Bill-E Focus Robot - Final Report

CM3040 - Physical Computing and the Internet of Things

Student number: 2[illegible]

Abstract

The Bill-E Focus Bot is an environmental and personal activity monitoring system with a focus on developing a useful set of tools that support sustained cognitive performance during long study and work sessions. The system leverages a range of environmental sensors, a personal activity tracker and the Pomodoro time management technique to increase awareness of the current state of a study environment and the user's personal activity, and then to provide feedback and timely interventions that support healthy and productive study patterns.

Initial user testing with the Bill-E Focus Bot found that it did achieve awareness raising regarding environmental and personal factors that can impact focus and concentration. User feedback also suggested the value of a modular sensor platform such as the one described here, in terms of a flexible and extensible platform that can add support for more types of sensors or functionality as needed. As with many IoT projects, future work may be required to address sensor calibration, accuracy, or edge case scenarios such as interrupted or intermittent Wi-Fi connectivity.

Keywords: IoT, productivity monitoring, ESP8266, MQTT, environmental sensing, biometric tracking

Introduction

Bill-E Focus Robot's design combines time management (Pomodoro), environmental control (climate and lighting), and personal health tracking (activity logging and movement prompts) to tackle the various aspects that affect concentration and productivity during prolonged tasks. The methodology employed by Bill-E Focus Robot for increasing user productivity is three-pronged, addressing time management, environmental factors, and user health simultaneously, as they are all factors that can have an impact on cognitive performance and the ability to maintain focus over long durations.

Distributed and decentralized to support resilience and extensibility with real time constraints on all subsystems, every part in this system has a unique function but also is a part of a larger orchestrated group of actions to achieve the entirety of what is considered the productivity system.

The development environment used for the project is the Arduino IDE with the C++ language. The program was written in an object-oriented style and broken into modules. Each board is fully self-sufficient, but coordinates with the other parts of the system through a shared language, JSON messages sent over MQTT through WiFi.

Requirements met:

1. Used 3 ESP-based microcontolers
2. Minimum 3 sensors / actuators per board
3. Developed using Arduino IDE
4. Controllers linked using MQTT
5. Home Assistant Dashboard
6. Used Raspberry Pi 5 for hosting Home Assistant (Optional)

Description of Major Components

System Architecture Overview

Bill-E is based on an IoT architecture with three layers:

1. Sensing
2. Communication
3. Presentation (HA dashboard)

The sensing layer is deployed on 3 ESP8266 microcontroller boards, each with a distinct set of responsibilities. These include environmental sensing, user interaction, and wearable activity monitoring. The idea of partitioning functionalities across multiple nodes is to avoid single points of failure, and to enable independent testing and development for each subsystem.

The communication layer is based on the MQTT protocol over local Wi-Fi. MQTT is a publish–subscribe-based communication, where each node pushes sensor readings to MQTT topics, and subscribes to topics they are interested in. With this approach, there is no need to maintain many-to-many connections between devices all the time, as compared to a traditional client-server model. This means lower communication overhead, and better scalability should more devices be added in the future.

The presentation layer is provided by Home Assistant (HA), a dashboard running on Raspberry Pi. HA provides a unified view of sensor readings, control over actuators, and system states. This is done in real time, and with the ability to set simple automation rules like turning on a fan when temperature crosses a threshold, or sending a reminder when a Pomodoro cycle is over.

Hardware Components

Main Brain Unit (ESP8266-1)

The main brain is the central unit of the system. It aggregates the data from other components and is the main communication point with the user.

Core Cycle Structure:

- **Work Session**: 25 minutes of focused work time
- **Short Break**: 5 minutes of rest after each work session
- **Long Break**: 15 minutes of extended rest after every 4th work session

State Transition Logic:

- **Work → Short Break**: After completing 1st, 2nd, or 3rd work sessions

- **Work → Long Break**: After completing 4th work session (every 4 cycles)
- **Any Break → Work**: After break completion, return to focused work
- **Manual Control**: Touch sensor allows forced transitions or confirmations

Bill-E Implementation Features:

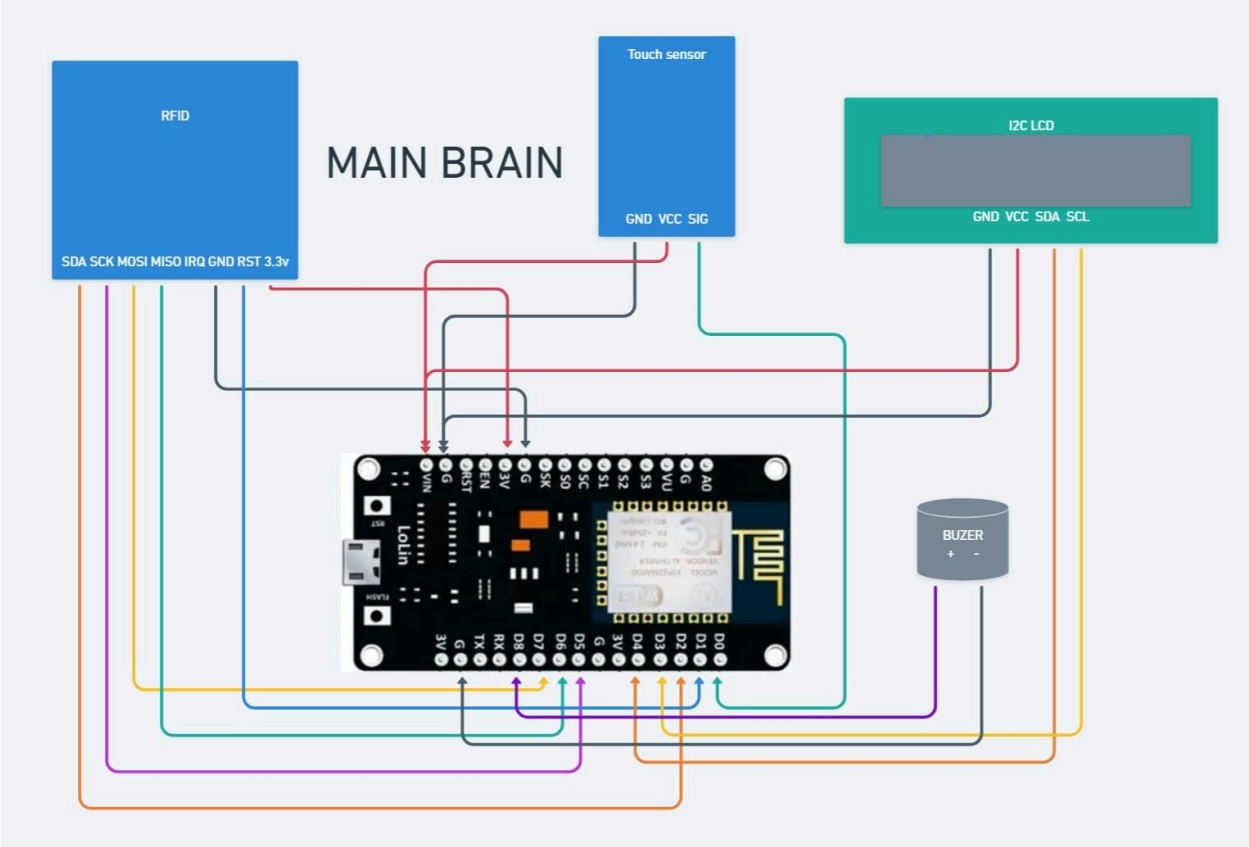
- **RFID Authentication**: Session starts only with valid card scan
- **Confirmation System**: Timer pauses and awaits touch confirmation when periods complete
- **Break Snoozing**: Users can extend breaks by 5 minutes using RFID card double-tap
- **Activity Integration**: Wearable tracker monitors movement compliance during breaks
- **Audio Feedback**: Different sounds for work start, break start, session complete, and alerts
- **Multi-Device Coordination**: Main brain broadcasts timer state via MQTT to all connected devices
- **Health Monitoring**: System tracks prolonged sitting and encourages movement during breaks

