**Assessment Cover Sheet**

|  |  |
| --- | --- |
| **Programme Name:** | Internet of Things |
| **Partner University Name:** | Newcastle College Group |
| **Semester 1 or 2:** | Semester 2 |
| **Module Code:** | CMP213 |
| **Name of Lecturer:** | Priyanthi Manjula Dassanayake |
| **Due Date:** | 17.10.2025 |
| **Assessment Type:** | Project |

|  |  |
| --- | --- |
| **Declaration of Authenticity (please check and sign):**  By submitting this assignment, I declare that this work is fully my own, and the work of others (including internet sources) is acknowledged by quotations and appropriate citing and referencing.  I declare that this work has not made use of the work of any other student(s) past or present at this or any other educational institution or source.  I declare that I have not commissioned another person to complete this work. This could include the use of professional essay-writing services, essay banks, or ghostwriting services.  I declare that I have not asked another person to complete this work for me. This could include the help of friends, classmates, other students, or family members.  I declare I have not used any AI software to help write this work outside of the assessment brief/guidelines (this includes translation tools, paraphrasing tools, or content creation websites such as ‘ChatGPT’), other than for proofreading.  I acknowledge that I have read all the texts listed in the bibliography or references list. I confirm all sources cited in the work appear in the bibliography or references list at the end. |  |
|  |
|  |
|  |
|  |
|  |
|  |
| **Signed: Nicolae Sterian** | |

# Configurable IoT Intruder Detection System

# Document Purpose & Scope

This document outlines the design plan for a basic intruder alert system using a Raspberry Pi 5. It explains what the system is meant to do, how it should work, and why certain components were chosen. The idea is to detect movement within a short range using an ultrasonic sensor, and if necessary, send an alert (with or without a photo) via Discord.

This is the planning stage, so everything written here is based on how the system is intended to work — before it’s actually built or tested. The goal is to create a clear blueprint that can guide the development and testing phases that follow.

1. **Executive Summary**

This project sets out to design a small and affordable intruder detection system using a **Raspberry Pi 5**, an **HC-SR04 ultrasonic sensor**, and optionally, a **Camera Module 3**. The main goal is to detect when someone or something comes too close and instantly send an alert to the user through **Discord**. If a camera is connected and enabled, the system will also try to take a photo and include it in the alert.

The system is meant for indoor use — in places like a garage, home office, or classroom — where simple motion detection could provide a useful warning. It works by constantly measuring distance, and when something moves closer than a set limit (like 35 cm), it triggers an alert. To avoid flooding the user with too many messages, there’s a short cooldown period between alerts.

For safety, especially in learning environments, the sensor is powered at *3.3V*, which slightly reduces range but keeps it safe for the **Raspberry Pi**. The system is also designed to be flexible — users can turn the camera on or off, change the distance threshold, or adjust the cooldown time just by changing environment variables, without touching the code.

This document outlines the plan for how the system will be built and how it’s supposed to work. It covers the architecture, the logic flow, and the reasons behind the technical decisions — from GPIO handling to alert methods — so the design remains safe, adaptable, and easy to understand.

# 2. Problem Statement & Requirements

## 2.1 Scenario & Constraints

This project is meant to be used indoors, in spaces like a garage, study room, workshop, or classroom. The idea is to provide a basic and low-cost way of detecting nearby movement and sending an instant alert. It's not a full security system, but more of a prototype to show how a **Raspberry Pi** can work with sensors and messaging tools like **Discord** to create a functional early warning setup.

The system is built with safety, simplicity, and demonstrations in mind. For example:

* The **HC-SR04 sensor** is powered at 3.3 volts to protect the Raspberry Pi’s GPIO pins.
* The range is intentionally limited to keep things manageable — perfect for close-range detection in a small area.
* All components are chosen to be affordable and easy to source.
* The design avoids cloud dependencies and doesn’t require long-term storage or user accounts — it just sends a real-time notification when something is detected.

By keeping the system lightweight and local, it becomes ideal for learning environments and quick testing scenarios where reliability, fast feedback, and safety are key.

