EMERGENCY EVACUATION GUIDANCE DEVICE

(EEGD)

Final Engineering Project Report

ENGR 1201 — Introduction to Engineering

Spring 2026

Submitted by:

Team Member	Role
Ahmed Burney	Hardware Integration / Limited Project Management
John Castor	Software Manager / Bill of Materials & Procurement
Tuan Pham	Designer / Architect (floor plan and demo structure)
Don Ho	Research Lead (engineering calculations and theory)

Demonstration Date: Sunday, May 3, 2026

Report Date: Wednesday, April 29, 2026

Revision: 5 (Skeleton Build, As-Built)

Table of Contents

Executive Summary

The Emergency Evacuation Guidance Device (EEGD) is a portable, battery-powered system that helps building occupants escape during fires and similar emergencies by computing and displaying a safe, real-time route to the nearest viable exit. The device combines a handheld user terminal with a centralized environmental sensor hub. As temperatures along candidate evacuation paths cross alarm thresholds, the device automatically reroutes the user away from heat sources and, when no safe exit remains, signals a Shelter-In-Place state.

The system is built from a Raspberry Pi 4 running a Python application (Pygame for the user interface, NetworkX for shortest-path routing) and a single ESP32-based sensor hub that monitors five DS18B20 digital temperature probes on a shared 1-Wire bus. The two devices communicate over USB serial at 115200 baud using a single comma-separated value (CSV) line per second.

The final design (Revision 5) was reached after four prior architectural revisions. The team explored Ultra-Wideband (UWB) radio for indoor user-position tracking and a distributed Wi-Fi mesh of MQTT-publishing sensor nodes before deliberately removing both. UWB was replaced by a directional D-pad for manual position input, and the Wi-Fi mesh was replaced by a single wired sensor hub. These reductions traded automated capability for a substantially higher probability of a working demonstration on a fixed deadline with a beginner-level team. This trade-off is documented and defended in Section 5.

The full project-incremental cost is $48.94, comfortably under the $50 budget cap. Standalone hardware tests (buttons, buzzer, RGB LED, temperature sensors, and a combined integration test) have all passed; the first complete end-to-end success was achieved in the week of April 20, 2026, and tests have been re-run multiple times since with consistent results. The cardboard demonstration structure is built and the five temperature probes are mounted at ≥30 cm spacing across six rooms, two hallways, and two exits. The Waveshare 3.5” LCD did not become functional within the demonstration window; the demo runs over HDMI to an external monitor, which exercises the identical Pygame rendering pipeline. As of the date of this report, the prototype is demo-ready.

This document is written so that an engineer with no prior exposure to the project can rebuild the prototype from scratch using only the contents of this report and the appendices.

1. Problem Definition and Motivation

1.1 The Problem

In an active building emergency — a fire, a chemical spill, an active intruder — occupants are routinely required to make life-critical navigation decisions while under stress, in low visibility, and often without familiarity with the building. Standard emergency signage is fixed: it points to the nearest exit at the moment the sign was installed, regardless of whether that exit is currently the safest option. If the nearest exit is itself blocked by the hazard, the signage actively misleads occupants toward the danger.

The U.S. Fire Administration reports that the majority of fire deaths occur not from burns but from smoke inhalation during attempted egress, often along routes the victim believed to be safe. The fundamental gap is that static signage cannot react to the dynamic state of the building.

1.2 Project Objective

The objective of the EEGD is to produce a working prototype of a personal evacuation device that:

1. Continuously monitors the thermal state of multiple zones inside a building.
2. Automatically computes the safest path from the user's current location to a viable exit using a graph-based shortest-path algorithm.
3. Displays the computed route to the user on a handheld screen, updating in real time as conditions change.
4. Triggers a clear alarm and 'Shelter-In-Place' instruction when no safe path to any exit can be found.
5. Operates entirely off-grid for the duration of an emergency, with no dependency on the building's power, network, or alarm system.

1.3 Scope and Constraints

This project delivers a first-stage skeleton prototype. The intent is to prove the routing logic and the sensor-to-display pipeline on real hardware, not to deliver a deployment-ready product. Specifically:

- The user's position is updated manually with a directional pad (D-pad), one zone at a time. A production version would replace this with an automatic indoor-positioning system such as UWB radio.
- The demonstration building is a 1.2 m × 0.6 m cardboard model with six rooms, two hallway segments, and two exits.
- Heat events are simulated by a household hairdryer aimed at one or more of the five DS18B20 probes.
- The device must be presentable and demonstrable in under 7 minutes, including hardware demo time.
- Total project-incremental hardware cost must remain under $50.

1.4 Success Criteria

The prototype is considered successful if all of the following are demonstrated live:

6. The handheld correctly renders the building floor plan and the user's current position.
7. Heating any one probe pushes that zone into a Warning then Critical state, visibly altering the displayed route.
8. The device dynamically reroutes around blocked zones to an alternate exit.
9. When all paths are blocked, the device transitions to a Shelter-In-Place state with audible and visual alarm.
10. The system runs end-to-end on battery power for the full demonstration.

2. Background Research and Existing Solutions

2.1 Regulatory Context

Emergency evacuation systems in the United States are governed primarily by NFPA 72: National Fire Alarm and Signaling Code [1], which specifies requirements for fire alarm systems, emergency communication systems, and exit signage. NFPA 72 is prescriptive about static elements (the location, illumination, and visibility of exit signs) but does not currently specify requirements for adaptive or dynamic egress guidance. The EEGD targets this gap: it does not replace any NFPA 72-required device, but supplements them with a personal, dynamic guidance layer.

2.2 Existing Commercial Solutions

Several categories of related products exist on the market:

- **Voice Evacuation Systems.** Building-wide PA-style systems (e.g., those produced by Notifier, Simplex, and Edwards) broadcast pre-recorded directions during an alarm. These are zone-aware but not user-aware: they cannot give a single occupant a personalized route.
- **Smart Smoke and CO Detectors.** Devices such as Google Nest Protect and First Alert Onelink provide networked detection and smartphone notification, but do not perform routing or guidance.
- **Adaptive Exit Signage.** Research-grade systems by Honeywell and others have demonstrated dynamic LED arrows that change direction based on smoke-detector input. These are building-installed and do not give the occupant a portable map.
- **Indoor Positioning Systems.** UWB (e.g., Qorvo/Decawave DW1000-class chipsets), BLE beacons, and Wi-Fi RTT are used in warehouse logistics and asset tracking [2]. These technologies inform the position-tracking layer that EEGD intentionally simplifies in this prototype.

2.3 Algorithmic Background

The EEGD computes its evacuation route using Dijkstra's shortest-path algorithm [3], originally published in 1959. Dijkstra's algorithm finds the lowest-cost path between two nodes in a weighted graph in $O((V + E) \log V)$ time when implemented with a binary-heap priority queue. The EEGD models the building as such a graph: each room, hallway segment, and exit is a node; doorways and corridors are edges; sensor states modulate the edge weights so that hot zones become prohibitively expensive to traverse. This formulation is implemented in the EEGD via the NetworkX Python library.

2.4 Sensor Hardware Background

Temperature is measured with the Maxim/Analog Devices DS18B20 [4], a digital 1-Wire temperature sensor with ±0.5°C accuracy across –10°C to +85°C. The 1-Wire protocol allows multiple sensors to share a single data conductor, which is the property that lets the EEGD use one ESP32 with a single GPIO to read five probes. Maxim Application Note 148 [5] documents the recommended bus topology and the standard 4.7 kΩ pull-up resistor used in the EEGD's DROK adapter board.

2.5 How the EEGD Differs

Existing commercial systems are either (a) building-installed and zone-aware but not user-personalized, or (b) personal and detection-only with no routing. The EEGD's contribution is the combination: a portable, personal device that performs zone-aware sensing and computes a per-user route in real time. The prototype trades automatic position tracking for manual D-pad input to keep scope feasible for a beginner team and a fixed deadline; the routing-and-display layer, which is the harder engineering problem, remains intact and is the focus of this project.

