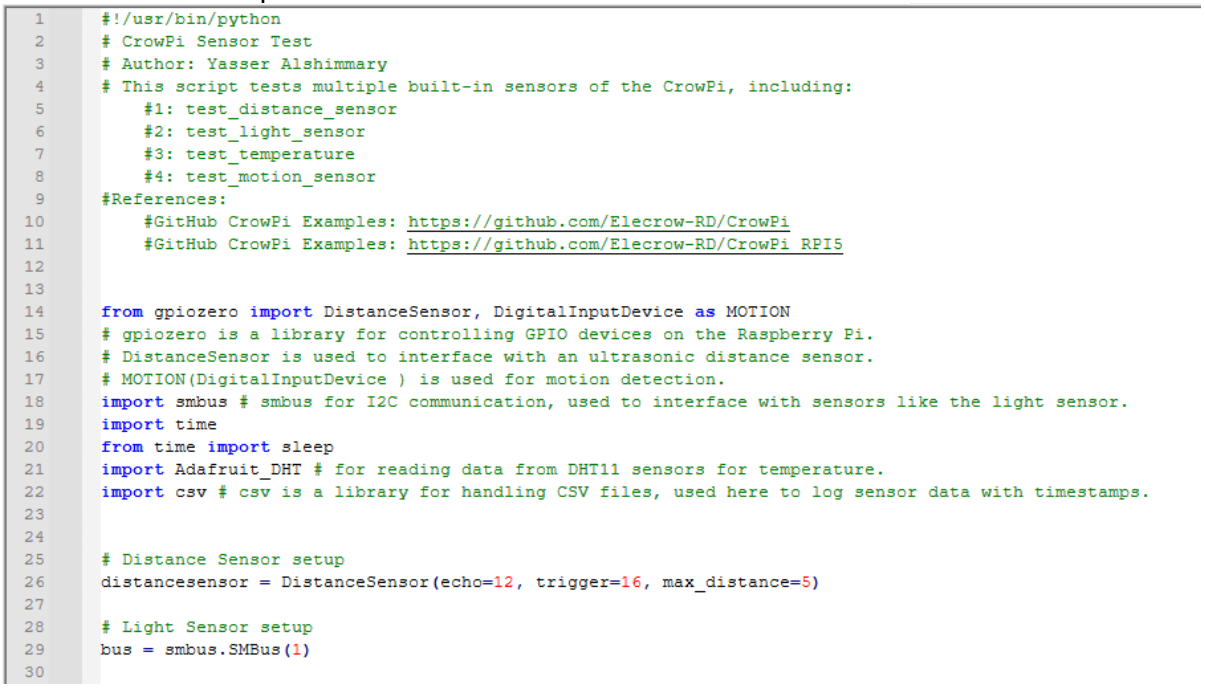
Appendix C.1: Testing Using Pre-Collected Feature Datasets

```
import numpy as np
import pandas as pd
from sklearn.ensemble import IsolationForest

# === Kalman Filter Class ===
class KalmanFilter:
   def __init__(self, F, H, Q, R, P, x0):
      self.F = F
      self.H = H
      self.Q = Q
      self.R = R
      self.P = P
      self.x = x0

   def predict(self):
      self.x = np.dot(self.F, self.x)
      self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q

   def update(self, z):
      S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
      K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
      y = z - np.dot(self.H, self.x)
      self.x = self.x + np.dot(K, y)
      I = np.eye(self.F.shape[1])
      self.P = np.dot(I - np.dot(K, self.H), self.P)

def initialize_kalman_filter():
   F = np.array([[1]])
   H = np.array([[1]])
   Q = np.array([[0.01]])
   R = np.array([[0.1]])
   P = np.array([[1]])
   x0 = np.array([[0]])
   return KalmanFilter(F, H, Q, R, P, x0)

# === Load and Prepare Data ===
train_file = 'F:/TUDublin Class/4th/2/project/2/final project/final day/Smart Hospital
System/Smart Hospital System - 1/trainingData.csv'
test_file = 'F:/TUDublin Class/4th/2/project/2/final project/final day/Smart Hospital
System/Smart Hospital System - 1/Test_Dataset.csv'

features = ["rms", "zcr"]

train_df = pd.read_csv(train_file)
test_df = pd.read_csv(test_file)
```