## 2.2 Functional Requirements

The system is expected to meet the following **core functional requirements**:

* **Continuous Distance Monitoring**: Measure distance using the HC-SR04 sensor, sampling at a fixed interval.
* **Intruder Detection**: If the measured distance falls below a configurable threshold (e.g., 35 cm), trigger an alert.
* **Cooldown Period**: After a valid alert is sent, enter a configurable cooldown phase where no new alerts are sent, even if movement continues.
* **Discord Notification**: Send an alert message to a pre-set Discord webhook URL with timestamp and optional image.
* **Optional Camera Capture**: If enabled and a camera is connected, take a photo and attach it to the Discord alert.
* **Fallback Behaviour**: If the camera is disabled or fails, still send a text-only alert to ensure core detection always works.

## 2.3 Non-Functional Requirements

To ensure the system is practical and educational, the following non-functional criteria are also set at design time:

* **Safe Electrical Operation**: Sensor operates at 3.3 V to match Pi 5 logic levels.
* **Latency**: Detection-to-alert time should be under **2 seconds**, even with image capture.
* **Configurability**: Threshold, cooldown, sample rate, webhook URL, and camera usage must be controlled via **environment variables** (no code edits required).
* **Resilience**: System should not crash if camera is missing or disabled.
* **Portability**: All required software should be installable on Bookworm with standard commands (e.g. apt, pip).
* **Clarity**: Log output must clearly indicate what triggered the alert (distance, time, status) for debug and verification purposes.

## 2.4 Acceptance Criteria

The following measurable criteria must be met during testing to validate the design:

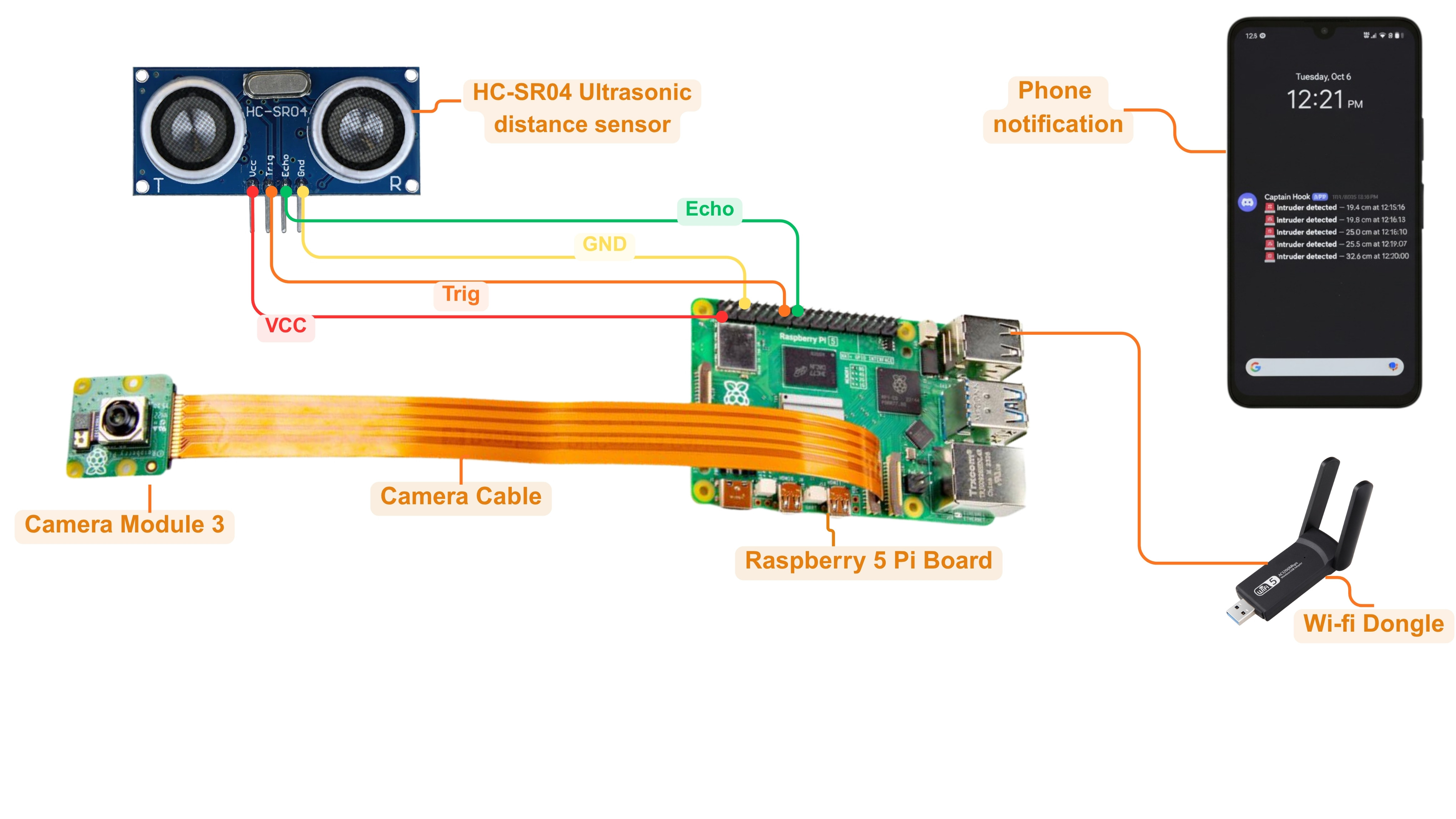
| **Testable Feature** | **Acceptance Condition** |
| --- | --- |
| Distance detection | Must trigger if object < threshold (e.g., 35 cm) |
| Cooldown behaviour | No duplicate alerts within cooldown window |
| Discord alert (text) | Must send successful message with correct timestamp |
| Discord alert (with photo) | If camera is enabled and working, attach photo to alert |
| Discord alert (fallback) | If camera is disabled or fails, alert still sends |
| Safety | Sensor powered at 3.3 V with no warnings or GPIO issues |
| Logging | Must log “TRIGGER”, “COOLDOWN”, and “CLEAR” events clearly |
| Configurability | Changes to env vars reflect in runtime without edits |