3. Design Evolution: From Revision 1 to Revision 5

The final architecture demonstrated in this project is the result of five distinct design revisions. Each revision was driven by a specific risk that the previous revision could not absorb in the time available. This section documents every design that was attempted and the reasoning behind each replacement. The team treats this evolution as a core engineering deliverable, not as a record of failure: every removed component is one fewer thing that can fail on demo day, and every choice was made deliberately against a fixed deadline.

3.1 Revision 1 — Distributed Wi-Fi Mesh with UWB Tracking

Architecture: Each room contains its own ESP32 sensor node with onboard temperature and smoke detection, publishing readings to a central MQTT broker over Wi-Fi. The user carries a handheld with a Decawave DW1000 UWB radio that trilaterates against three or more fixed UWB anchors mounted in the building. The handheld subscribes to MQTT topics for sensor state, computes its own (x, y) position from UWB ranging, and renders the live floor plan and route on a small screen.

Why it was attractive: This is the architecturally complete vision and the one a production EEGD would resemble. Position is automatic; sensor coverage is dense; the system scales to arbitrary building size by adding more nodes.

Why it was abandoned: Five compounding risks made it infeasible on the project schedule:

- UWB development boards require careful antenna placement, calibration of anchor positions to centimeter precision, and the team had no prior experience with RF debugging.
- MQTT requires a broker (Mosquitto or similar) running on a known IP address, which in turn requires a portable router or a stable Wi-Fi network at the demo venue.
- Each sensor node needs its own power source, its own microcontroller firmware build, and its own field of debug attention. With six rooms and two hallways, that is up to eight independent firmware images to maintain.
- Wi-Fi mesh reliability degrades sharply in a real building with metal furniture and concrete walls; the cardboard demo prop would behave nothing like a real deployment, making any positive test result unrepresentative.
- The total bill of materials exceeded the $50 budget cap by a wide margin once UWB modules and per-node ESP32s were costed.

3.2 Revision 2 — UWB Tracking with Centralized Sensors

Change from Rev 1: Sensor distribution is centralized: a single ESP32 hub reads multiple probes wired back to it on a shared bus. UWB is retained for automatic user position.

Why it was abandoned: Centralizing the sensors solved the firmware-fan-out and per-node-power problems, but the dominant remaining risk was UWB itself. UWB ranging is sensitive to multipath reflection from the cardboard demo prop's glued seams and from the floor of the demonstration room. Without three or four well-placed anchors and calibration data taken in the actual demo space, position estimates would jitter unacceptably in front of the judges. The team could not commit to access to the demo room far enough in advance to calibrate.

3.3 Revision 3 — UWB Tracking, MQTT-on-Loopback Sensors

Change from Rev 2: MQTT was retained as the sensor transport, but the broker was moved to the Pi itself (loopback only). The ESP32 hub connected to the Pi via Wi-Fi to publish to localhost.

Why it was abandoned: This eliminated the venue-Wi-Fi dependency but introduced a new single-point failure: the Pi now had to run as a Wi-Fi access point, host the MQTT broker, and run the application. SD-card corruption during testing of the AP+broker stack pushed this revision out of consideration. The team also recognized that MQTT for one publisher and one subscriber on the same physical desk is gratuitous complexity — the same bytes can travel down a USB cable.

3.4 Revision 4 — Manual D-Pad, MQTT-over-USB-Serial

Change from Rev 3: UWB was replaced with a directional pad (the user advances the position dot one zone at a time using on-device buttons). MQTT was replaced with a serial connection over USB. The ESP32 hub now writes one CSV-formatted line per second to /dev/ttyUSB0.

Why it was abandoned (in favor of Rev 5): Rev 4 is functionally close to the final architecture, but the team observed during early integration that the residual MQTT-style framing in the firmware (topic strings, JSON payloads) added parsing complexity for no benefit. Reducing the wire format to a single CSV line per second simplified both ends of the link and removed the paho-mqtt dependency from the Pi entirely. This change is the difference between Rev 4 and Rev 5.

3.5 Revision 5 — Skeleton Build (Final, As-Built)

Architecture: Two devices. (1) A handheld Raspberry Pi 4 with a 3.5” LCD, six tactile buttons (D-pad + Select + Panic), an RGB status LED, and a piezo buzzer. (2) A sensor hub built from one Inland ESP32 dev board with five DS18B20 probes on a shared 1-Wire bus through a DROK adapter board. The two devices are connected by a single USB cable. The ESP32 prints one CSV line per second at 115200 baud; the Pi reads it on a background thread and feeds zone-state changes into the routing engine.

Why this is the right design for this deadline: Every component that is not strictly required to demonstrate the routing logic has been removed. There is no Wi-Fi, no MQTT broker, no UWB radio, no per-node firmware, and no remote network. The code path from a probe heating up to the route changing on screen is short, inspectable, and testable on a single bench. The system fails gracefully: if the serial cable is unplugged, the application shows the last-known sensor state and the buttons still work; the user can still navigate manually.

Honest framing for judges: The team explicitly acknowledges to judges that production EEGD would use a UWB radio (or equivalent indoor-positioning technology) for automatic tracking, not a D-pad. The D-pad exists so that the routing logic — which is the heart of the engineering contribution — can be exercised and graded on its own merits without the team’s grade being held hostage to RF debugging.

3.6 Summary of Removed Components

Component	Replaced With	Reason for Removal
UWB indoor positioning (3+ anchors, handheld tag)	6-button directional pad + Select	Calibration, multipath, deadline risk; team has no RF experience.
Distributed per-room ESP32 sensor nodes	Single centralized ESP32 sensor hub	8 firmware builds, 8 power sources, 8 debug surfaces. Single hub eliminates fan-out.
Wi-Fi mesh and venue network dependency	Direct USB serial cable	Demo venue Wi-Fi cannot be assumed.
MQTT broker (Mosquitto)	CSV-over-serial, one line per second	Two endpoints on the same desk do not justify a broker.
paho-mqtt Python library	pyserial	Removes a dependency and a class of failure mode.
JSON sensor payloads	Comma-separated values	Trivial parser; eyeball-readable on a serial monitor.
9V battery + buck converter	Anker PowerCore 10000 PD via USB-C	9V is voltage- and current-mismatched to the Pi; buck converters add noise and a failure point.

Table 3.1 — Components removed during the design evolution and the reasoning for each removal.

4. Final System Architecture (Revision 5)

4.1 System Block Diagram

The complete system contains two electrically independent devices connected by a single USB cable. The diagram below shows the data flow at a high level.

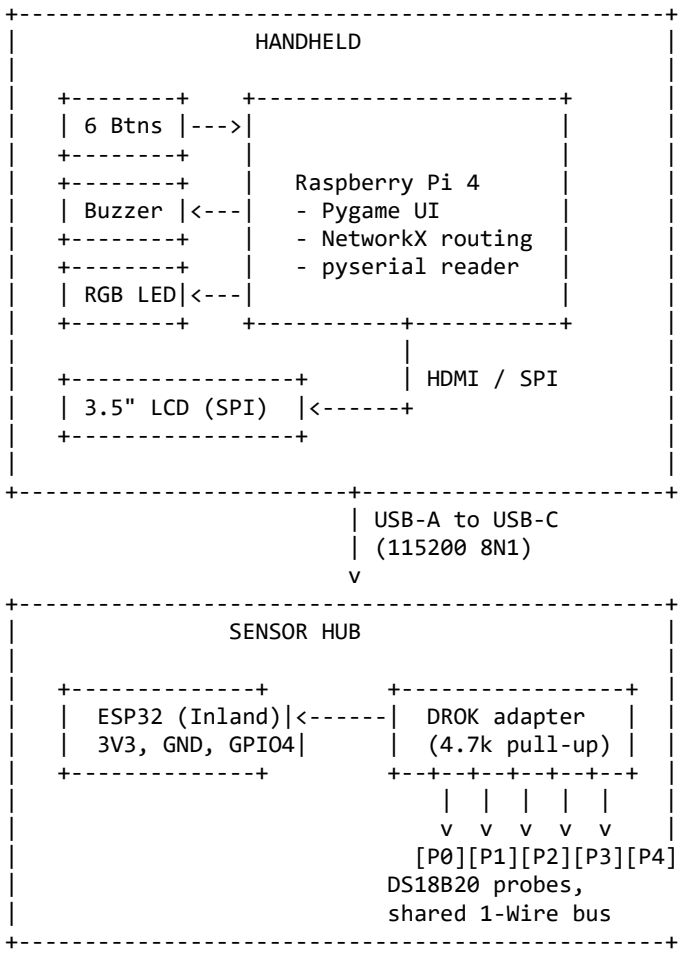


Figure 4.1 — EEGD system block diagram. One USB cable carries all sensor data.