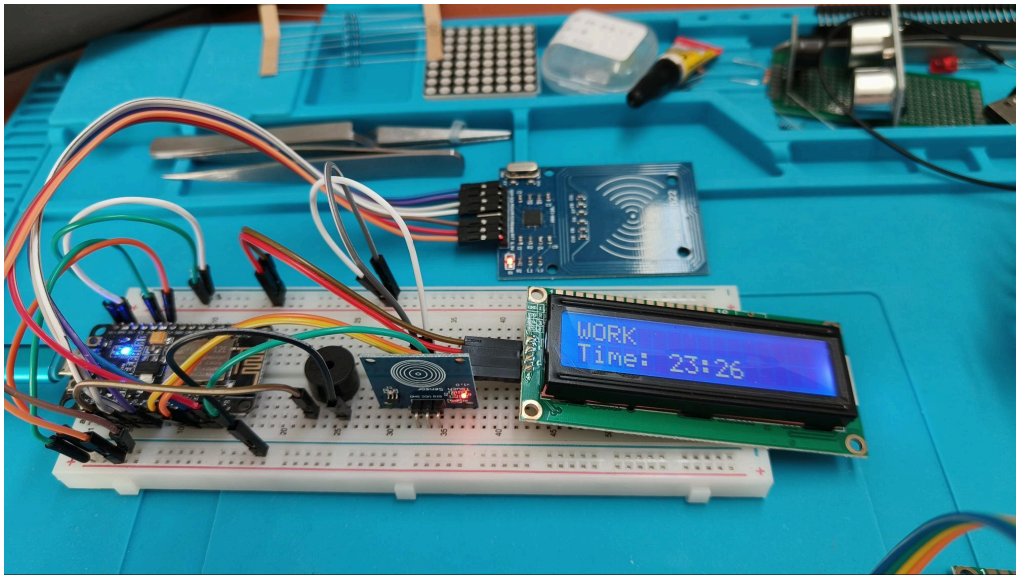
Design Rationale:

A distributed, but centralized control architecture eases state management in the system and also gives users a single point of interaction. RFID authentication combined with touch controls present a compromise between security and usability. This is a critical balance for a personal productivity device. Human computer interaction studies show that multimodal feedback (visual, audio, tactile) leads to higher user engagement and system compliance (Wickens & Hollands, 2000).

Key Components:

- RFID authentication system (RC522)
- Touch sensor for manual state control (TTP223)
- Pomodoro timer implementation logic
- Visual information and state display (I2C 2x16 LCD display)
- Buzzer for audio feedback (passive buzzer)


RFID
SDA SCK MOSI MISO IRQ GND RST 3.3v
MAIN BRAIN
Touch sensor
GND VCC SIG
I2C LCD
GND VCC SDA SCL
BUZER
+ -


WORK
Time: 23:26

Environment Monitor (ESP8266-2)

The Environment Monitor's function is to monitor the local study or work environment for factors that are scientifically proven to have an influence on cognitive effectiveness. Here, outside of the standard sensors and actuators, I also opened up the fan I bought a the local store and soldered wires to it to bypass the original control system.

Core Monitoring Functions:

- **Temperature & Humidity**: DHT11 sensor provides baseline climate data
- **Light Level**: KY-018 photoresistor measures ambient lighting with curve-fitted lux calculation
- **Noise Detection**: KY-038 sensor monitors both analog noise levels and digital sound events
- **Data Publishing**: Environmental readings transmitted via MQTT every 10 seconds

Automatic Fan Control System:

- **Temperature Thresholds**: Fan activates at ≥24°C, deactivates at ≤22°C
- **Hysteresis Control**: Prevents rapid on/off cycling with 2°C temperature buffer
- **Manual Override**: Remote MQTT commands allow manual fan control (on/off/auto)
- **Relay Interface**: 5V relay module controls 4.5V desktop fan safely

Alert Generation Logic:

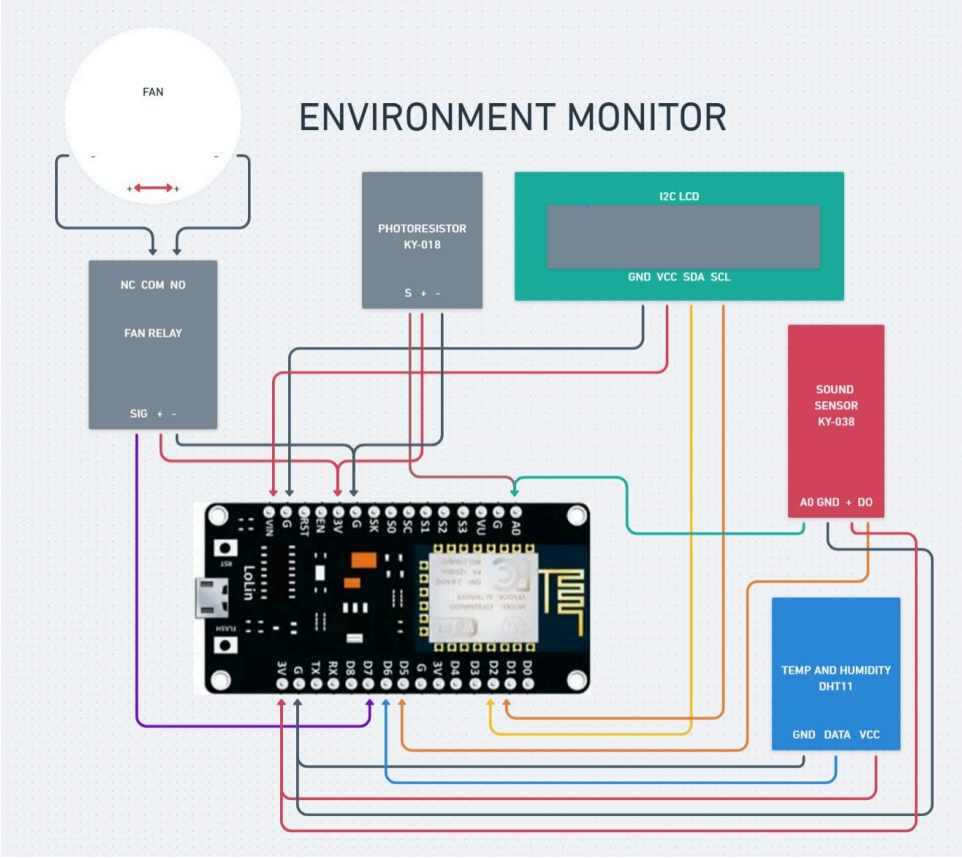
- **Temperature Alerts**: Warns if too cold (<20°C) or too hot (>26°C) for optimal focus
- **Noise Alerts**: Triggers when noise level exceeds threshold (>4 units)
- **Lighting Alerts**: Notifies when illumination is insufficient (<270 lux)
- **MQTT Broadcasting**: Environmental alerts published to main brain for user notification

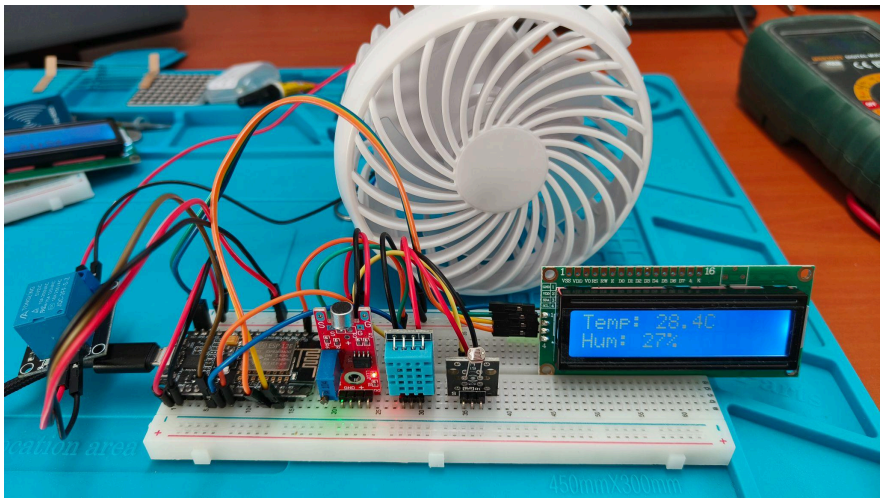
Design Rationale:

Environmental conditions affect concentration and efficiency. Temperature, light and noise levels have been shown to have an effect, with a thermal range of 20-24°C, 300-500 lux (computer work) and less than 50dB (extended concentration) being most beneficial (Wyon, 1974; Boyce et al., 2006). The intelligent fan controller uses proportional control algorithms to maintain thermal comfort with lower cycling and thus, less energy consumption.