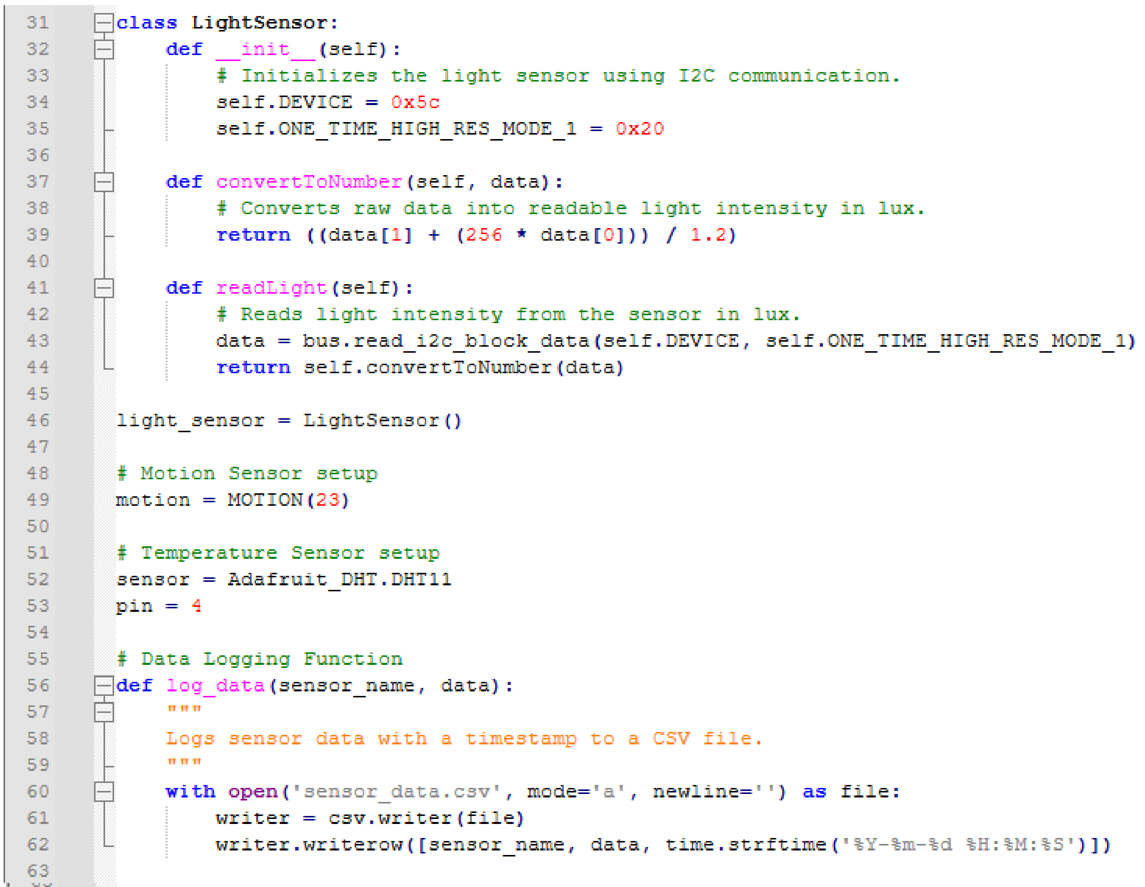
```
# === Adjusted Filtering to Avoid 0 Samples ===
filtered_train = train_df[
    (train_df['label'] == 'normal') &
    (train_df["rms"].between(0.002, 0.05)) &
    (train_df["zcr"].between(0.002, 0.03))
]

X_train = filtered_train[features].values
print("Training samples used:", len(X_train))

# === Train Isolation Forest ===
iso_model = IsolationForest(contamination=0.1)
iso_model.fit(X_train)

# === Initialize Kalman Filters (for rms and zcr) ===
kalman_filters = [initialize_kalman_filter() for _ in features]

# === Run Detection ===
iso_preds = []
kalman_preds = []

for _, row in test_df.iterrows():
    values = row[features].values.astype(float)

    # Isolation Forest
    iso_result = iso_model.predict([values])[0]
    iso_preds.append("abnormal" if iso_result == -1 else "normal")

    # Kalman Filter
    kalman_result = []
    for i, val in enumerate(values):
        kf = kalman_filters[i]
        kf.predict()
        kf.update(np.array([[val]]))
        residual = abs(kf.x[0][0] - val)
        threshold = 0.004 if features[i] == "rms" else 0.003
        kalman_result.append(residual > threshold)
    kalman_preds.append("abnormal" if any(kalman_result) else "normal")

# === Save Results to CSV ===
test_df["isolation_result"] = iso_preds
test_df["kalman_result"] = kalman_preds
test_df.to_csv("anomaly_detection_results.csv", index=False)
print("Results saved to 'anomaly_detection_results.csv'.")

# === Evaluation Function ===
def evaluate(true, pred):
    TP = sum((t == "abnormal" and p == "abnormal") for t, p in zip(true, pred))
    TN = sum((t == "normal" and p == "normal") for t, p in zip(true, pred))
    FP = sum((t == "normal" and p == "abnormal") for t, p in zip(true, pred))
    FN = sum((t == "abnormal" and p == "normal") for t, p in zip(true, pred))

    total = TP + TN + FP + FN
    accuracy = (TP + TN) / total if total else 0
    precision = TP / (TP + FP) if (TP + FP) else 0
    recall = TP / (TP + FN) if (TP + FN) else 0

    return {
        "TP": TP, "TN": TN, "FP": FP, "FN": FN,
        "Accuracy": round(accuracy, 2),
        "Precision": round(precision, 2),
        "Recall": round(recall, 2)
    }

# === Print Metrics ===
true_labels = test_df["label"].values
iso_metrics = evaluate(true_labels, iso_preds)
kalman_metrics = evaluate(true_labels, kalman_preds)

print("\nIsolation Forest Metrics:")
print(iso_metrics)
print("\nKalman Filter Metrics:")
print(kalman_metrics)
```