# 3. System Architecture (at design time)

The system is built around a straightforward data pipeline that begins with the **ultrasonic distance sensor** and ends with an alert message sent to a **Discord webhook**. At the core of the architecture is a simple decision-making logic that determines when to trigger an alert and how to respond depending on the current state (cooldown or not, camera available or not).

The diagram below (Figure D1) illustrates the key components and how data flows between them:

**Figure D1:** High-Level Block Diagram

****

## 3.1 Data Flow Overview

The data flow proceeds as follows:

1. **Sensor Input**  
   The HC-SR04 ultrasonic sensor measures the distance to the nearest object in front of it. This value is sampled at a fixed interval (default: 0.25 seconds).
2. **Threshold & Cooldown Logic**  
   The system checks whether the detected distance is below a predefined threshold (e.g., 35 cm). If it is and the system is not currently in a cooldown period, an alert is triggered. After a valid alert, a cooldown timer starts, during which further alerts are temporarily suppressed to avoid spamming.
3. **Camera Capture (Optional)**  
   If the photo feature is enabled and a camera is connected, the system attempts to capture an image of the scene. It uses the best available tool in this order: rpicam-still (Bookworm), libcamera-still, then fswebcam.
4. **Discord Notification**  
   A message is composed with the timestamp and distance data, and sent to a Discord webhook URL. If a photo was successfully taken, it is included as an attachment. If the camera is unavailable or the capture fails, the system still sends the alert as plain text.
5. **Logging**  
   Throughout the process, the system logs key events such as when the sensor detects motion (“TRIGGER”), when an alert is skipped due to cooldown, and when the space becomes clear again (“CLEAR”).

This architecture is designed to be lightweight, reliable, and easy to debug — making it suitable for demonstration, learning, and basic motion alert use cases.

# 4. Operation Flow (Intended)

The system is designed to follow a simple, clear control flow built around four key states: **Idle**, **Trigger**, **Cooldown**, and **Clear**. These states define how the sensor monitors the space, how alerts are triggered, and how the system avoids sending repeated messages during short bursts of motion.