Key Components:

- Temperature/humidity monitoring (DHT11)
- Light level sensing (KY-018)
- Noise detection (KY-038)
- Intelligent fan control with manual override (Relay)
- Measurements display (I2C 2x16 LCD display)





Wearable Tracker (ESP8266-3)

The wearable tracker is made to track the user's physical activity. It's meant to be a personal device, just like smart watch. In many studies mentioned in the project proposal, it is stated that physical movement plays a crucial role in mental and cognitive performance

Core Biometric Monitoring:

- **Accelerometer Processing**: MPU6050 provides 3-axis acceleration data converted to g-force magnitude
- **Step counting**: The steps are estimated based on the current activity.
- **Activity Classification**: Real-time categorization into Sitting, Still, Moving, Walking, or Running based on acceleration patterns
- **Movement Tracking**: Timestamps last significant movement for health alert calculations

Health Monitoring System:

- **Work Session Alerts**: Movement reminders after 20+ minutes of inactivity during focus periods
- **Break Compliance**: Encourages physical activity during Pomodoro breaks (movement within 2 minutes)
- **Sedentary Warnings**: Escalating alerts for prolonged sitting periods
- **Context-Aware**: Different thresholds and messages based on the current Pomodoro state

Multi-Screen OLED Display:

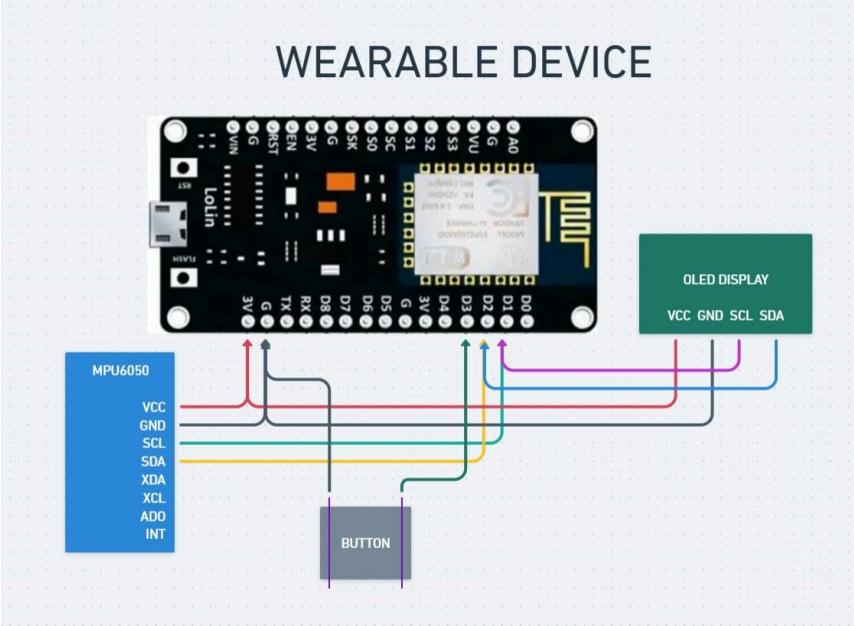
- **Mode 0**: Biometric overview (steps, activity, session context)
- **Mode 1**: Activity details (acceleration, movement timestamps)
- **Mode 2**: Pomodoro progress (timer, cycles, completion percentage)
- **Mode 3**: Break compliance feedback with visual encouragement
- **Button Navigation**: Push button cycles through display modes

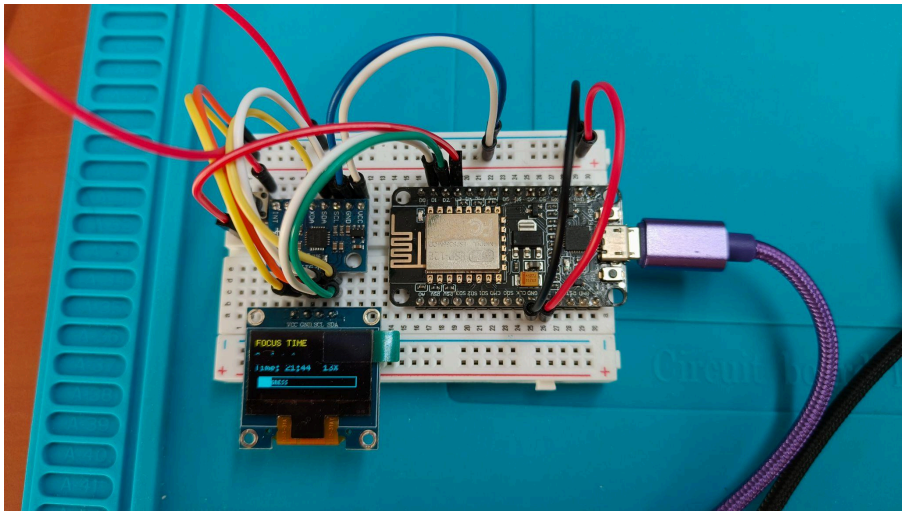
Design Rationale:

The Wearable Tracker keeps track of the user's physical activities and movement patterns. It is used to analyze the correlation between the physical activities and the productivity of the user. This component has an advanced motion detection algorithms for accurate tracking of physical activities.

Key Components:

- Movement detection (MPU6050)
- Visual information (OLED display)
- Display switch (Button)





Development Lifecycle

Development Methodology

An iterative Agile development methodology was used in this project, spanning 5 major phases. This included prototyping, testing, and design, with user feedback and refinement built into each phase. Functionality was designed to be deployed in working components that could be validated independently before being incorporated into the full system.

Cycles followed an Agile "inspect and adapt" methodology, with each review and retrospective session used to identify changes and improvements. This was especially useful for components that had constraints or limitations in the hardware design,

An iterative approach was critical to the success of this project. A number of key decisions, like the move to MQTT and simplifying the displays, were not realised until early prototypes were evaluated. The Agile approach made it possible to pivot without losing momentum.

Development Stages

Requirements Analysis and Design (Phase 1)

The first phase involved the collection of all necessary hardware and an analysis of potential configurations. Circuit diagrams were created to better understand possible sensor and actuator connections. Also, multiple architectural options were sketched out. The intention here was not to determine specific technologies, but rather to map out high-level architecture of the system.

It also became clear that a modular design was a necessity. The project was broken up into three nodes based on ESP8266, which allowed for individual planning, testing and documentation. These early design decisions set the stage for every later phase.

Individual Component Development (Phase 2)

The ESP8266 nodes were assembled and tested one by one, and sensors were added sequentially. In this phase, the objective was to establish that each sensor could provide accurate readings and actuators were working as expected in response to basic commands.

The testing process at this stage used mainly the Arduino IDE serial monitor. For instance, some of the sensors provided raw output early on, which allowed for the correct mapping and transformation to the standardised units. Another example is the KY-038 sound sensor, which also had to be calibrated by trial and error to avoid false readings.

Integration and Communication (Phase 3)

Initially, the boards communicated using a mesh network, where they would exchange information with each other. However, this approach made it difficult to integrate the boards with Home Assistant (HA), aggregate the data and analyse it in a web dashboard.

The switch to MQTT with HA as the central node was a breakthrough. The data flow was much cleaner; each node only had to publish to its own set of topics and let HA visualise and automate. The nodes were also no longer dependent on each other.

This step involved a lot of communication logic refactoring, but the system became much more robust and flexible. The HA dashboard was customised with cards for environmental information, Pomodoro state, and wearable feedback.

Enhancement and Optimisation (Phase 4)

With the three nodes talking to each other reliably, the project's emphasis moved to usability and maintainability. The code bases were well over 700 lines of code per node and were quickly becoming unmanageable. The main re-factor was a code re-organization that split the code into many smaller, well-defined components. Legacy functions were stripped, and redundant logic was replaced by concise routines.

At this stage, many functional improvements were made. The step counting algorithm was re-tuned to minimize false positives, and the Pomodoro touch sensor was implemented to give the user more control.

System Testing and Final Adjustments (Phase 5)

This last stage was the testing of the entire system in a real-world scenario. First was the 24-hour continuous operation test which proved the MQTT communication and ensured the HA log sensor data properly. In testing with users, the biggest complaint was the information overload on the displays in where the wearable OLED and LCD automatically cycled through the sensor data. As result the buttons were added to cycle through the views and states manually.

I also realised through the testing the philosophical lessons in design that “less is more”. It was very tempting to just show as much data as possible, but my personal experience and user feedback would prove otherwise, and a simpler setup worked best. This is the main lesson of this principle that meeting technical success doesn't mean a success of a good user experience.

At the end of this stage, all of the key system requirements were met: environmental monitoring, wearable activity monitoring, pomodoro management, and central dashboard integration.