Appendix D – Smart room Sensor Testing:
The smart room setup began with testing the built-in sensors available on the smart room kit. The main goal was to make sure each sensor worked correctly and could provide useful data for the system.
Sensors Tested:
Distance Sensor: Measures how far an object is from the sensor using ultrasonic waves. The distance is shown on the screen in centimetres and saved in a CSV file with a timestamp.
Light Sensor: Reads the room’s light level (in lux) and logs the readings in real-time.
Temperature Sensor: Uses the DHT11 sensor to measure ambient room temperature. The script includes error handling, if a reading fails, it tries again until it works. Data is saved with timestamps.
Motion Sensor: Detects movement in the room. Each time movement is detected (or not), a message is logged.

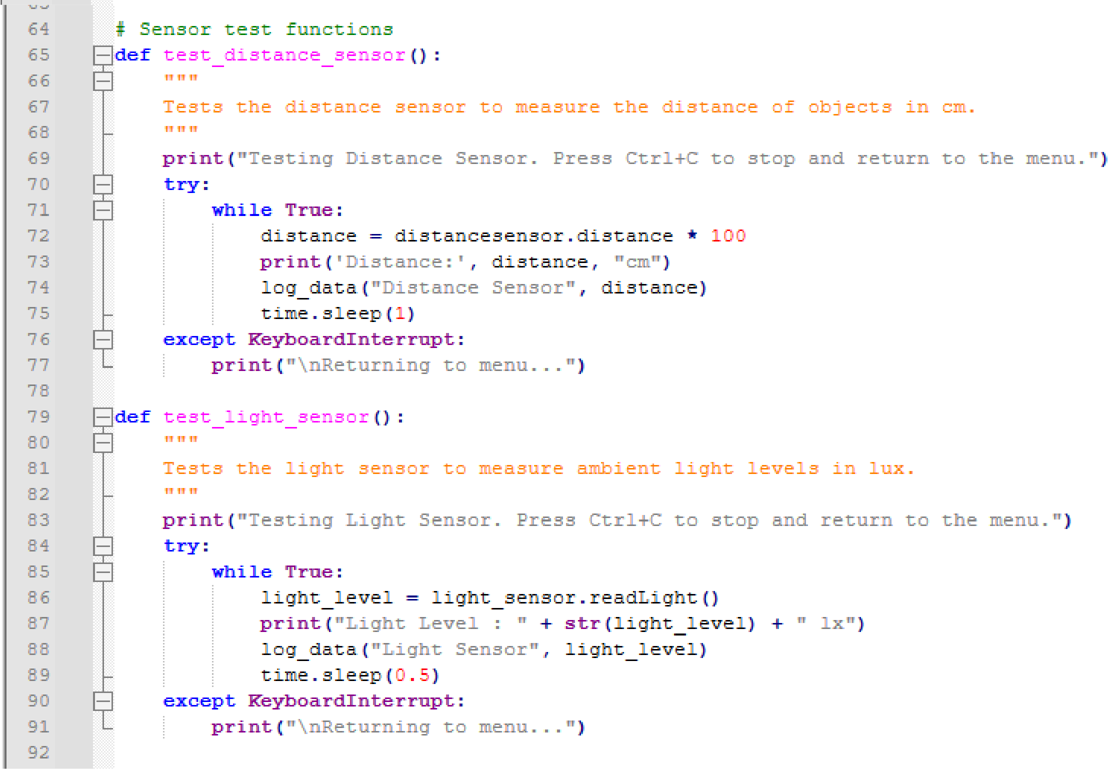
How the Code Works:
A Python script was created to test all four sensors. The program is interactive — it shows a menu so the user can choose which sensor to test. Once selected, the sensor runs in a loop and prints live readings to the screen. It also logs each reading in a CSV file with a timestamp. This was helpful to check if the sensors work properly and to collect sample data.