The control loop will be written in Python and will run continuously while keeping logs that clearly show what’s happening at every stage. It relies on two main checks: the measured distance from the sensor, and whether the cooldown timer has expired since the last alert.

***Figure D2*:** PlantUML Flowchart  
A diagram of a program

AI-generated content may be incorrect.

## 4.1 Operation Walkthrough

* **Idle**  
  The default state. The sensor measures distance repeatedly. As long as nothing is detected within the defined range (e.g. over 35 cm), the system stays in this state.
* **Trigger**  
  If the distance suddenly drops below the set threshold (meaning something came close), the system logs a trigger event. If it’s not in cooldown mode, it moves on to send an alert.
* **Cooldown**  
  Once an alert has been sent, the system starts a short cooldown timer (default: 30 seconds). During this period, new triggers are detected and logged but won’t send more alerts. This avoids spamming the user while still showing motion is happening.
* **Clear**  
  When the sensor starts seeing no object in range again, it logs a “CLEAR” event to show that the area has returned to normal. After that, it goes back to Idle and waits for the next event.

This state flow makes the system predictable and efficient. It’s especially useful for testing or classroom demos, as you can see each phase in the logs and know exactly what’s happening. By keeping the logic modular and event-driven, it’s easy to adjust or expand later if needed.

# Design Decisions & Rationale

This section explains the reasoning behind each major design decision, both hardware and software. The aim is to keep the system safe, simple, and demonstrable, while ensuring that it functions reliably in a classroom or workshop environment.

## 5.1 Hardware Selections

**Raspberry Pi 5:**  
The project is designed to run on a Raspberry Pi 5 because it offers better performance and improved GPIO control compared to previous models. It also provides direct support for the newer camera modules and is fully compatible with the Bookworm version of Raspberry Pi OS, which is required for the rpicam-still tool used for image capture. The Pi 5 is powerful enough to handle both sensor readings and quick image processing without lag, making it a reliable choice for this type of prototype.

**HC-SR04 Ultrasonic Sensor (3.3V Safe Mode):**  
The HC-SR04 sensor was selected because it’s inexpensive, widely available, and very easy to integrate with the Raspberry Pi. To keep the project safe for classroom use, it will run directly at 3.3V instead of 5V. This prevents potential damage to the Pi’s GPIO pins while still offering a usable range of about 1.5 metres — more than enough for short-range indoor detection. It’s ideal for simple setups such as a desk area, small room, or demonstration environment.

**Camera Module 3 + Adapter (22-pin to 15-pin)**:  
The Camera Module 3 (IMX708 sensor) is selected to capture optional snapshots of intrusions. Since the Pi 5 uses a 22-pin CSI port, a 22-to-15 ribbon cable will be used to adapt the standard camera module for compatibility.

## 5.2 Software Stack

* **Raspberry Pi OS (Bookworm)**:  
  Bookworm is chosen as the latest stable Pi OS. It includes built-in support for the new camera stack (rpicam-apps) and avoids compatibility issues present in older releases.
* **Python + gpiozero with lgpio backend**:  
  The system is coded in Python for ease of development and readability. The gpiozero library is used for controlling the sensor and GPIOs. On the Pi 5, the lgpio backend is preferred due to better performance and compatibility. The system automatically falls back to pigpio or the default backend if needed.
* **Camera utilities (rpicam-still, libcamera-still, fswebcam)**:  
  Image capture is handled through a layered fallback system:
  1. rpicam-still (preferred, modern stack)
  2. libcamera-still (fallback if rpicam fails)
  3. fswebcam (USB camera fallback)
* **Discord webhook**:  
  The system sends alerts using a Discord webhook because it’s simple to set up, widely used, and supports both text and image notifications. Unlike email or SMS services, Discord allows instant delivery without complex configuration or extra costs. This makes it a practical choice, especially for quick demos or classroom use where alerts need to be seen straight away.