Git commit history:

1. changed the displayed data in main brain
2. project ready for delivery
3. added button to werable device + fixes in main brain and env monit
4. Touch sensor for switching pomodoro state
5. integrated fan control
6. refactor complete

7. refactored main_brain and wearable_tracker. The latter doens't function properly
8. implemented MQTT integration between the boards
9. integrate pomodoro to wearable tracker
10. added pomodoro buzzer and pomodoro timer
11. added mesh to wearable Bill-e
12. weareable tracker init
13. env monitor with mesh
14. main brain with added mesh
15. Init

Critical Analysis and Reflection

Successful Implementations

Distributed System: The integration of three ESP8266 nodes over MQTT protocol resulted in successful and consistent communication between different devices and the centralisation of information from them. The choice of separating functionality into several nodes helped in avoiding single points of failure and also in being able to work on each module separately without affecting the other two. This architecture held strong over long periods of time and allowed for modular upgrades to the system as a whole.

Dashboard: The unified dashboard shows a live view of the environmental conditions, the current states of the system, and usage statistics. It allowed for remote monitoring and control of the system. The switch for controlling the fan was considered a success, as users could manually turn it on or off whenever they felt necessary.

User Interface: The multi-modal feedback system of visual display, audio alerts, and tactile controls was found to be effective. Test users commented that it was "easy to use and intuitive" suggesting a successful human-computer interface design. The RFID-authentication in conjunction with touch sensors provides an acceptable level of security, as well as usability for personal productivity applications.

Challenges and Solutions

Problem with pulse sensor: The KY-036 heart rate sensor gave erratic readings. The variance in raw data output was too high. The sensor had a low signal-to-noise ratio and was too sensitive to environmental factors. It was impossible to consistently detect heart rate readings even after calibrating the sensor several times.

Solution: This sensor was completely removed from the project.

RFID Module Initialisation Issues: The RC522 RFID reader needed certain initialization steps, including a "cleaning" process, before each session. Failure to initialise correctly resulted in the module not responding to card detection, interrupting the user authentication process.

Solution: Implementation of reliable initialisation routines with memory flushing and automatic retries.

Step Counting Algorithm Accuracy: I couldn't get the algorithm to detect the steps accurately. Even though I tried to implement the solutions that I've found online, I still couldnt get them to give me reliable measurements

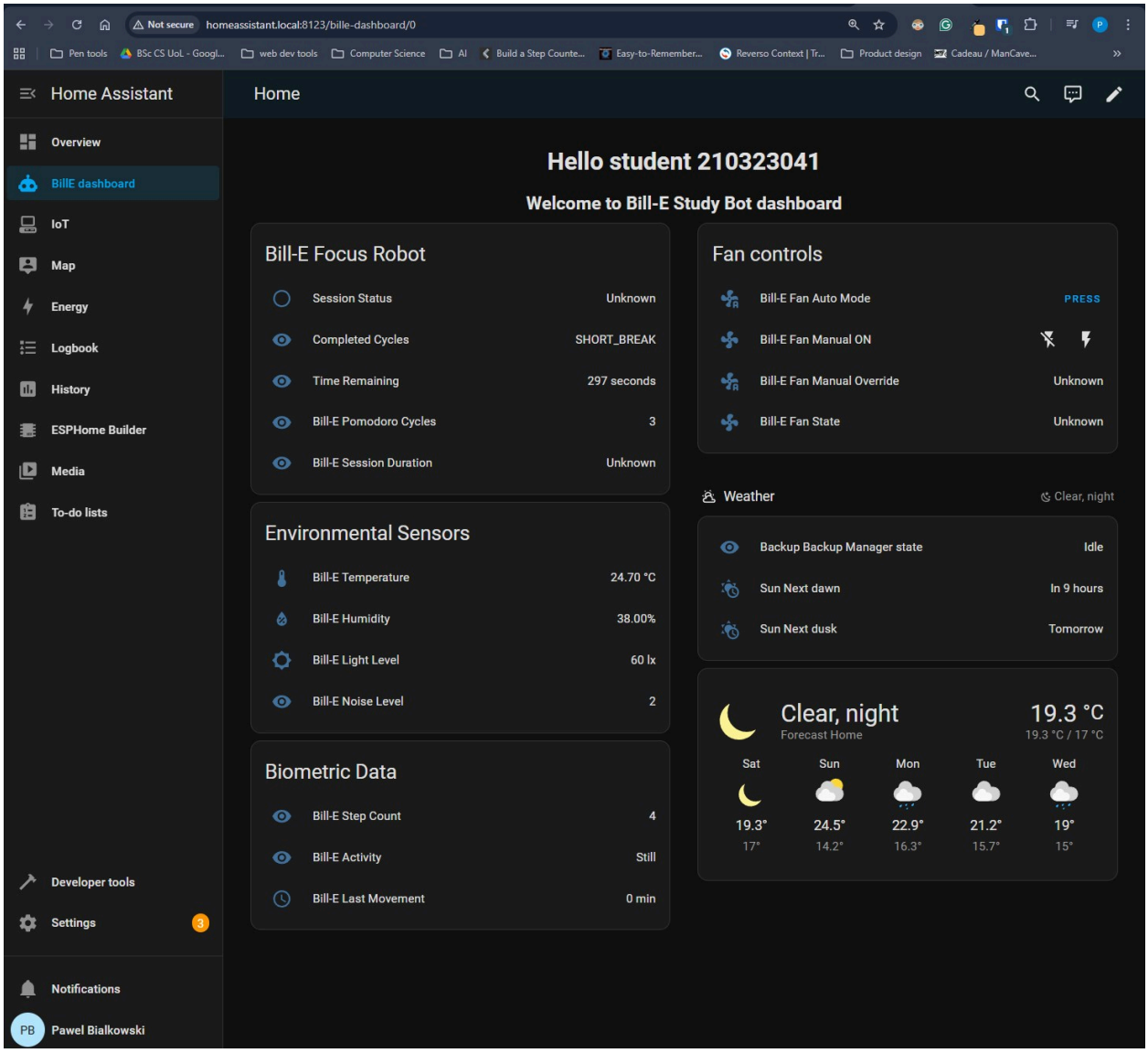
Solution: I checked the estimated step count per activity (walking and running), and since I already tweaked activityDetection algorithm to be quite accurate, I count the average steps per activity (120/min for walking and 180/min for running)

Mesh network problems: The first implementation with mesh network would not work well with HA dashboard. The issue was that ESP8266 cannot have two WiFi connections simultaneously. I tried to have the hybrid, where the system would switch between networks, but that didn't work well either.

Solution: I shifted to a fully centralised MQTT broker architecture with HA as the data aggregator and dashboard.

Appendices

Home Assistant Dashboard



Wiring

Main brain

ESP8266-1 → MFRC522:
3.3V → VCC
GND → GND
D1→ RST
D2 → SDA/SS
D5 → SCK
D7 → MOSI
D6 → MISO
IRQ → Not connected

ESP8266-1 → LCD
(I2C): 5V → VCC
GND → GND
D3 → SDA
D4 → SCL

ESP8266-1 → Buzzer
GND → -
D8 → +

ESP8266-1 → TTP223
5V → VCC
GND → GND
D1 → SIG

Environment monitor:

ESP8266-2 → DHT11:
3.3V → VCC
GND → GND
D6 → DATA

ESP8266-2 → KY-018 (Photoresistor):
3.3V → VCC

GND → GND
A0 → AO

ESP8266-2 → KY-038 (Microphone):
3.3V → VCC
GND → GND
A0 → AO (shared with KY-018)
D5 → DO (digital sound detection)

ESP8266-2 → LCD:
5V → VCC
GND → GND
D1 → SDA
D2 → SCL

ESP8266-2 → Relay:
3.3v → VCC
GND → GND
D7 → S

Relay → Fan
$-_1$ → COM
$-_2$ → NO

Wearable device:

ESP8266-3 → OLED display:
3.3v → VCC
GND → GND
D1 → SCL
D2 → SDA

ESP8266-3 → MPU6050
3.3v → VCC
GND → GND
D1 → SCL
D2 → SDA

ESP8266-3 → Button
D3 → pin1
GND → pin2

Libraries used

Liquid crystal by Arduino 1.0.7
WiFi by Arduino 1.2.7
ArduinoJason by Benoit Blanchon 6.21.5
DHT sensor library by Adafruit 1.4.6
MFRC522 by Githubcommunity 1.4.12
MPU6050 by Electronic Cats 1.4.4
PubSubClient by Nick O'Leary 2.8.0
U8g2 by oliver 2.35.30