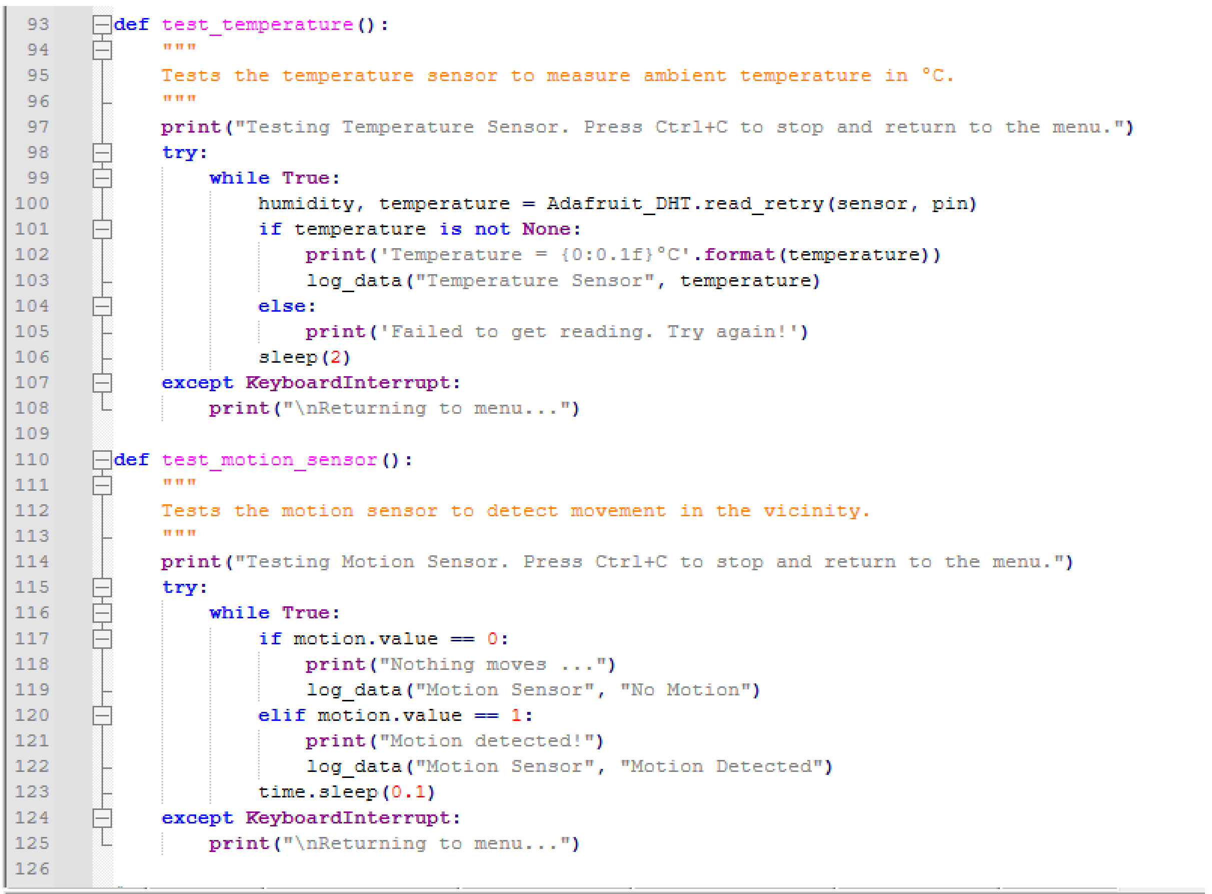
```python
#!/usr/bin/python
# CrowPi Sensor Test
# Author: Yasser Alshimmary
# This script tests multiple built-in sensors of the CrowPi, including:
    #1: test_distance_sensor
    #2: test_light_sensor
    #3: test_temperature
    #4: test_motion_sensor
#References:
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi_RPI5


from gpiozero import DistanceSensor, DigitalInputDevice as MOTION
# gpiozero is a library for controlling GPIO devices on the Raspberry Pi.
# DistanceSensor is used to interface with an ultrasonic distance sensor.
# MOTION(DigitalInputDevice ) is used for motion detection.
import smbus # smbus for I2C communication, used to interface with sensors like the light sensor.
import time
from time import sleep
import Adafruit_DHT # for reading data from DHT11 sensors for temperature.
import csv # csv is a library for handling CSV files, used here to log sensor data with timestamps.


# Distance Sensor setup
distancesensor = DistanceSensor(echo=12, trigger=16, max_distance=5)

# Light Sensor setup
bus = smbus.SMBus(1)