## 5.3 Threshold & Cooldown Model

* **Distance Threshold**:  
  The system will allow users to set a detection distance — with 35 cm as the default. This lets the system ignore normal background movement and only trigger when something enters a defined close-range area. The goal is to focus alerts on meaningful motion, not random noise.
* **Cooldown Period**:  
  To avoid sending multiple alerts in a row, the system includes a cooldown timer. Once an alert is sent, it will wait a set amount of time (default: 30 seconds) before it can send another. This helps prevent message spam in the Discord channel and keeps the alert log readable and useful.

## 5.4 Event-Driven vs Continuous Logging

Rather than logging every distance reading, the system is designed to react only to key events — like when an object enters or exits the threshold range. This event-driven model keeps logs much cleaner and makes the code more efficient. It also avoids wasting system resources on constant background updates that don’t lead to any action.

* **Why event-driven**:
  + Less noise in terminal/log output
  + Easier to track real events (TRIGGER, CLEAR)
  + Reduces unnecessary Discord traffic
  + Ideal for systems where visual monitoring isn't possible (e.g., headless Pi)

This design makes the alert system feel responsive and intelligent without wasting resources.

# Assumptions, Risks & Mitigations

Before starting the build, a few key assumptions are made about where and how this system will be used. Alongside that, possible risks are considered early to help avoid issues later. For each risk, the design includes some kind of safety measure or backup plan. This way, the system stays reliable, safe to use, and easy to demonstrate live — whether in a classroom, workshop, or small indoor space.

## 6.1 Assumptions

* **Stable Wi-Fi**:  
  The system assumes the Raspberry Pi has an active and stable Wi-Fi connection. This is necessary to send notifications to the Discord webhook in real time.
* **Indoor Environment**:  
  The acoustic behaviour of the HC-SR04 ultrasonic sensor works best in quiet, indoor spaces with minimal background interference. Echoes from hard surfaces or irregular objects may slightly affect readings, but are manageable within the short-range design.
* **Discord Webhook Availability**:  
  It is assumed the user has already set up a valid Discord webhook URL and has access to the target Discord server or channel.
* **Sufficient Lighting (optional)**:  
  For the optional camera snapshot feature, ambient lighting is assumed to be good enough for usable images. No flash or IR components are included in this design.

## 6.2 Risks and Their Impacts

| **Risk** | **Potential Impact** |
| --- | --- |
| **Sensor Noise / False Readings** | May trigger unwanted alerts or miss real movement |
| **Camera Not Detected** | No image is sent, or script crashes if unhandled |
| **Webhook URL Leak** | Unauthorized access to Discord channel or spam risk |
| **GPIO Miswiring or Overvoltage** | Permanent damage to the Pi or sensor |
| **Power Supply Fluctuations** | Inconsistent behaviour or shutdowns during operation |
| **Improper Cooldown Logic** | Flooding of alerts or missed detections |
| **Slow Discord API / Latency** | Alerts may appear delayed or out of order |

## 6.3 Mitigations

* **Sensor Noise → queue\_len & Planned Hysteresis**:  
  The DistanceSensor uses queue\_len=3 to average values and smooth out sudden spikes. A further improvement involves adding hysteresis logic (not yet implemented) to require the sensor to stay in-range for multiple cycles before triggering.
* **Camera Issues → Fallback Checks**:  
  The script uses a cascading fallback system: if the camera fails to capture an image, the alert is still sent with text-only. This prevents the main functionality from failing.
* **Webhook Leakage → Use .env Files / Secrets**:  
  The webhook URL is kept out of the source code by reading from an environment variable. This avoids accidental leaks when sharing or uploading the code.
* **GPIO Damage → Safe Power Design**:  
  To protect the Raspberry Pi, the HC-SR04 sensor is powered directly from the 3.3V pin. This avoids the risk of sending a 5V signal back into a GPIO pin, which could damage the board. The wiring follows a clear plan to make sure everything is connected safely.
* **Power Supply Stability → Quality PSU & Safe Wiring**:  
  A standard official Raspberry Pi 5 power supply is used to avoid voltage dips or instability. All wiring is double-checked and only done while the Pi is powered off, to prevent short circuits or unintentional GPIO contact.
* **Cooldown Logic → Time-Based Gate & Logging**:  
  The system uses a timestamp to track when the last alert was sent. If another detection happens during the cooldown period, it’s logged but not sent — making the system easier to test and the logs easier to follow.
* **Latency Awareness → Logged Timestamps**:  
  While the system isn’t built for real-time guarantees, log entries include exact timestamps. These help track how long it takes from detecting movement to seeing the message in Discord, which is useful for evaluating responsiveness.