MQTT topics

Main Brain (ESP8266-1) - Central Coordinator

Published Topics:

- `bille/session/state` - Session status with user ID and duration
- `bille/session/active` - Boolean session status (true/false)
- `bille/pomodoro/state` - Complete Pomodoro timer state (JSON)
- `bille/pomodoro/cycles` - Number of completed Pomodoro cycles
- `bille/pomodoro/time_remaining` - Seconds remaining in current state
- `bille/pomodoro/current_state` - Current timer state (WORK/SHORT_BREAK/LONG_BREAK/IDLE)
- `bille/status/system` - System health and connectivity status
- `bille/status/mainbrain` - Main brain online presence (retained)
- `bille/alerts/movement` - Movement reminder messages

Subscribed Topics:

- `bille/data/environment` - Environmental sensor data from monitor
- `bille/data/biometric` - Biometric data from wearable tracker
- `bille/commands/session` - Remote session control (start/stop)
- `bille/commands/pomodoro` - Remote Pomodoro commands (snooze/skip)

Environment Monitor (ESP8266-2) - Environmental Sensing

Published Topics:

- `bille/data/environment` - Combined environmental data package (JSON)
- `bille/sensors/temperature` - Temperature reading in Celsius
- `bille/sensors/humidity` - Humidity percentage

- `bille/sensors/light` - Light level in lux
- `bille/sensors/noise` - Noise level measurement
- `bille/sensors/fan_state` - Fan status (ON/OFF)
- `bille/sensors/fan_manual_override` - Manual override status (true/false)
- `bille/status/fan` - Detailed fan control information (JSON)
- `bille/status/environment` - Environment monitor online presence (retained)
- `bille/alerts/environment` - Environmental quality alerts

Subscribed Topics:

- `bille/environment/request` - Data requests from main brain
- `bille/session/state` - Session status updates
- `bille/commands/fan` - Fan control commands (manual_on/manual_off/auto/status)

Wearable Tracker (ESP8266-3) - Biometric Monitoring

Published Topics:

- `bille/data/biometric` - Complete biometric data package (JSON)
- `bille/sensors/steps` - Current step count
- `bille/sensors/activity` - Activity classification (Sitting/Walking/Running/etc.)
- `bille/sensors/last_movement_minutes` - Minutes since last movement detected
- `bille/status/wearable` - Wearable tracker online presence (retained)
- `bille/alerts/health` - Health monitoring alerts (movement reminders, break compliance)

Subscribed Topics:

- `bille/session/state` - Session start/stop notifications
- `bille/pomodoro/state` - Pomodoro timer updates for display context
- `bille/wearable/request` - Data requests from main brain
- `bille/alerts/movement` - Movement reminder notifications

Topic Categories by Function

System Status & Presence

- `bille/status/mainbrain` (retained)
- `bille/status/environment` (retained)
- `bille/status/wearable` (retained)
- `bille/status/system`

- `bille/status/fan`

Session Management

- `bille/session/state`
- `bille/session/active`
- `bille/commands/session`

Pomodoro Timer System

- `bille/pomodoro/state`
- `bille/pomodoro/cycles`
- `bille/pomodoro/time_remaining`
- `bille/pomodoro/current_state`
- `bille/commands/pomodoro`

Environmental Monitoring

- `bille/data/environment`
- `bille/sensors/temperature`
- `bille/sensors/humidity`
- `bille/sensors/light`
- `bille/sensors/noise`
- `bille/alerts/environment`

Fan Control System

- `bille/sensors/fan_state`
- `bille/sensors/fan_manual_override`
- `bille/status/fan`
- `bille/commands/fan`

Biometric & Activity Tracking

- `bille/data/biometric`
- `bille/sensors/steps`
- `bille/sensors/activity`
- `bille/sensors/last_movement_minutes`

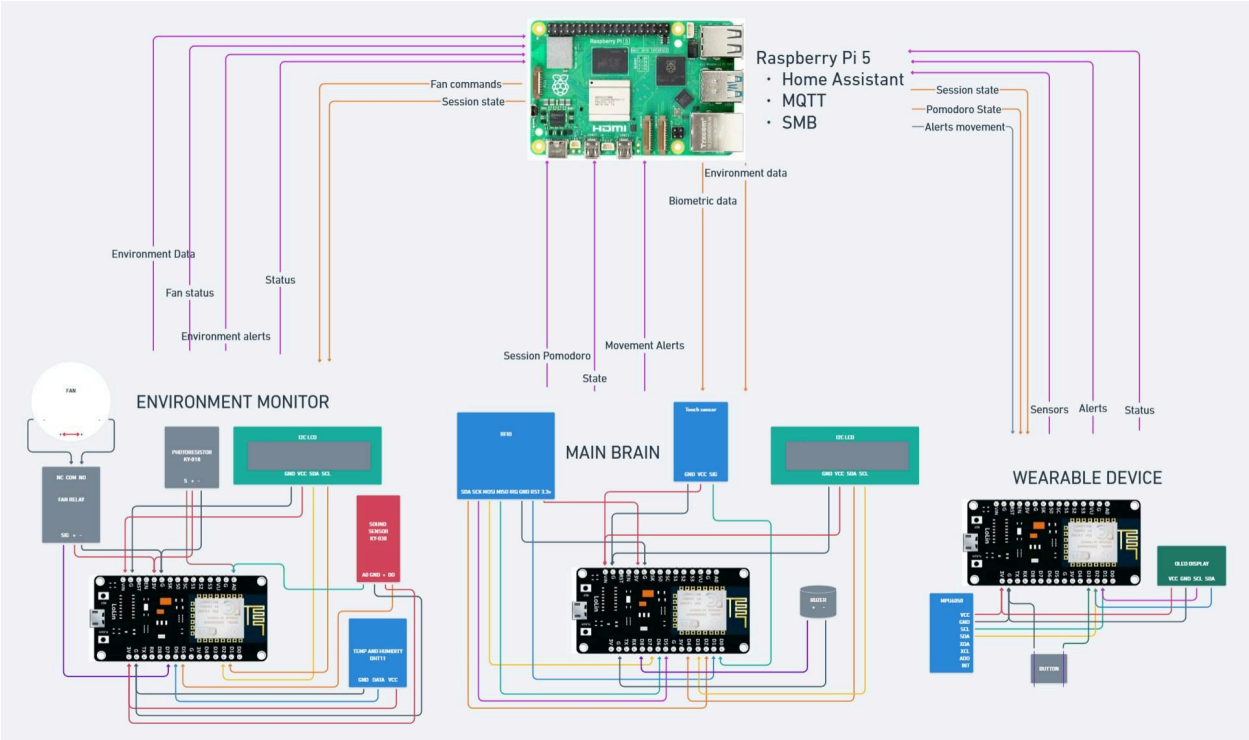
Health & Alert System

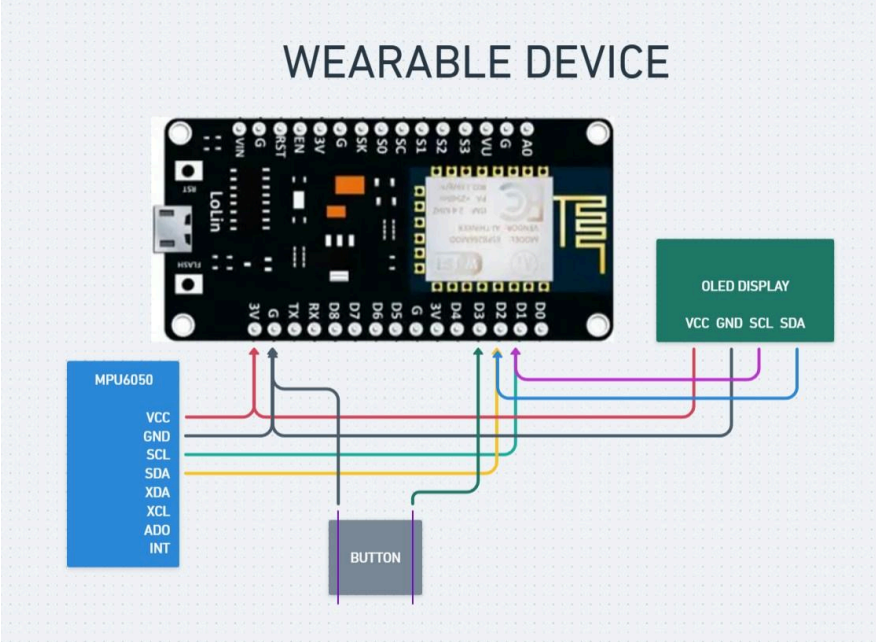
- `bille/alerts/health`
- `bille/alerts/movement`
- `bille/alerts/environment`