```



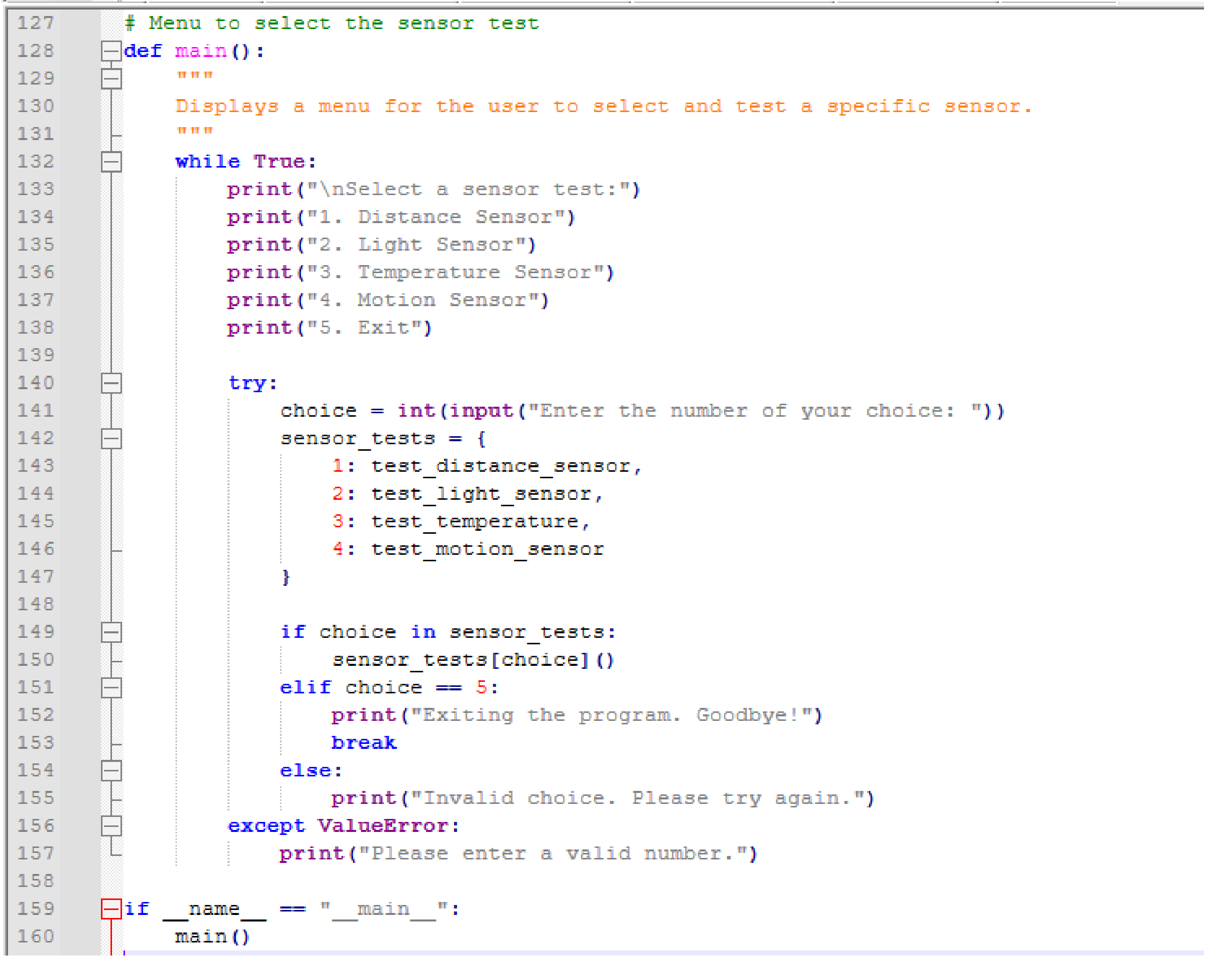
```python
class LightSensor:
    def __init__(self):
        # Initializes the light sensor using I2C communication.
        self.DEVICE = 0x5c
        self.ONE_TIME_HIGH_RES_MODE_1 = 0x20

    def convertToNumber(self, data):
        # Converts raw data into readable light intensity in lux.
        return ((data[1] + (256 * data[0])) / 1.2)

    def readLight(self):
        # Reads light intensity from the sensor in lux.
        data = bus.read_i2c_block_data(self.DEVICE, self.ONE_TIME_HIGH_RES_MODE_1)
        return self.convertToNumber(data)

light_sensor = LightSensor()

# Motion Sensor setup
motion = MOTION(23)

# Temperature Sensor setup
sensor = Adafruit_DHT.DHT11
pin = 4

# Data Logging Function
def log_data(sensor_name, data):
    """
    Logs sensor data with a timestamp to a CSV file.
    """
    with open('sensor_data.csv', mode='a', newline='') as file:
        writer = csv.writer(file)
        writer.writerow([sensor_name, data, time.strftime('%Y-%m-%d %H:%M:%S')])

```



```python
# Sensor test functions
def test_distance_sensor():
    """
    Tests the distance sensor to measure the distance of objects in cm.
    """
    print("Testing Distance Sensor. Press Ctrl+C to stop and return to the menu.")
    try:
        while True:
            distance = distancesensor.distance * 100
            print('Distance:', distance, "cm")
            log_data("Distance Sensor", distance)
            time.sleep(1)
    except KeyboardInterrupt:
        print("\nReturning to menu...")

def test_light_sensor():
    """
    Tests the light sensor to measure ambient light levels in lux.
    """
    print("Testing Light Sensor. Press Ctrl+C to stop and return to the menu.")
    try:
        while True:
            light_level = light_sensor.readLight()
            print("Light Level : " + str(light_level) + " lx")
            log_data("Light Sensor", light_level)
            time.sleep(0.5)
    except KeyboardInterrupt:
        print("\nReturning to menu...")