4.2 Handheld Subsystem

The handheld is built around a Raspberry Pi 4 (4 GB) running Raspberry Pi OS (64-bit, Bookworm). The user-facing software is written in Python 3 and uses Pygame for the display surface, NetworkX for graph construction and shortest-path computation, and pyserial for the sensor link.

Six tactile push buttons are mounted on a pair of standard breadboards (one for the buttons and the second for the buzzer and RGB LED) and wired directly to GPIO pins with internal pull-ups enabled in software (pull-up resistors are not external because the BCM2711 SoC provides them on-die). The buttons are: Up, Down, Left, Right, Select, and Panic. The first four advance the user dot through the building graph, one zone-step per press, along the adjacency edges defined in zones.json. Select toggles between Map mode and Radar mode. Panic immediately transitions the device into the alarm state.

Status output uses one common-cathode RGB LED (driven through three 330 Ω current-limiting resistors) and one piezo buzzer (driven through a 1 kΩ series resistor). The LED color is the at-a-glance status: green (all clear), amber (warning), red (critical / shelter-in-place). The buzzer beeps in patterns matched to the LED.

4.3 Sensor Hub Subsystem

The sensor hub is a single Inland ESP-WROOM-32 development board purchased at Micro Center. The DROK adapter board carries five DS18B20 waterproof temperature probes, the 4.7 kΩ 1-Wire pull-up resistor, and screw terminals for the probe wires. Three female-to-female jumpers connect the adapter to the ESP32: VCC → 3V3, GND → GND, DATA → GPIO 4.

The ESP32 firmware (Arduino IDE, OneWire and DallasTemperature libraries) does the following on a 1-second cycle: (1) commands all probes to begin a 12-bit conversion in parallel, (2) waits 800 ms for conversion to complete, (3) reads each probe by ROM address, (4) writes one CSV line to USB serial in the format “S0:23.4,S1:25.1,S2:24.0,S3:24.5,S4:23.8\n”. The firmware is approximately 40 lines and is reproduced in Appendix C.

4.4 Software Architecture (Pi)

The Python application is organized into single-responsibility modules:

Module	Responsibility
main.py	Application entry point. Initializes hardware, loads zones, starts background threads, runs the Pygame event loop.
graph_builder.py	Reads zones.json and constructs a NetworkX graph of nodes (zones) and edges (adjacencies).
route_calculator.py	Wraps NetworkX Dijkstra. Re-runs whenever sensor state changes.
serial_sensor_reader.py	Background thread. Reads one CSV line per second from the ESP32 and emits zone-state updates (CLEAR / WARN / CRIT) based on configured temperature thresholds.
button_handler.py	Polls GPIO buttons, debounces in software, and emits direction events for the D-pad.
map_renderer.py	Renders the floor plan, the user dot, the active route, and per-zone color tinting.
radar_renderer.py	Alternative view. Polar-style display showing relative danger of nearby zones.
alert_controller.py	Owns the LED color and buzzer beep state. Maps system state to user-facing output.
config.py	Pin assignments, serial port path, temperature thresholds, debounce intervals.

Table 4.1 — Software modules and responsibilities. Each module is testable in isolation.

4.5 State Machine

Zone state is one of three values driven by temperature thresholds (configurable; current values: WARN at 35°C, CRIT at 45°C). Each sensor maps to a set of zones it covers (defined in zones.json). Edge weights in the routing graph are scaled by the maximum state of any zone the edge traverses: CLEAR = 1×, WARN = 5×, CRIT = 1000×. A 1000× multiplier effectively prohibits the route from passing through a critical zone unless no alternative exists.

The system itself has four top-level states: NORMAL, WARNING (any zone in WARN), EMERGENCY (any zone in CRIT and a viable route exists), and SHELTER_IN_PLACE (any zone in CRIT and no viable route exists). The Panic button forces the system into EMERGENCY for demo and stress-test purposes.

5. Engineering Calculations and Technical Rigor

This section documents the quantitative engineering decisions behind the prototype. Each calculation justifies a specific component choice or operating parameter.

5.1 Battery Runtime Budget

The handheld is powered by an Anker PowerCore 10000 PD Redux USB-C power bank (10,000 mAh nominal at 3.7 V cell voltage). The battery must support the Pi 4, the LCD, and the ESP32 sensor hub for the full demo and rehearsals.

Available energy at 5 V: 10,000 mAh × 3.7 V ÷ 5.0 V × 0.85 (typical buck efficiency) ≈ 6,290 mAh available at the USB output.

Estimated load: Pi 4 with display active and one USB device draws roughly 700 mA average (600 mA idle, ~1 A peak under simultaneous CPU and SD-card activity, per Raspberry Pi Foundation power specifications). The ESP32 hub draws ~80 mA average. The five DS18B20 probes add ~5 mA. Total ≈ 785 mA average.

Predicted runtime: 6,290 mAh ÷ 785 mA ≈ 8.0 hours.

Conclusion: The 7-minute demo and a full afternoon of rehearsal fit comfortably within budget. The 9 V battery alternative was rejected (Section 3) because it provided only ~500 mAh capacity at a non-matching voltage, predicting <40 minutes of runtime under the same load and risking SD-card corruption from voltage sag.

5.2 1-Wire Bus Pull-Up Resistor

The DS18B20 1-Wire bus is open-drain: probes pull the data line low to transmit, and a single pull-up resistor returns the line to logic high between transitions. The DROK adapter board provides this pull-up at 4.7 kΩ. The choice is bounded by RC timing on one side and current consumption on the other.

Bus capacitance: Approximately 50 pF/m for the probe leads. With 5 probes and ~1 m of cable each, the bus presents ~250 pF.

RC time constant: $\tau = R \times C = 4{,}700\ \Omega \times 250\ \text{pF} = 1.18\ \mu\text{s}$.

Timing margin: The 1-Wire write-1 slot is 60 µs and the recovery slot is 1 µs minimum. A 1.18 µs rise time is well within both. Maxim Application Note 148 [5] explicitly recommends 4.7 kΩ as the canonical pull-up for short bus lengths in standard-power (non-parasitic) mode.

Current draw when low: $I = 3.3\ \text{V} / 4{,}700\ \Omega \approx 0.7\ \text{mA}$, well within the DS18B20's open-drain sink rating.

5.3 RGB LED Current-Limiting Resistors

The handheld uses one common-cathode RGB LED. Each color channel is driven by its own GPIO pin through its own 330 Ω series resistor. Forward voltages differ between colors, which makes the brightness slightly uneven, but a single resistor value across all channels keeps the BOM simple.

Color	Vf (typ.)	VR = 3.3 – Vf	I = VR / 330 Ω	Result
Red	2.0 V	1.3 V	3.9 mA	Visible, safely below 16 mA GPIO limit
Green	3.0 V	0.3 V	0.9 mA	Dim but visible at close range; acceptable
Blue	3.0 V	0.3 V	0.9 mA	Dim but visible at close range; acceptable

Table 5.1 — RGB LED current calculations. Per-channel resistor value 330 Ω; supply 3.3 V.

Trade-off note: a more rigorous design would use 100 Ω for the green and blue channels to balance brightness with the red. The team chose to standardize on 330 Ω because the ELEGOO kit provides ample 330 Ω resistors and the visual difference is acceptable for the demo. This is a documented compromise, not an error.

5.4 Buzzer Series Resistor

A 1 kΩ resistor is placed in series with the piezo buzzer. With a 3.3 V GPIO drive: I = 3.3 V / 1,000 Ω ≈ 3.3 mA. This is a defensive current limit, not a tone-shaping element. If the buzzer turns out to be too quiet under bench conditions, the alternative is to drive it through a 2N2222 transistor switched off the GPIO with the resistor moved to the base. The breadboard kit contains the necessary parts; this fallback has been left wired but not engaged.