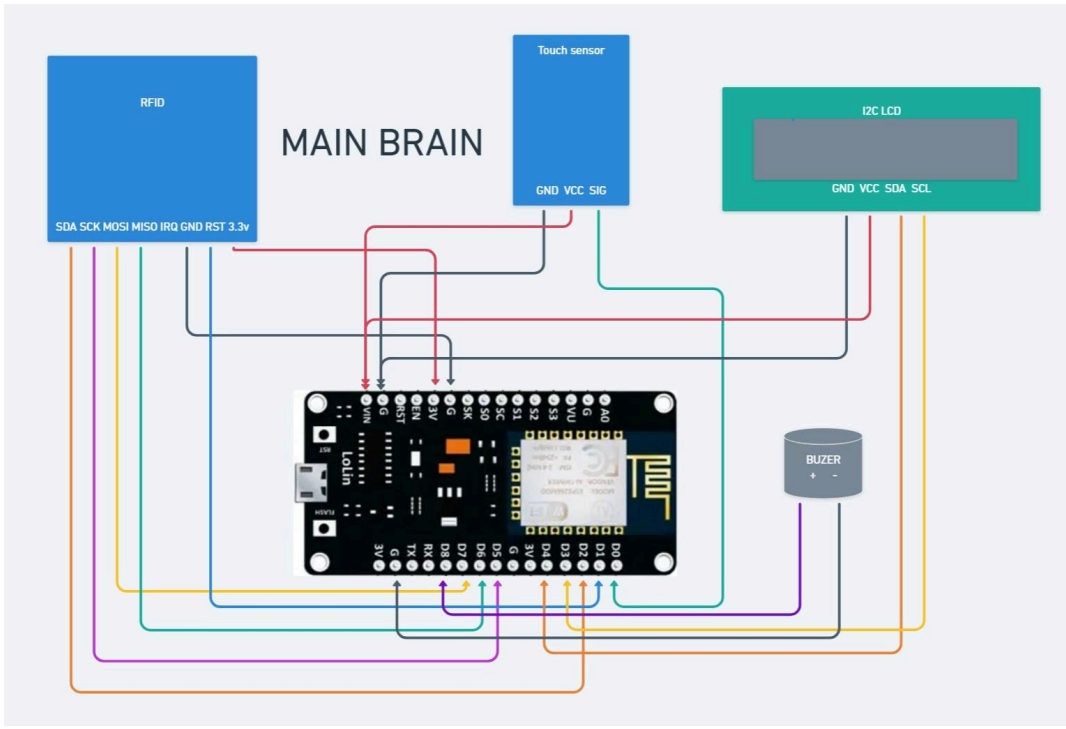
Data Request System

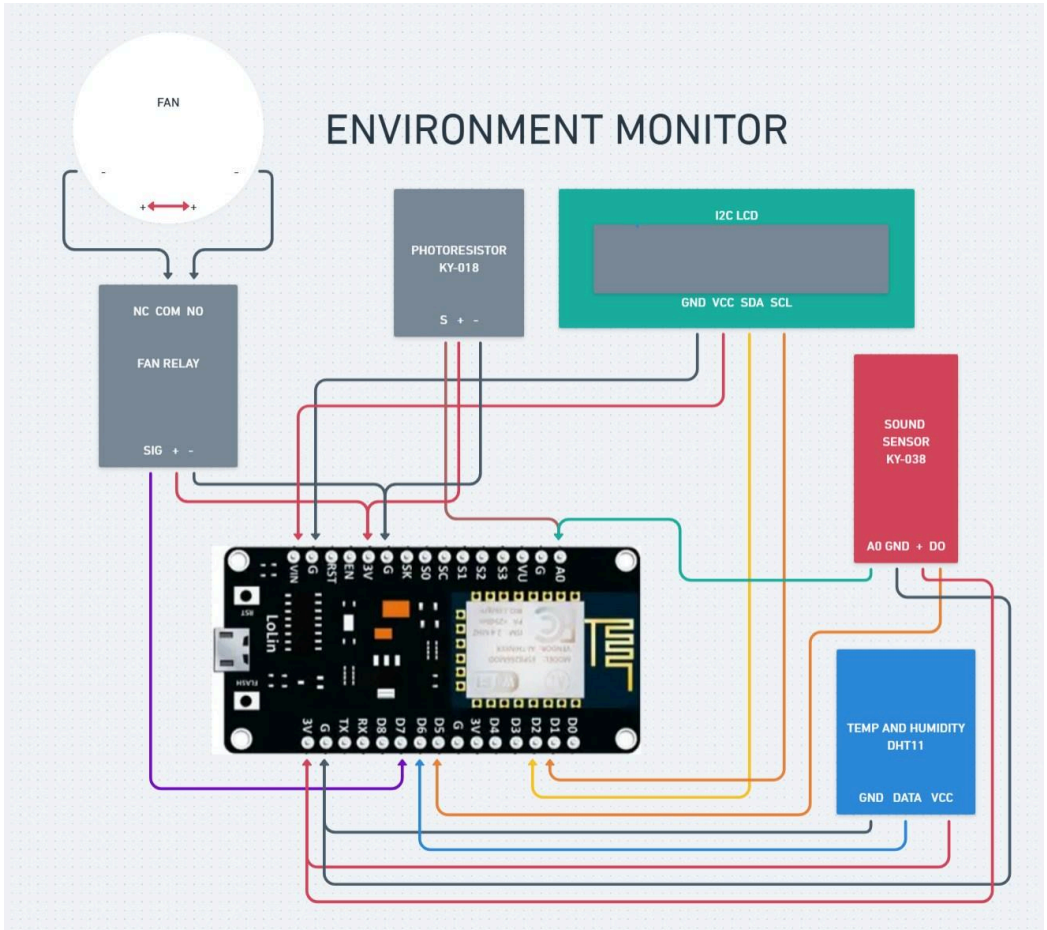
- `bille/environment/request`
- `bille/wearable/request`

Diagrams


Raspberry Pi 5
· Home Assistant
· MQTT
· SMB
Fan commands
Session state
Session state
Pomodoro State
Alerts movement
Environment data
Biometric data
Environment Data
Fan status
Status
Environment alerts
Session Pomodoro
State
Movement Alerts
Sensors
Alerts
Status
ENVIRONMENT MONITOR
MAIN BRAIN
WEARABLE DEVICE


WEARABLE DEVICE
OLED DISPLAY
VCC GND SCL SDA
MPU6050
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
BUTTON


MAIN BRAIN
RFID
SDA SCK MOSI MISO IRQ GND RST 3.3v
Touch sensor
GND VCC SIG
I2C LCD
GND VCC SDA SCL
BUZER
+ -


ENVIRONMENT MONITOR
FAN
NC COM NO
FAN RELAY
SIG + -
PHOTORESISTOR
KY-018
S + -
I2C LCD
GND VCC SDA SCL
SOUND
SENSOR
KY-038
A0 GND + DO
TEMP AND HUMIDITY
DHT11
GND DATA VCC

RFID management

```
13  void handleRFID() {
14    if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) {
15      return;
16    }
17
18    String cardId = "";
19    for (byte i = 0; i < rfid.uid.size; i++) {
20      cardId += String(rfid.uid.uidByte[i], HEX);
21    }
22
23    Serial.println("Card detected: " + cardId);
24
25     // Simple authentication - add your known card IDs here
26    if (cardId == "9c13c3") { // Temporary: accept any card->  || cardId.length() > 0
27      if (!sessionActive) {
28        startSession(cardId);
29      } else {
30        // Double-tap RFID during break = snooze
31        if (pomodoro.currentState == SHORT_BREAK || pomodoro.currentState == LONG_BREAK) {
32          snoozeBreak();
33        } else {
34          endSession();
35        }
36      }
37    } else {
38      // Authentication failed
39      playAuthFailSound();
40      Serial.println("Authentication failed for card: " + cardId);
41    }
42
43    rfid.PICC_HaltA();
44    rfid.PCD_StopCrypto1();
45  }
```