```



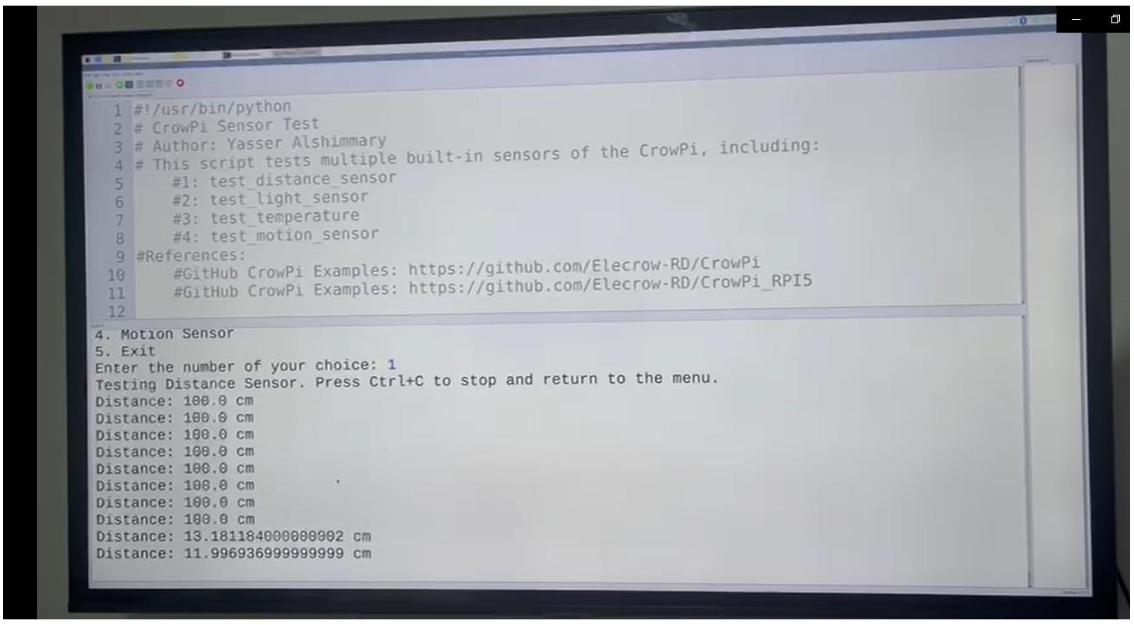
```python
def test_temperature():
    """
    Tests the temperature sensor to measure ambient temperature in °C.
    """
    print("Testing Temperature Sensor. Press Ctrl+C to stop and return to the menu.")
    try:
        while True:
            humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
            if temperature is not None:
                print('Temperature = {0:0.1f}°C'.format(temperature))
                log_data("Temperature Sensor", temperature)
            else:
                print('Failed to get reading. Try again!')
            sleep(2)
    except KeyboardInterrupt:
        print("\nReturning to menu...")

def test_motion_sensor():
    """
    Tests the motion sensor to detect movement in the vicinity.
    """
    print("Testing Motion Sensor. Press Ctrl+C to stop and return to the menu.")
    try:
        while True:
            if motion.value == 0:
                print("Nothing moves ...")
                log_data("Motion Sensor", "No Motion")
            elif motion.value == 1:
                print("Motion detected!")
                log_data("Motion Sensor", "Motion Detected")
            time.sleep(0.1)
    except KeyboardInterrupt:
        print("\nReturning to menu...")

```



```python
# Menu to select the sensor test
def main():
    """
    Displays a menu for the user to select and test a specific sensor.
    """
    while True:
        print("\nSelect a sensor test:")
        print("1. Distance Sensor")
        print("2. Light Sensor")
        print("3. Temperature Sensor")
        print("4. Motion Sensor")
        print("5. Exit")

        try:
            choice = int(input("Enter the number of your choice: "))
            sensor_tests = {
                1: test_distance_sensor,
                2: test_light_sensor,
                3: test_temperature,
                4: test_motion_sensor
            }

            if choice in sensor_tests:
                sensor_tests[choice]()
            elif choice == 5:
                print("Exiting the program. Goodbye!")
                break
            else:
                print("Invalid choice. Please try again.")
        except ValueError:
            print("Please enter a valid number.")

if __name__ == "__main__":
    main()
```

Purpose of This Testing:
This testing was an important first step to building the smart room. It helped confirm that the sensors are reliable and ready to be connected to the rest of the system. The data collected during these tests was used to plan the MQTT communication, database storage, and data filtering in later stages.