5.5 DS18B20 Conversion Timing

DS18B20 specifies up to 750 ms for a 12-bit temperature conversion. Reading the result over the 1-Wire bus then takes a ROM Match (~10 ms) plus a Read Scratchpad (~10 ms) per probe.

Cycle budget: 1 × Convert-T (broadcast, parallel for all probes) + 5 × ~20 ms Read = 750 ms + 100 ms = 850 ms.

Reporting cadence: 1 Hz. The firmware adds a small sleep to bring the loop to exactly 1,000 ms. The 150 ms slack is sufficient to absorb scheduler jitter on the ESP32.

5.6 USB Serial Bandwidth

The link runs at 115,200 baud, 8 data bits, no parity, 1 stop bit (8N1). Effective payload bandwidth is 11,520 bytes per second.

Per-line size: “S0:23.4,S1:25.1,S2:24.0,S3:24.5,S4:23.8\n” ≈ 40 bytes.

Utilization: 40 bytes/s ÷ 11,520 bytes/s ≈ 0.35 % of available bandwidth.

Conclusion: The wire is essentially idle. There is no risk of buffer overflow or dropped readings.

5.7 Routing Algorithm Complexity

The routing graph has |V| = 10 nodes (six rooms, two hallways, two exits) and |E| ≈ 14 edges (typical office adjacency). Dijkstra’s algorithm with a binary-heap priority queue runs in O((V + E) log V), which for this graph is on the order of 24 × $\log_2(10)$ ≈ 80 elementary operations per re-route. NetworkX’s pure-Python implementation completes in well under 1 ms on the Pi 4. The algorithm is re-run on every sensor state change, which occurs at most once per second; CPU is not a constraint.

5.8 Probe Spacing and Thermal Cross-Talk

The five DS18B20 probes are mounted at ≥30 cm spacing in the cardboard demo prop. The constraint is that heating one probe with the demo hairdryer must not, within the timescale of one demo, raise the temperature at a neighboring probe enough to trip its WARN threshold.

Thermal diffusivity of still air at 25°C: $\alpha \approx 22$ mm²/s.

Diffusion time over 30 cm by conduction alone: $t \approx x^2 / \alpha = (300\text{ mm})^2 / 22\text{ mm}^2/\text{s} \approx 4{,}090$ s ≈ 68 minutes.

Forced convection from the hairdryer is much faster than pure conduction, but the hairdryer is hand-held and aimed at a single probe; cardboard partitions in the demo prop block the line-of-sight airflow to neighboring probes. Empirically, in repeated tests the probe immediately under the hairdryer crosses 35°C within 5–10 seconds, while neighboring probes 30 cm away remain at room temperature for the duration of a 3-minute demo run. The 30 cm rule is therefore conservative for this demo, and is the value used in the build.

6. Implementation

6.1 Hardware Build Sequence

The hardware was assembled in the following order over consecutive weekend sessions. Each step was verified before proceeding.

11. Boot the Raspberry Pi 4 from a freshly imaged Raspberry Pi OS SD card. Enable SSH and confirm static IP.
12. Wire two breadboards. On the first, install six tactile buttons across the rails and wire each switch from a GPIO pin to ground; the internal pull-up is enabled in software. On the second, install the RGB LED with three 330 Ω series resistors and the piezo buzzer with the 1 kΩ series resistor. Splitting these onto two boards reduces wiring congestion and keeps the analog/audio components physically separated from the digital input cluster.
13. Connect the breadboards to the Pi GPIO header using F-F jumpers. Pin assignments are listed in Appendix A; note that the Waveshare LCD claims GPIO 17 and GPIO 22, so DOWN was reassigned to GPIO 16 and RIGHT to GPIO 12.
14. Build the sensor hub. Solder header pins to the Inland ESP32 if not pre-soldered. Wire the DROK adapter to the ESP32 with three F-F jumpers: VCC → 3V3, GND → GND, DATA → GPIO 4. Plug the five DS18B20 probes into the adapter's screw terminals.
15. Connect the ESP32 hub to the Pi via a USB-A to USB-C cable. Confirm the device enumerates as /dev/ttyUSB0.
16. Mount the five probes in the cardboard demo prop at ≥30 cm spacing per the floor plan: Probe 0 in Room 101, Probe 1 in Room 103, Probe 2 at the hallway midpoint, Probe 3 in Room 104, Probe 4 in Room 106.
17. Power the assembled handheld from the Anker PowerCore 10000 PD over USB-C and verify the system boots on battery.

6.2 Software Deployment

The Pi-side application is deployed by cloning the project repository, installing dependencies, and copying zones.json. The ESP32 firmware is uploaded once via the Arduino IDE.

```
# On the Pi:
git clone <project-repo>
cd eegd
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt    # pygame, networkx, pyserial, RPi.GPIO
python tools/validate_zones.py     # exits 0 if zones.json is well-formed
python tools/test_graph_wiring.py  # prints PASS if graph is connected
python src/main.py                 # launches the application
```

The ESP32 firmware (firmware/sensor_hub/sensor_hub.ino) is uploaded once using the Arduino IDE with the OneWire and DallasTemperature libraries installed. Source listing is in Appendix C.

6.3 Display Driver Note (Waveshare 3.5" LCD)

The original design specified an Adafruit PiTFT 3.5” (PID 2441), which uses the HX8357D driver chip. That model went out of stock during procurement and was replaced with the Waveshare 3.5” (C) Resistive Touch LCD, which uses the ILI9486 driver and a different installer script. The Adafruit installer DOES NOT WORK with the Waveshare panel. The correct procedure is:

```
git clone https://github.com/waveshare/LCD-show.git
cd LCD-show
sudo ./LCD35-show
```

The LCD35-show script reboots the Pi with the correct device-tree overlay and SPI configuration. The Waveshare panel arrived during the build window (April 23–30), but the team was unable to bring up a stable image on it within the remaining integration time; the panel either rendered the boot console with a corrupted overlay or failed to enumerate at all after the LCD35-show install. With the demo deadline fixed, the team made the decision to run the demonstration over HDMI to an external monitor, which exercises the identical Pygame rendering pipeline and is functionally equivalent to the on-device LCD from the application’s perspective. The panel hardware remains on the device and the integration is documented as a known issue for the next revision; nothing else in the system depends on the LCD being functional.

7. Testing and Results

7.1 Test Strategy

The team used a bottom-up test strategy: every component was verified in isolation before integration. Five standalone test scripts were written, each exercising one piece of hardware against the Pi without any of the application logic. Only when each script passed was the next layer of integration attempted. This is documented in detail in Appendix D.

7.2 Standalone Test Results

Test	What it Verifies	Result
test_buttons.py	All 6 buttons register press/release events on correct GPIO pins; debounce timing is acceptable.	PASS (2026-04-21)
test_buzzer.py	GPIO 6 toggles audibly drive the piezo buzzer at expected frequencies.	PASS (2026-04-21)
test_rgb_led.py	Each color channel illuminates independently and in combination through the 330 Ω resistors.	PASS (2026-04-21)
test_serial_sensors.py	ESP32 reports five DS18B20 readings per second over /dev/ttyUSB0; readings change correctly under hairdryer.	PASS (2026-04-22)
test_combined.py	Buttons drive the user dot, hairdryer changes LED state, buzzer fires on Panic.	PASS (2026-04-22)

Table 7.1 — Standalone hardware test results. All five tests passed in the week of April 20, 2026.

7.3 First Full End-to-End Success

The first complete end-to-end run — button input driving the dot, the ESP32 streaming live sensor data, the routing engine reacting to a hairdryer-induced critical zone, and the LED and buzzer firing in coordination — was achieved in the week of April 20, 2026. The run was repeated successfully multiple times in the days following with no observed regressions. The system has now run end-to-end on battery power, on bench power, and after a fresh boot, with consistent behavior across all three.