# 7. Planned Interfaces & Pin Map

This section shows how the sensor is physically connected to the Raspberry Pi 5, and how alerts are sent out using a Discord webhook. Both the GPIO wiring and the external interface are designed to be simple, safe, and easy to test in a classroom or demo setup.

## 7.1 GPIO Wiring Plan

The HC-SR04 ultrasonic sensor needs four basic connections: power, ground, trigger, and echo. To make sure everything runs safely on the Raspberry Pi 5, the sensor will be powered from the 3.3V pin. This limits its maximum range a bit, but avoids the risk of overloading the Pi’s GPIO with 5V signals from the ECHO pin.

Here’s the planned pin layout:

| **Pin** | **Connected To** | **Purpose** |
| --- | --- | --- |
| 3.3V (Pin 1) | VCC on HC-SR04 | Power supply (safe voltage) |
| GND (Pin 6) | GND on HC-SR04 | Common ground |
| BCM 23 (Pin 16) | TRIG on HC-SR04 | Sends trigger pulse |
| BCM 24 (Pin 18) | ECHO on HC-SR04 | Receives distance echo |

This setup avoids the need for level shifting and keeps the wiring simple and safe — especially important during hands-on demos or quick setups in class.

## 7.2 External Interface: Discord Webhook

The system sends alerts using a Discord webhook. This acts as the main communication channel between the Raspberry Pi and the user. Depending on whether the camera is available, the messages sent through Discord will be either text-only or include a photo.

* **Text-Only Message (Default)**  
  If the camera is not connected or disabled, the alert is just a message showing the time and the distance recorded when motion was detected.  
  *Example:*

Intruder detected — 33.4 cm at 14:03:12

* **Alert with Image (Optional)**If the camera is working, the system also captures a snapshot and attaches it to the alert.  
  *Example:*

Intruder detected — 28.9 cm at 12:57:46  
[attached file: intruder.jpg]

To keep things secure, the Discord webhook URL is stored in an environment variable (WEBHOOK\_URL) rather than hardcoded into the script. This avoids accidental leaks if the code is shared online or uploaded to GitHub.

Together, this wiring setup and the webhook interface are the key links between the physical detection and the user-facing notification.

# Bill of Materials (Planned)

This project is designed to stay simple and affordable, using components that are easy to find and safe to use in a classroom or at home. Below is the list of the planned parts needed to build the system, along with a short explanation of what each one does:

| **Component** | **Purpose** |
| --- | --- |
| **Raspberry Pi 5** | Core processing unit — runs Python script, handles GPIO, camera, networking |
| **HC-SR04 Ultrasonic Sensor** | Detects distance to nearby objects; used for motion detection |
| **Camera Module 3 (IMX708)** | Optional image capture when movement is detected |
| **22↔15 Pin Camera Ribbon** | Converts Camera Module 3 (22-pin) to Pi-compatible 15-pin connector |
| **Jumper Wires (M→F)** | Connects the sensor to the GPIO pins on the Raspberry Pi. |

Each item was picked specifically to work well with the Raspberry Pi 5. For example, the HC-SR04 will be powered directly from the Pi’s 3.3V pin, which removes the need for any extra voltage converters or level shifters. This makes the build cleaner and safer, especially for hands-on demonstrations.

The special ribbon cable is needed because the Camera Module 3 uses a newer 22-pin connector, while the Pi itself still uses the traditional 15-pin slot. Without this cable, the camera wouldn’t be able to connect.

Other extras like a breadboard or a small case could be added later to make the build more stable, but the table above shows the essential parts planned during the design phase.