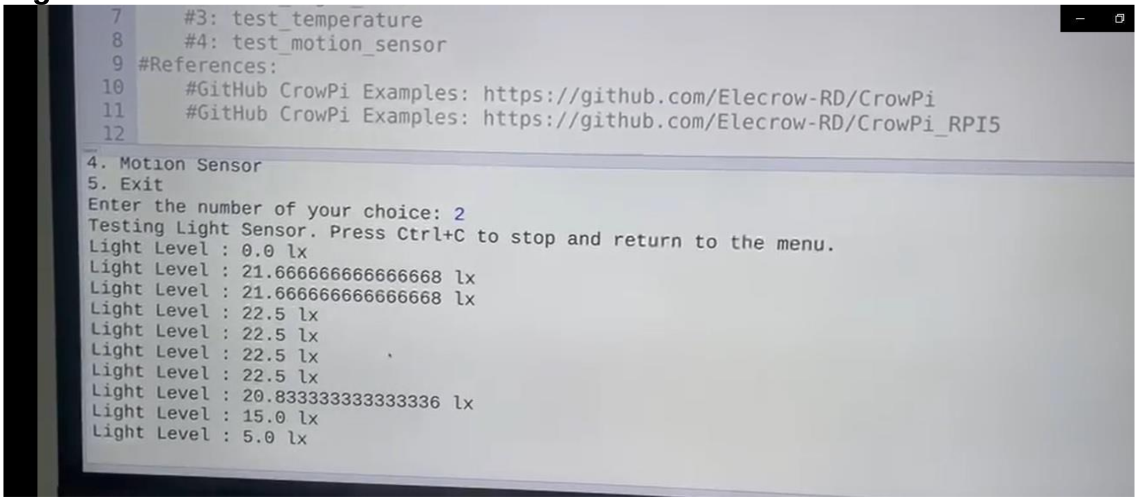
Distance sensor:



```
#!/usr/bin/python
# CrowPi Sensor Test
# Author: Yasser Alshimmary
# This script tests multiple built-in sensors of the CrowPi, including:
    #1: test_distance_sensor
    #2: test_light_sensor
    #3: test_temperature
    #4: test_motion_sensor
#References:
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi_RPI5
```

```
4. Motion Sensor
5. Exit
Enter the number of your choice: 1
Testing Distance Sensor. Press Ctrl+C to stop and return to the menu.
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 100.0 cm
Distance: 13.181184000000002 cm
Distance: 11.996936999999999 cm
```

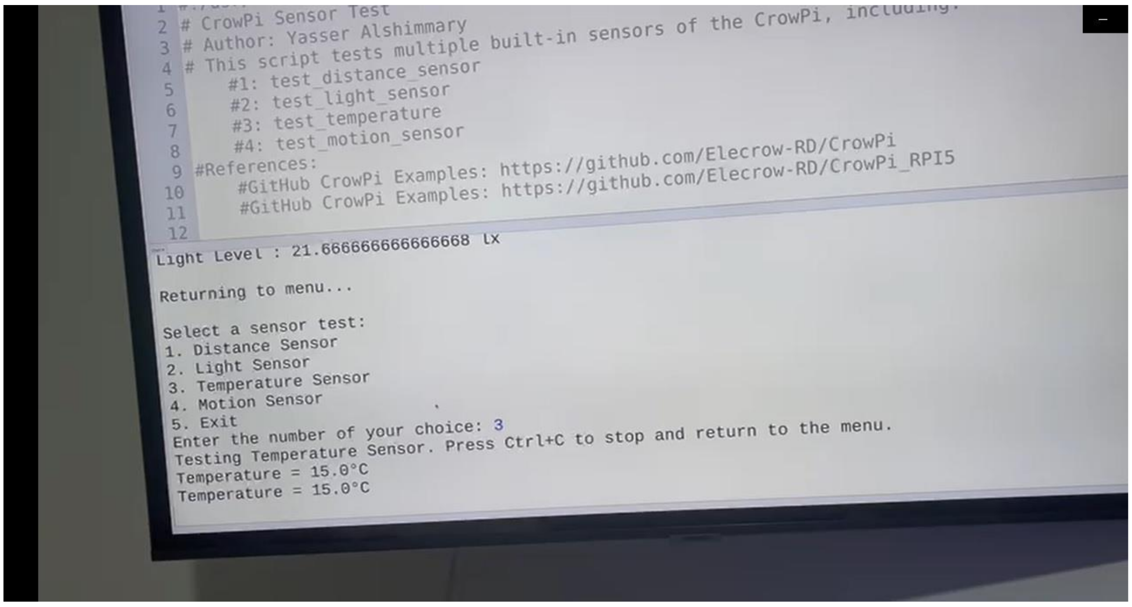
Light sensor:



```
    #3: test_temperature
    #4: test_motion_sensor
#References:
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi_RPI5
```

```
4. Motion Sensor
5. Exit
Enter the number of your choice: 2
Testing Light Sensor. Press Ctrl+C to stop and return to the menu.
Light Level : 0.0 lx
Light Level : 21.666666666666668 lx
Light Level : 21.666666666666668 lx
Light Level : 22.5 lx
Light Level : 22.5 lx
Light Level : 22.5 lx
Light Level : 22.5 lx
Light Level : 20.833333333333336 lx
Light Level : 15.0 lx
Light Level : 5.0 lx
```