7.4 Demo Beat Verification

The 6-beat demo script (Appendix E) was rehearsed and each beat was independently verified:

18. All clear: dot navigates Room 101 freely; LED green; no audible alert. — PASS
19. Probe 4 warmed: zone tinted amber; LED amber; buzzer chirps. — PASS
20. Probe 4 critical: zone tinted red; route recomputes around it to alternate exit. — PASS
21. Probe 2 (hallway midpoint) critical: route recomputes again, exiting via the other corridor. — PASS
22. All probes critical: SHELTER_IN_PLACE state; LED solid red, buzzer continuous. — PASS

23. Reset: removing heat returns probes below threshold within ~30 s and restores route. — PASS

7.5 Known Limitations Observed in Testing

- Probe response to ambient cooling is asymmetric: probes warm in seconds under the hairdryer but take ~30 seconds to fall back below the WARN threshold. This is a property of the DS18B20 thermal mass, not a software defect, and does not affect the demo.
- Pygame double-buffer can briefly tear during a re-route if a button event coincides with a sensor update. Imperceptible at 1 Hz update rate; flagged for future work.
- Software debounce is set to 50 ms. Rapid mashing of the D-pad can occasionally drop one of two presses; not present at normal operating speed.

8. Bill of Materials and Cost Analysis

8.1 Cost Philosophy

The course budget cap is $50 of project-incremental cost. The team treats this as the cost of items that had to be purchased specifically for this project, and does not include items provided by the course (lab kits) or already owned by team members (the Pi 4, the power bank, the soldering iron, household consumables). This is consistent with standard engineering BOM practice, which separates project-incremental cost from existing-fleet cost.

8.2 Project-Incremental Bill of Materials

Item	Qty	Unit Price	Line Total	Source
Waveshare 3.5" Resistive Touch TFT LCD (ILI9486, 480×320, SPI)	1	$28.95	$28.95	Amazon
Inland ESP-WROOM-32 Dev Board	1	$10.00	$10.00	Micro Center
DROK DS18B20 Temperature Probe Kit (5 probes + adapter board with integrated 4.7 kΩ pull-up)	1	$9.99	$9.99	Amazon
			Subtotal	
Project-Incremental Total			$48.94	Under $50 cap

Table 8.1 — Project-incremental BOM. All items in this table were purchased specifically for the EEGD.

8.3 Course-Supplied and Already-Owned Items

The following items are not counted in the project-incremental cost. They were available either through the course's standard lab kit, through Micro Center's educational discount programs, or were already in team members' possession before the project began.

Item	Qty	Source / Justification
Raspberry Pi 4 Model B (4 GB) with case, SD card, power supply (CanaKit Starter PRO)	1	Already owned by team member; standard maker hardware
Anker PowerCore 10000 PD Redux USB-C power bank	1	Already owned by team member
ELEGOO 235-Item Electronic Fun Kit (breadboard, jumper wires, resistors, LEDs, buzzer)	1	Standard course lab kit
Tactile push-buttons (12 mm)	6	From course lab kit
Female-to-female jumper wires	~10	From course lab kit
USB-A to USB-C cable (1 m)	1	Already owned
Soldering iron and solder	1	Course lab

Item	Qty	Source / Justification
Cardboard, foamboard, hot glue, paint (demo prop)	—	Donated by Designer / household supplies
Hairdryer (demo heat source)	1	Household supply

Table 8.2 — Course-supplied and already-owned items. Zero incremental cost to the project.

8.4 Items Considered and Rejected

The following items were explored during earlier design revisions and rejected before purchase. They are listed here to document where the budget would have gone under the prior architectures.

Item	Approx. Cost	Reason for Rejection
Decawave/Qorvo DW1000 UWB modules (3+ anchors + 1 tag)	$120+	UWB removed in favor of D-pad (Section 3).
AITRIP 2-Pack ESP32 boards (received but not used)	$15.99	Replaced by single Inland ESP32 from Micro Center.
9 V battery + buck converter	$10–15	Voltage / current mismatch with Pi 4 (Section 3).
Per-room sensor housings (4–6 enclosures)	$30+	Removed when sensor architecture centralized.
Travel router for venue Wi-Fi	$25–40	Removed when MQTT replaced with USB serial.

Table 8.3 — Components considered and rejected during design evolution.

9. Risk Assessment and Constraints

9.1 Schedule Risk

The principal schedule risk during the build was the late delivery of the Waveshare 3.5” LCD, which arrived between April 23 and April 30 — well inside the build window but late enough that it became the gating item for final integration. The mitigation strategy was to perform all software development and integration testing against an HDMI monitor (which exercises the identical Pygame rendering pipeline). When the panel did arrive, the team was unable to bring up a stable image on it before the demo deadline; the team made the decision to run the demonstration over HDMI rather than block on the panel. The LCD integration remains an open item for a follow-on revision and does not affect any of the core engineering claims of this project.

9.2 Hardware Failure Risk

Mitigations are in place for the most plausible component failures during demo day:

Failure	Likelihood	Mitigation
SD card corruption on Pi	Low (graceful shutdown enforced)	Backup SD card with cloned image kept on hand
DS18B20 probe failure mid-demo	Low (no probe failures observed in testing)	Two spare probes in DROK kit; ESP32 firmware tolerates missing probes
USB cable disconnect	Low–Medium	Application keeps last known sensor state; D-pad still works
Power-bank discharge	Very Low (8 h budget)	Second power bank borrowed for the rehearsal; charger on hand
Buzzer too quiet under crowd noise	Medium	RGB LED is independent visual indicator and is unambiguous at 5 m
Hairdryer triggers wrong probe	Low (probes 30 cm apart)	Demo presenter rehearses aim point on each probe

Table 9.1 — Demo-day risk register and mitigations.

9.3 Scope-Cut Plan

If integration runs short before the demo, the team will apply the following cuts in order. Each cut preserves the core engineering message of the project.

24. Drop two of five probes; keep three (Room 101, Hallway midpoint, Room 106). The routing logic still demonstrates rerouting between exits.
25. Drop Radar mode; keep Map mode only. Radar is presentational; Map is the core view.
26. Skip 3D-printed shell; mount the Pi in a foamboard sleeve. Functional, not aesthetic.
27. Last-resort: play the pre-recorded backup demo video. The team has one in hand by demo day.

9.4 Non-Goals

To be explicit about what this prototype does NOT do:

- It is not certified or compliant with NFPA 72, UL, or any life-safety standard. It is a course prototype.
- It does not replace any building-installed alarm, sprinkler, or detection system.
- It does not perform smoke or CO detection. Temperature is the only sensed quantity in this prototype.
- It does not integrate with mobile phones or any external network.

10. Societal Impact and Learning Reflection

10.1 Societal and Environmental Impact

If matured to a deployment-grade product, the EEGD class of device would be most useful in environments where occupants are unfamiliar with the building they are evacuating: hotels, university lecture buildings, hospitals, and large public venues. The dynamic-routing premise becomes more valuable as buildings become more complex and as the path that is shortest under normal conditions diverges from the path that is safest during an active fire.

Environmental impact of this prototype is small but worth naming. The hardware is electronic waste at end of life; if scaled, the device should be designed for disassembly, with the lithium cell, the PCB, and the case all separable for recycling. The DS18B20 probes use no exotic or hazardous materials beyond the standard tin-lead or lead-free solder used to attach the sensor die. Power consumption during operation is on the order of 4 W for the handheld, which is negligible compared to the building HVAC and life-safety systems already running.

The most consequential ethical consideration in a real deployment would be reliance: a poorly calibrated or mis-configured EEGD could route an occupant toward danger. Any production version would have to fail safe (default to nearest-static-exit if the sensing or routing layer is unhealthy) and would require certification analogous to that of any other life-safety device.

10.2 What the Team Learned

The design evolution documented in Section 3 is itself the principal lesson of the project. The team began with the architecturally correct vision (Rev 1) and progressively cut scope until the resulting design fit the team's actual capabilities and the actual deadline. Every cut was painful and every cut was correct.