Detect activity

```
String detectActivity() {
  unsigned long timeSinceMovement = millis() - currentBio.lastMovement;

  if (timeSinceMovement < 1000) {
    if (currentBio.acceleration > 1.5) {
      return "Running";
    } else if (currentBio.acceleration > 1.2) {
      return "Walking";
    } else {
      return "Moving";
    }
  } else if (timeSinceMovement < 30000) { // 30 seconds
    return "Still";
  } else {
    return "Sitting";
  }
}
```

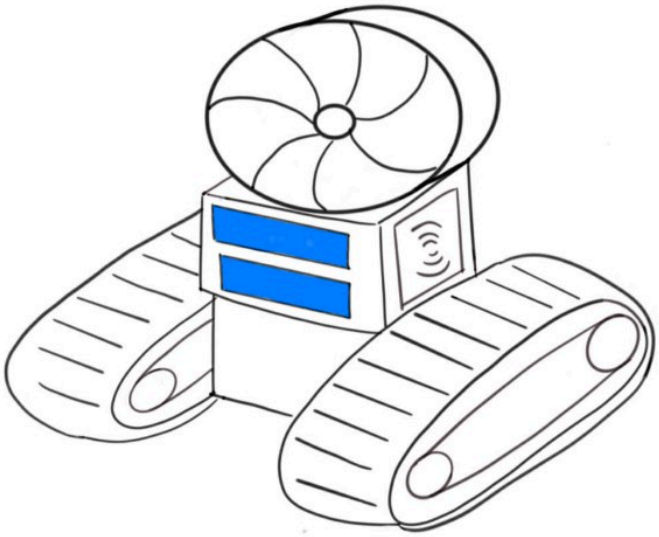
Pomodoro cycles

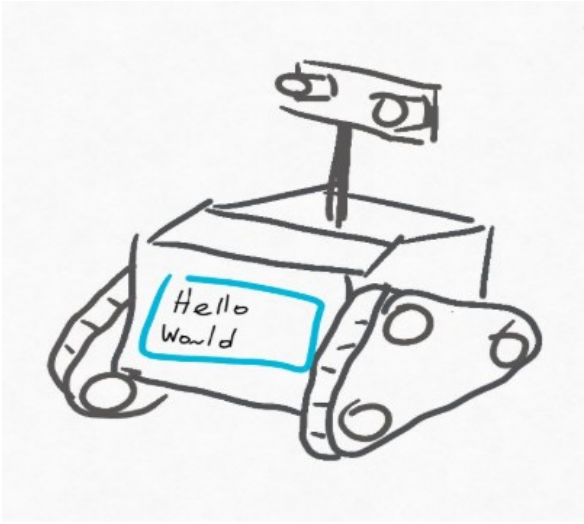
```
void transitionToNextState() {
  switch (pomodoro.currentState) {
    case WORK_SESSION:
      pomodoro.completedCycles++;

      // Long break every 4 cycles, otherwise short break
      if (pomodoro.completedCycles % 4 == 0) {
        pomodoro.currentState = LONG_BREAK;
        pomodoro.stateDuration = pomodoro.longBreakDuration * 60 * 1000UL;
        playLongBreakStartSound();
        Serial.printf("Long break started! Cycle %d completed.\n", pomodoro.completedCycles);
      } else {
        pomodoro.currentState = SHORT_BREAK;
        pomodoro.stateDuration = pomodoro.shortBreakDuration * 60 * 1000UL;
        playBreakStartSound();
        Serial.printf("Short break started! Cycle %d completed.\n", pomodoro.completedCycles);
      }
      break;

    case SHORT_BREAK:
    case LONG_BREAK:
      pomodoro.currentState = WORK_SESSION;
      pomodoro.stateDuration = pomodoro.workDuration * 60 * 1000UL;
      playWorkSessionStartSound();
      Serial.println("Work session started!");
      break;

    default:
      break;
  }

  pomodoro.stateStartTime = millis();
  pomodoro.breakSnoozed = false;
  pomodoro.snoozeCount = 0;
  pomodoro.breakComplianceChecked = false;

  playTouchAcknoledgmentSound();
  publishPomodoroState();
}
```

Fan control

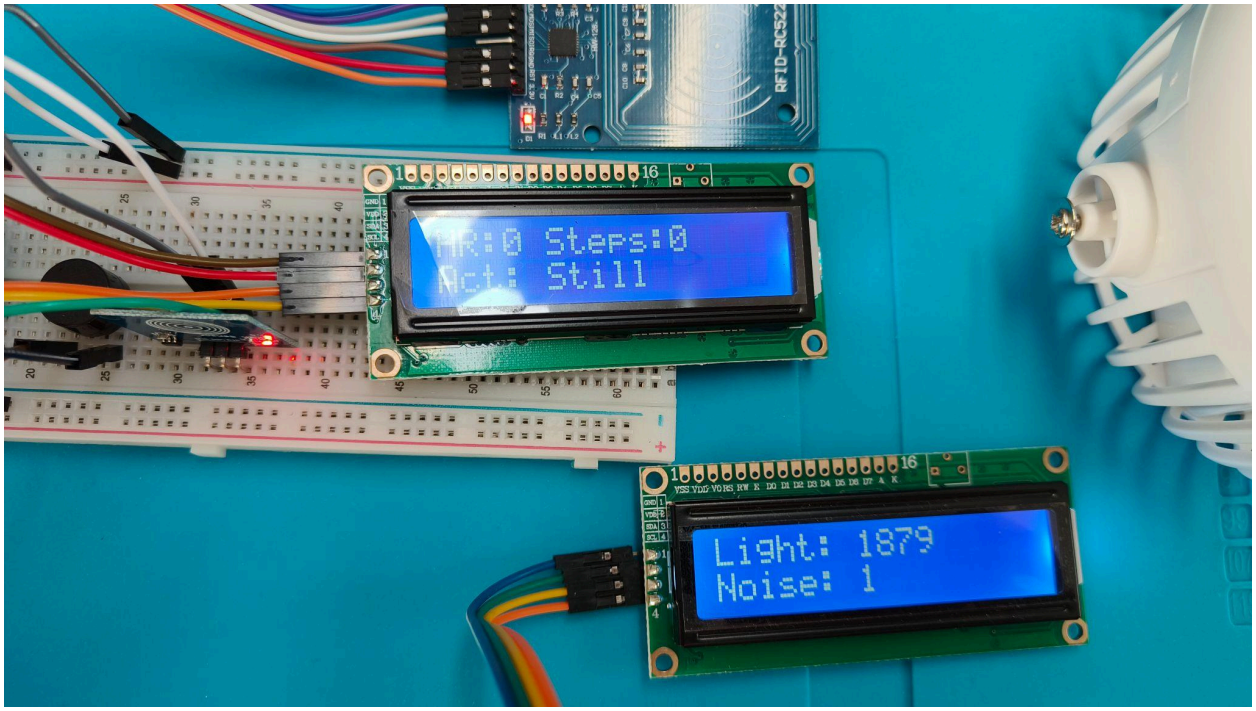
```
void controlFan() {
  bool newFanState = fanState;

  // Check if manual override is active
  if (manualOverride) {
    newFanState = manualFanState;
    Serial.println("Fan manual override active: " + String(newFanState ? "ON" : "OFF"));
  } else {
    // Automatic temperature-based control with hysteresis
    if (currentEnv.temperature >= FAN_ON_TEMP && !fanState) {
      newFanState = true;
      Serial.printf("Auto fan ON: Temperature %.1f°C >= %.1f°C\n", currentEnv.temperature, FAN_ON_TEMP);
    } else if (currentEnv.temperature <= FAN_OFF_TEMP && fanState) {
      newFanState = false;
      Serial.printf("Auto fan OFF: Temperature %.1f°C <= %.1f°C\n", currentEnv.temperature, FAN_OFF_TEMP);
    }
  }

  // Update fan state if changed
  if (newFanState != fanState) {
    fanState = newFanState;
    digitalWrite(FAN_RELAY_PIN, fanState ? HIGH : LOW);

    Serial.println("Fan state changed to: " + String(fanState ? "ON" : "OFF"));
    publishFanStatus();
  }
}
```