# Verification Strategy (Intended)

To make sure the system behaves as planned, a set of tests has been prepared based on the requirements outlined earlier. Each test links directly to one of the main features or expectations from the design phase. These tests are meant to be practical and realistic — easy to carry out in a classroom or home environment using the final build.

The goal is to confirm that the detection works correctly at different distances, that the cooldown feature behaves as expected, and that the photo feature works properly (or falls back safely when the camera isn’t available). We’ll also observe how long it takes for an alert to appear in Discord after movement is detected.

**Planned Test Cases**

| **Test ID** | **What it Verifies** | **How it Will Be Tested** |
| --- | --- | --- |
| T01 | Distance detection below threshold triggers alert | Place object at ~30 cm and observe alert + log output |
| T02 | No alert sent when object is outside threshold | Place object at ~60 cm and confirm no alert is triggered |
| T03 | Cooldown prevents repeat alerts during cooldown window | Repeated movement at ~30 cm within 30s should only trigger 1 alert |
| T04 | Optional photo is attached when camera is active | Enable SEND\_PHOTO, move object at ~30 cm, verify image is sent |
| T05 | Text-only alert sent when photo mode is ON but camera is unplugged | Simulate unplugged camera and verify graceful fallback (text-only alert) |
| T06 | Basic latency from detection to alert received on Discord | Measure time between object detection and alert visible in Discord channel |

**Traceability Table: Requirements → Tests**

| **Requirement** | **Test ID** |
| --- | --- |
| Detect object within threshold distance | T01 |
| Do not alert when object is outside threshold | T02 |
| Enforce cooldown between alerts | T03 |
| Attach image if photo mode is enabled | T04 |
| Send alert even if camera fails | T05 |
| Alert must arrive within reasonable latency | T06 |

These planned tests are designed to prove that the core functionality works as expected before calling the project complete. The results of these tests will be recorded and analysed in the **Implementation & Testing Report** later on.

# 10. Project Plan Notes

To keep the build process manageable and focused, the project is broken down into a series of clear milestones. Each one targets a key function or feature, making it easier to troubleshoot and verify progress along the way:

* **Wiring Complete** – The HC-SR04 sensor and Camera Module 3 will be safely connected to the Raspberry Pi 5, using the 3.3 V power rail to protect the GPIO pins and keep the range suitable for short-distance detection.
* **Camera Check Passed** – The system will check if the camera is recognised by the Pi and confirm that a test image can be captured using the rpicam-still tool.
* **First Trigger Logged** – Movement detection will be tested by placing an object within the threshold range. A “TRIGGER” event should appear in the console log to show that the sensor and script are working.
* **Cooldown Mechanism Verified** – Repeated movement during the cooldown window should not result in multiple alerts. This will confirm that the time-based gate is working as intended.
* **Photo Attach Function Confirmed** – With SEND\_PHOTO=1 and the camera active, an image should be attached to the alert sent through Discord. This final step ensures that the optional photo feature is functioning properly.

These checkpoints help confirm that both hardware and software are set up correctly before moving into full testing and documentation of results.

# References

The following sources and documentation were used during the design phase to ensure correct hardware handling, safe software setup, and practical configuration of the system:

* Raspberry Pi Documentation. (2023). *Camera support on Raspberry Pi OS Bookworm*. Raspberry Pi Ltd. Available at: https://www.raspberrypi.com/documentation/computers/camera.html
* GPIO Zero Project. (2023). *API Reference and GPIO Backends*. Available at: <https://gpiozero.readthedocs.io/en/stable/api_input.html>
* HC-SR04 Ultrasonic Sensor Datasheet. (n.d.). Available at: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
* Discord Developer Portal. (2023). *Webhook message formatting*. Available at: https://discord.com/developers/docs/resources/webhook
* PiCamera2 Overview (for rpicam-still and libcamera-still commands). Raspberry Pi Ltd. (2023). Available at: https://www.raspberrypi.com/documentation/computers/camera\_software.html
* GitHub – gpiozero Source and Issue Tracker. (2023). For backend support and known compatibility issues with Pi 5: <https://github.com/gpiozero/gpiozero>

These materials guided the system design, component selection, and environmental configurations described in this blueprint.