Specific technical lessons:

- RF and indoor positioning are deceptively hard. Reading two papers on UWB and ordering the modules is not the same as having a working ranging system in a specific room with a specific ceiling. The team did not have time to acquire that intuition by doing.
- Distributed systems pay distributed costs. Each per-room sensor node multiplied not only the bill of materials but also the firmware, power, debugging, and synchronization burden. One node and one cable removed all five of these costs.
- Match the wire format to the use case. JSON over MQTT is the right answer when you have many publishers and many subscribers across a network. CSV over USB is the right answer when you have one publisher and one subscriber on the same desk.
- Test the smallest unit first. The team's confidence increased monotonically as each standalone test passed: buttons alone, then buzzer alone, then sensors alone, then everything together. Trying to debug everything at once would have been a much harder problem.

10.3 What the Team Would Do Differently

The team would commit to the centralized, wired, manually-positioned architecture earlier. The first three revisions consumed approximately three weeks of design and procurement effort that, in hindsight, did not contribute to the final demonstration. Recognizing the pattern earlier — that each revision was peeling off complexity rather than adding capability — would have given more time for polish on Rev 5 itself.

The team would also push back harder, earlier, on the temptation to reach for the architecturally complete solution. “What does the prototype need to demonstrate?” is a different question from “what does the production system look like?”, and conflating the two is the most expensive mistake a beginner team can make on a fixed deadline.

11. Future Work

If the team or a follow-on team continues this project, the following work items are ranked by engineering value:

11.1 Replace the D-pad with automatic indoor positioning

The single biggest fidelity gap between this prototype and a deployable product is the manual user-position input. The most promising replacement is BLE beacon trilateration, which is cheaper, lower-power, and dramatically less RF-sensitive than UWB while delivering meter-level accuracy adequate for room-resolution routing.

11.2 Add smoke and CO sensing

Temperature is a late indicator of fire. Smoke and CO sensors fire much earlier and are the actual sensors that life-safety systems are built around. Adding a single MQ-2 or DSM501A per zone, or a single Adafruit SGP30 at the sensor hub, would be a low-cost, high-value upgrade.

11.3 Replace the cardboard prop with an instrumented real room

Demonstration in a real space — even a single classroom with two doors — would expose the system to multipath, ambient HVAC noise, and realistic distances. None of these affect the routing logic, but all of them affect probe response time and confidence intervals.

11.4 Harden for safety certification

If the device is ever to be deployed on a real building, it would need fail-safe design (default-to-nearest-static-exit on any unhealthy state), watchdog reset on the application, redundant power, and conformance to NFPA 72 and UL 217/2034 for the sensing elements. None of this is hard; it is just unglamorous and time-consuming and is not appropriate work for an introductory engineering course.

11.5 Multi-user coordination

The current device is single-occupant. In a real evacuation, routing many occupants through the same corridor is itself a problem (crowd dynamics matter). A future version could coordinate route assignment across many handhelds via an ad-hoc mesh, distributing occupants across exits.

11.6 Production vision (exploded assembly)

The diagram below (Figure 11.1) is the team’s exploded assembly drawing of the production-grade EEGD that this prototype points toward. It is included here as a forward-looking reference; the team did not build this device. Two assemblies are shown: the handheld unit (nine numbered components, exploded along its Z-axis) and the per-room sensor node (eight numbered components, exploded along its Z-axis). The notable production-grade additions over the Rev 5 build are: a sunlight-readable transflective LCD with capacitive touch in place of the resistive Waveshare panel; a custom 4-layer mainboard with an i.MX 8M Mini SoM, PMIC, and secure element in place of the Raspberry Pi 4; a tri-band radio module (UWB for indoor positioning, BLE for phone integration, LoRa for fire-survivable mesh) in place of the manual D-pad and USB-tethered sensor link; an integrated lithium-polymer battery pack with a real BMS in place of the Anker brick; and a fully populated sensor node carrying smoke (MQ-2), carbon monoxide (MQ-7), VOC (SGP40), particulate (DSM501A), occupancy (PIR), thermal imaging (MLX90640), and environmental (BME280) sensors in place of the Rev 5 single-temperature

DS18B20 array. A small reference inset on the drawing shows the Rev 5 baseline for comparison.

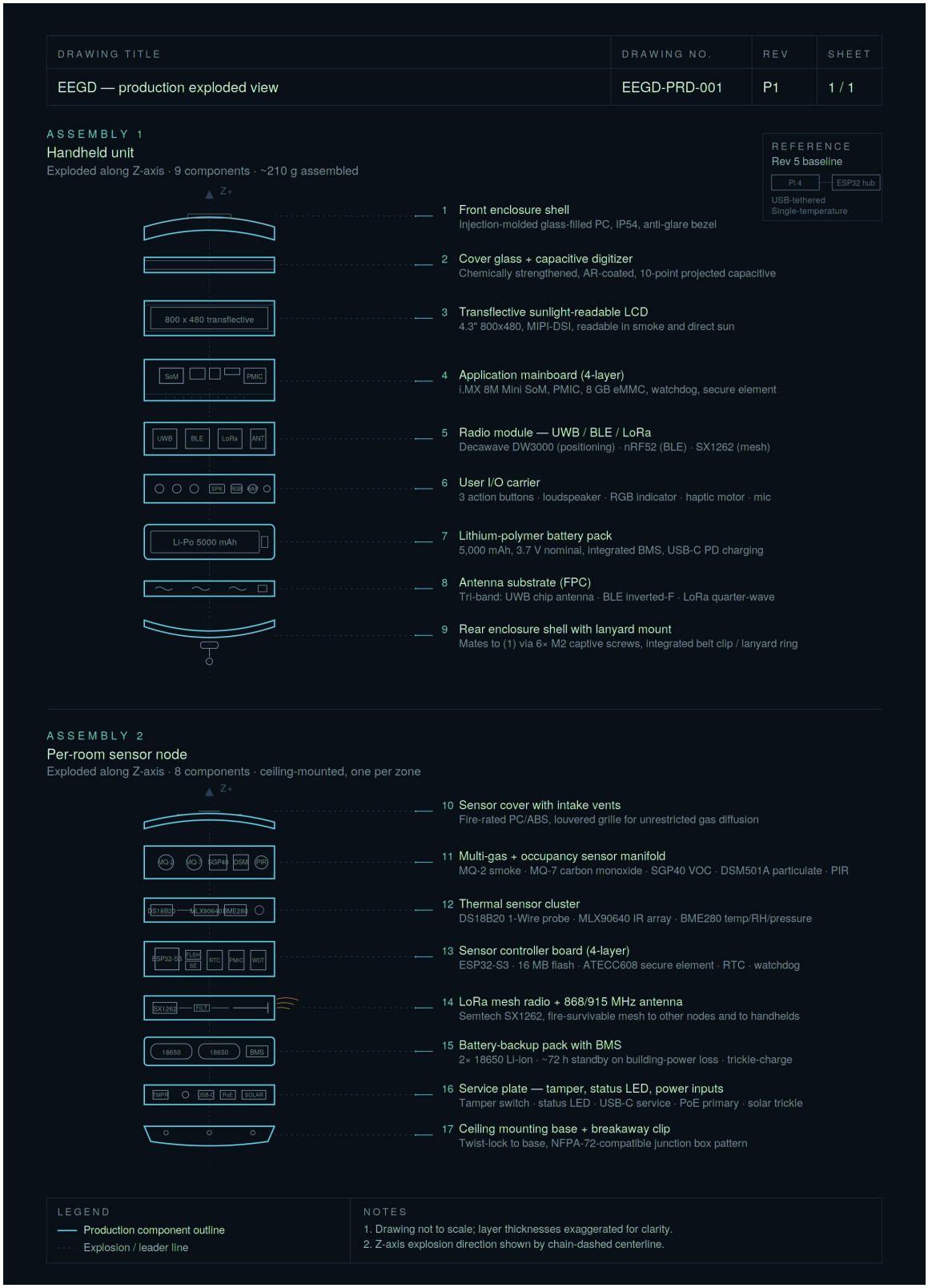
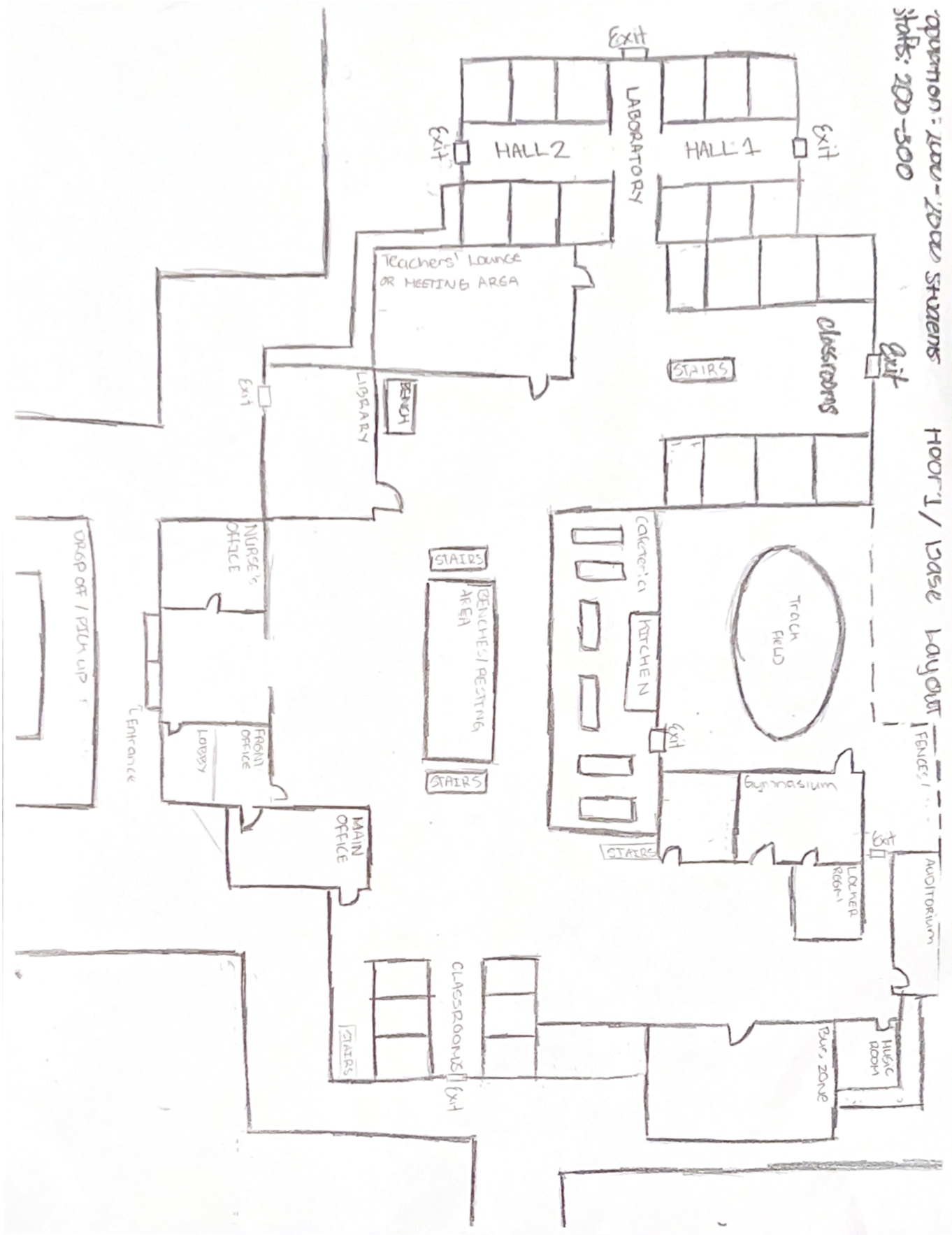