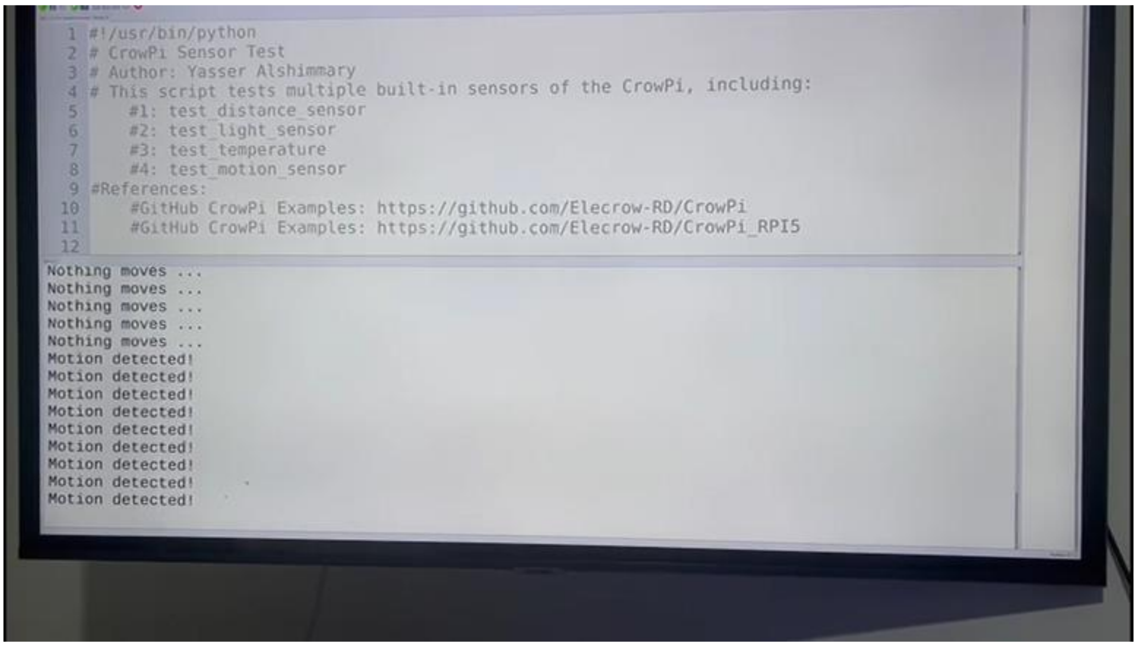
Temperature Sensor:



```
# CrowPi Sensor Test
# Author: Yasser Alshimmary
# This script tests multiple built-in sensors of the CrowPi, including:
    #1: test_distance_sensor
    #2: test_light_sensor
    #3: test_temperature
    #4: test_motion_sensor
#References:
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi_RPI5
```

```
Light Level : 21.666666666666668 lx

Returning to menu...

Select a sensor test:
1. Distance Sensor
2. Light Sensor
3. Temperature Sensor
4. Motion Sensor
5. Exit
Enter the number of your choice: 3
Testing Temperature Sensor. Press Ctrl+C to stop and return to the menu.
Temperature = 15.0°C
Temperature = 15.0°C
```

Motion sensor:



```
#!/usr/bin/python
# CrowPi Sensor Test
# Author: Yasser Alshimmary
# This script tests multiple built-in sensors of the CrowPi, including:
    #1: test_distance_sensor
    #2: test_light_sensor
    #3: test_temperature
    #4: test_motion_sensor
#References:
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi
    #GitHub CrowPi Examples: https://github.com/Elecrow-RD/CrowPi_RPI5
```

```
Nothing moves ...
Nothing moves ...
Nothing moves ...
Nothing moves ...
Nothing moves ...
Motion detected!
Motion detected!
Motion detected!
Motion detected!
Motion detected!
Motion detected!
Motion detected!
Motion detected!
Motion detected!
```

Appendix D – Setting Up the MasterPi Robot

During development, the MasterPi Robot faced system limitations due to its default operating system. The robot originally used the armv7l (32-bit) OS, which caused problems when trying to run YOLOv8 and other Python-based AI tools.
Why the OS Had to Be Upgraded:
The armv7l (32-bit) system could not run certain applications like YOLOv8 and PyTorch because they require 64-bit support. These tools are built for modern systems that include advanced instructions not available on 32-bit devices. When trying to run YOLOv8, errors such as "Illegal instruction" appeared, meaning the system could not process the required commands.
To solve this, the robot's operating system was upgraded to aarch64 (64-bit).
Steps Taken to Upgrade the Robot's System:

- Disassembled the Robot: The robot had to be taken apart carefully to remove the microSD card from the Raspberry Pi 4 board.
- Backed Up Important Data: Important files and folders were copied before formatting the card. This included:
    - hiwonder-toolbox
    - MasterPi
    - MasterPi_PC_Software
- Installed 64-bit Raspberry Pi OS: Used the Raspberry Pi Imager to flash the aarch64 (64-bit) OS onto the microSD card.
  Inserted the card back into the robot's Raspberry Pi and powered it up.
- Reinstalled Key Services and Tools: After the OS upgrade, several services had to be reinstalled:
  - hw_button_scan.service
  - hw_find.service
  - hw_remote.service
  - masterpi.service
  - Hiwonder Arm Python application.