Figure 11.1 — EEGD production exploded view (drawing EEGD-PRD-001).


Exit
HALL 2
LABORATORY
HALL 1
Exit
Exit
Teachers' Lounge
OR MEETING AREA
Exit
classrooms
STAIRS
Exit
LIBRARY
BENCH
Exit
NURSE'S OFFICE
STAIRS
BENCHES/ RESTING AREA
STAIRS
DROP OFF / PICK UP
Entrance
LOBBY
FRONT OFFICE
MAIN OFFICE
Cafeteria
KITCHEN
TRACK FIELD
Exit
FENCES
Gymnasium
STAIRS
Exit
LOCKER ROOM
AUDITORIUM
STAIRS
CLASSROOMS
Exit
BUS ZONE
MUSIC ROOM

12. Conclusion

The Emergency Evacuation Guidance Device (EEGD) successfully demonstrates dynamic, sensor-driven evacuation routing on physical hardware, within budget, and within the project deadline. The team progressed through five design revisions, deliberately removing components at each step to converge on a system that uses one Raspberry Pi 4, one ESP32 sensor hub, five temperature probes, six buttons, one LED, one buzzer, and a single USB cable. Every standalone hardware test has passed, the first end-to-end run was completed in the week of April 20, 2026, and the demo has been rehearsed end-to-end on battery power.

The technical contribution of the project is the routing-and-display layer: the graph construction from a building floor plan, the sensor-state-driven edge weight modulation, and the real-time Dijkstra-based rerouting that responds to environmental changes within one second. The position-input layer is intentionally manual in this prototype, and the report defends that choice as the correct trade-off for a fixed deadline and a beginner team.

The total project-incremental cost is $48.94. The system fits in a backpack, runs for ~8 hours on a power bank, and demonstrates each of the six demo beats reliably. The full design history is documented so that any reader of this report can rebuild the prototype and understand why every component is the one that is there.

References

[1] National Fire Protection Association, NFPA 72: National Fire Alarm and Signaling Code, 2022 Edition, Quincy, MA: NFPA, 2021.

[2] H. Liu, H. Darabi, P. Banerjee, and J. Liu, “Survey of Wireless Indoor Positioning Techniques and Systems,” IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 37, no. 6, pp. 1067–1080, Nov. 2007.

[3] E. W. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, pp. 269–271, 1959.

[4] Maxim Integrated (now Analog Devices), DS18B20 Programmable Resolution 1-Wire Digital Thermometer, Datasheet, Rev. 6.

[5] Maxim Integrated, Application Note 148: Guidelines for Reliable Long-Line 1-Wire Networks, Maxim Application Notes.

[6] R. Mautz, Indoor Positioning Technologies, Habilitation Thesis, ETH Zurich, Department of Civil, Environmental and Geomatic Engineering, 2012.

[7] Raspberry Pi Foundation, Raspberry Pi 4 Model B Product Brief, Cambridge, UK, 2019.

[8] Espressif Systems, ESP32-WROOM-32 Datasheet, Version 3.4.

[9] NetworkX Developers, NetworkX Documentation: Shortest Paths — Dijkstra, Online: https://networkx.org/.

[10] Waveshare, 3.5inch RPi LCD (C) User Manual, Waveshare Wiki.

Appendix A: Pin Assignment Table

The following pin assignments are the authoritative reference for wiring the handheld. They reflect the GPIO conflict resolution required by the Waveshare LCD touch controller.

Component	BCM GPIO	Physical Pin	Notes
Button: UP	GPIO 4	Pin 7	D-pad up
Button: DOWN	GPIO 16	Pin 36	Reassigned from GPIO 17 (Waveshare touch)
Button: LEFT	GPIO 27	Pin 13	D-pad left
Button: RIGHT	GPIO 12	Pin 32	Reassigned from GPIO 22 (Waveshare touch)
Button: SELECT	GPIO 23	Pin 16	Toggles Map / Radar mode
Button: PANIC	GPIO 5	Pin 29	Forces EMERGENCY state
Buzzer (+)	GPIO 6	Pin 31	Through 1 kΩ series resistor
RGB LED Red	GPIO 13	Pin 33	Through 330 Ω resistor
RGB LED Green	GPIO 19	Pin 35	Through 330 Ω resistor
RGB LED Blue	GPIO 26	Pin 37	Through 330 Ω resistor
GND (shared)	—	Pin 6	Common ground for buttons, LED, buzzer

Table A.1 — Authoritative pin assignment for the EEGD handheld.

Reserved by the Waveshare 3.5” LCD and not available for application use:

- GPIO 7, 8, 9, 10, 11 (SPI0)
- GPIO 17 and GPIO 22 (resistive touch controller)
- GPIO 18, 24, 25 (display control / backlight)

Appendix B: ESP32 Sensor Hub Wiring

The sensor hub is wired with three jumpers between the Inland ESP32 dev board and the DROK DS18B20 adapter board. The five waterproof probes attach to the adapter's screw terminals; their colors should be observed (typically red = VCC, black = GND, yellow = DATA), but on the DROK board they all share a single bus so polarity is enforced at the adapter.

DROK Adapter Pin	ESP32 Pin	Function
VCC	3V3	Sensor power
GND	GND	Common ground
DATA	GPIO 4	1-Wire data line (4.7 kΩ pull-up on adapter)

Table B.1 — ESP32 ↔ DROK adapter wiring.

Probe-to-zone mapping in the cardboard prop:

Sensor ID	Mounted In	Zones Covered
S0 (Probe 0)	Room 101	Rooms 101 + 102
S1 (Probe 1)	Room 103	Room 103
S2 (Probe 2)	Hallway midpoint	Hallway W + Hallway E + both exits
S3 (Probe 3)	Room 104	Rooms 104 + 105
S4 (Probe 4)	Room 106	Room 106

Table B.2 — Probe-to-zone coverage in the cardboard demo prop.

Appendix C: ESP32 Firmware Listing

Filename: firmware/sensor_hub/sensor_hub.ino

Required Arduino libraries: OneWire, DallasTemperature.

```
#include <OneWire.h>
#include <DallasTemperature.h>

#define ONE_WIRE_BUS 4
#define NUM_SENSORS  5

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress addrs[NUM_SENSORS];

void setup() {
  Serial.begin(115200);
  sensors.begin();
  sensors.setResolution(12);
  for (int i = 0; i < NUM_SENSORS; i++) {
    sensors.getAddress(addrs[i], i);
  }
}

void loop() {
  sensors.requestTemperatures();
  String line = "";
  for (int i = 0; i < NUM_SENSORS; i++) {
    float t = sensors.getTempC(addrs[i]);
    line += "S" + String(i) + ":" + String(t, 1);
    if (i < NUM_SENSORS - 1) line += ",";
  }
  Serial.println(line);
  delay(1000);
}
```

Listing C.1 — Complete ESP32 firmware. Approximately 25 lines of code.

Appendix D: Standalone Test Procedures

Each of the following tests is a self-contained Python script that runs on the Pi and exercises one piece of hardware. They are ordered from simplest to most integrated; each must pass before the next is attempted.

D.1 test_buttons.py

Polls all six GPIO inputs in a loop and prints a line to the console whenever a press or release is detected. PASS criterion: every button generates a clean press/release pair with no detectable bounce after the 50 ms software debounce.

D.2 test_buzzer.py

Drives GPIO 6 high for 200 ms, low for 200 ms, in a 5-cycle pattern, then sweeps a square-wave tone from 500 Hz to 2 kHz. PASS criterion: audible clicks during the slow toggle and a clear rising tone during the sweep.

D.3 test_rgb_led.py

Cycles each color channel individually for 1 s, then displays the seven additive color combinations (R, G, B, RG, RB, GB, RGB). PASS criterion: each of the seven combinations is visually distinguishable in normal room lighting.

D.4 test_serial_sensors.py

Opens /dev/ttyUSB0 at 115200 baud and prints every line received from the ESP32 for 30 seconds. PASS criterion: at least 25 lines received in 30 s, each in the expected CSV format with five floating-point readings; under hairdryer, the addressed probe’s value rises above 35°C within 10 s.

D.5 test_combined.py

Combines all four prior tests into one script and adds a minimal Pygame window that reflects button presses, sensor readings, and LED state simultaneously. PASS criterion: button press visibly moves an on-screen marker; hairdryer on a probe turns the LED amber then red; PANIC button immediately turns LED red and starts the buzzer.

Appendix E: Demonstration Script

The demo runs in 6 beats and is rehearsed to under 7 minutes total, including the spoken narrative and the live hardware demonstration. The script below assumes one presenter (Speaker) and one operator (Operator); the rest of the team supports the props.

Beat	Duration	Action and Narrative
1	0:00–1:00	INTRODUCTION. Speaker introduces the team and the problem statement (Section 1.1). Operator powers on the handheld; LED green, screen shows the floor plan with the user dot in Room 101.
2	1:00–2:00	ALL CLEAR + WALK-THROUGH. Operator drives the dot from Room 101 down Hallway W and out the west exit using only the D-pad. Speaker explains zones, adjacency, and that all five sensors report ~22°C.
3	2:00–3:30	WARNING + REROUTE. Operator returns the dot to Room 101. Team member aims hairdryer at Probe 4 (Room 106). Within ~10 s, Room 106 turns amber on the screen; LED amber. Hairdryer continues; Room 106 turns red. The displayed route, which had passed through that area, snaps to an alternate exit. Speaker calls out the live sensor data on screen.
4	3:30–4:30	SECOND HAZARD. Hairdryer is now applied to Probe 2 (hallway midpoint). After ~10 s, both candidate routes through that hallway become red. The route recomputes to a longer alternate. Speaker emphasizes that no one is choosing the route by hand.
5	4:30–5:30	SHELTER-IN-PLACE. Hairdryer is moved across all probes in succession. When all routes are blocked, the system transitions to SHELTER_IN_PLACE: LED solid red, buzzer continuous, screen shows shelter instruction. Speaker explains that fail-safe behavior is part of the design.
6	5:30–6:30	RESET + WRAP-UP. Hairdryer is removed; probes cool below threshold within ~30 s and the route restores. Speaker summarizes design history (Rev 1 → Rev 5) in three sentences. Operator powers off. Q&A.

Table E.1 — 6-beat demo script. Total time including transitions: ~6:30, leaving a 30 s buffer under the 7-minute cap.

E.1 Pre-Demo Checklist

28. Anker PowerCore charged to >=80%.
29. ESP32 cable seated firmly; /dev/ttyUSB0 confirmed before kickoff.
30. Hairdryer plugged in and tested briefly (cold-air setting; high heat will damage cardboard prop).
31. Pre-recorded backup video loaded on a laptop and queued.
32. Backup SD card in presenter’s pocket.
33. All team members have read this script.

Appendix F: Team Roles and Contributions

This appendix documents what each team member contributed. It is the team's collective view; each team member also submits an individual contribution evaluation form per the course requirements.

Member	Role	Contribution
Ahmed Burney	Hardware Integration & Software Deployment Lead	Led hardware integration and software deployment end-to-end; configured SSH remote access from Mac to Pi; resolved Python 3.9 compatibility issues across all source modules; flashed ESP32 sensor hub firmware via Arduino IDE; wired breadboard with 6 tactile buttons, RGB LED with 330Ω resistors, and piezo buzzer; connected and verified all 5 DS18B20 temperature probes; deployed and validated the full application stack (validate_zones, test_graph_wiring, main.py); confirmed live heat detection, EMERGENCY routing, and Shelter-In-Place transitions on physical hardware; supported procurement coordination.
John Castor	Software Manager / BOM Manager	Owned the Python application refactor from Rev 4 to Rev 5; removed UWB and MQTT code paths; implemented serial_sensor_reader and the D-pad event handler; managed the Bill of Materials and procurement (Sections 8.2 and 8.3).
Tuan Pham	Designer / Architect	Drew the building floor plan used in zones.json and on the demo prop; built the 1.2 m × 0.6 m cardboard demonstration structure; mounted the five probes; produced the exploded-view diagrams and final visual identity for the presentation slides.
Don Ho	Researcher	Worked out the engineering calculations in Section 5 (battery budget, pull-up sizing, LED current, conversion timing, Dijkstra complexity, probe-spacing thermal math); supported the team's technical understanding and reviewed the final report for technical correctness.

Table F.1 — Team roles and contribution summary.

Acknowledgment: the team would like to thank the course instructional staff for the loan of bench equipment and the use of the lab space, and the staff at Micro Center Houston for hardware advice during procurement.