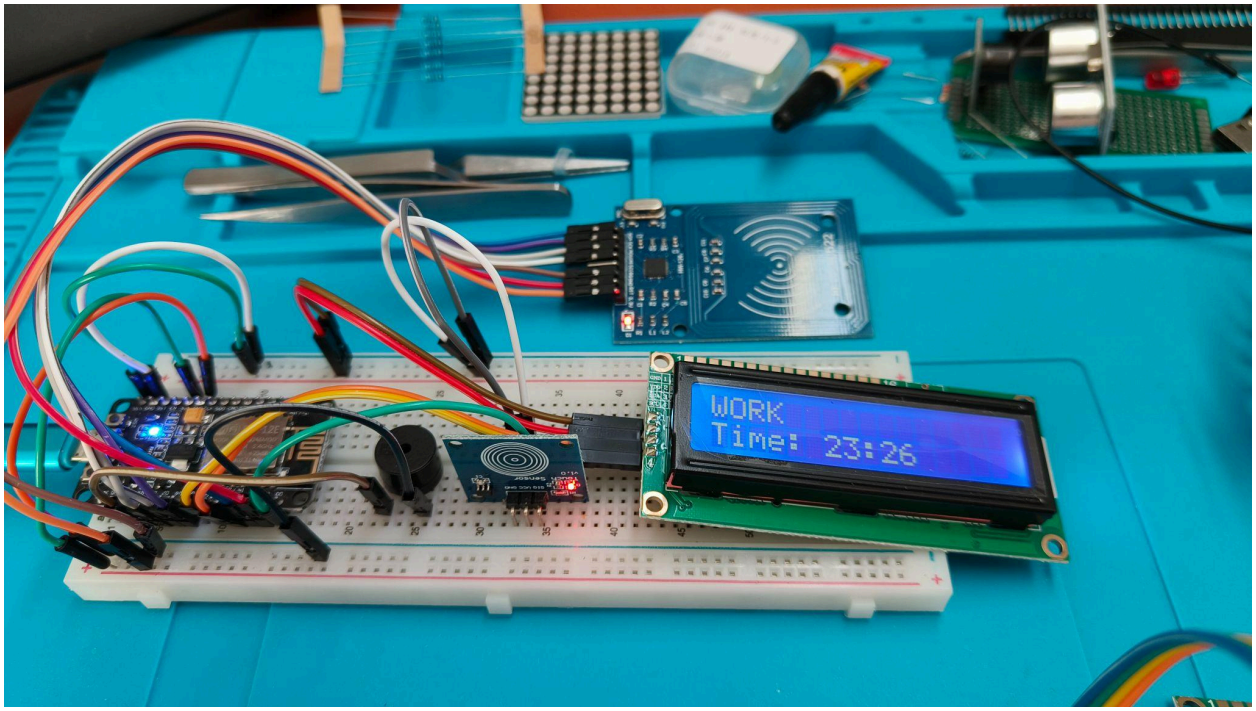
Designs

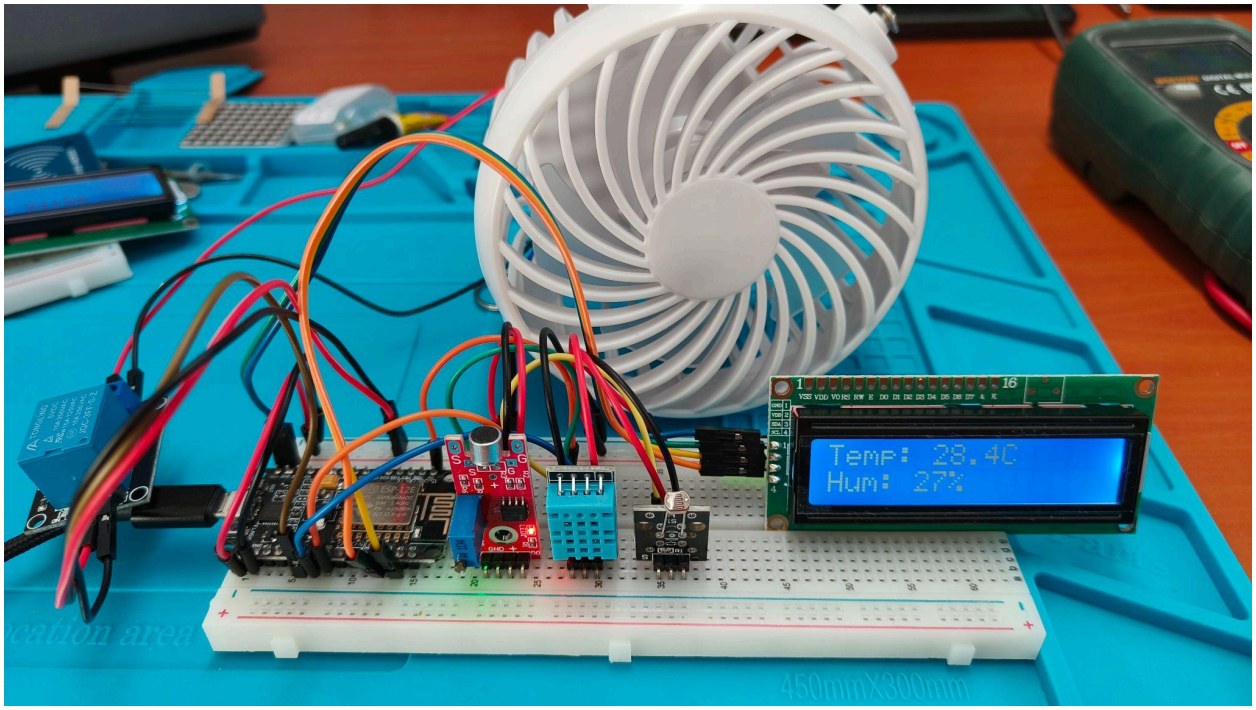


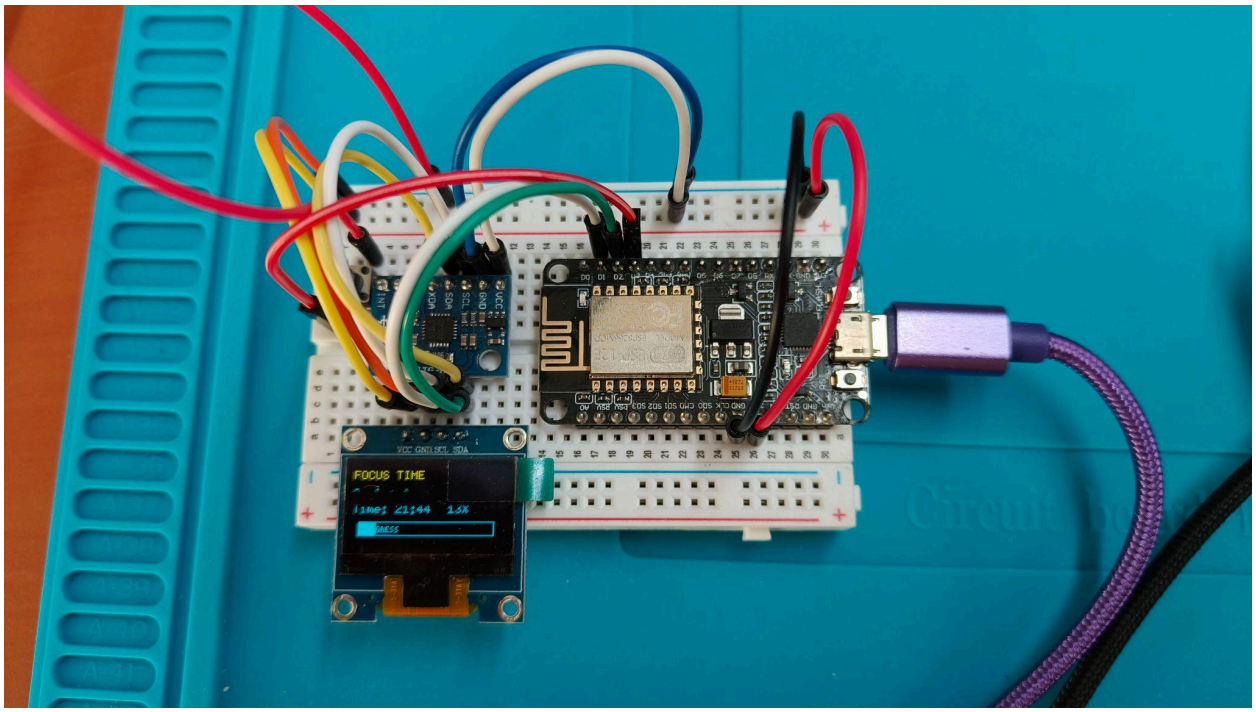


Development

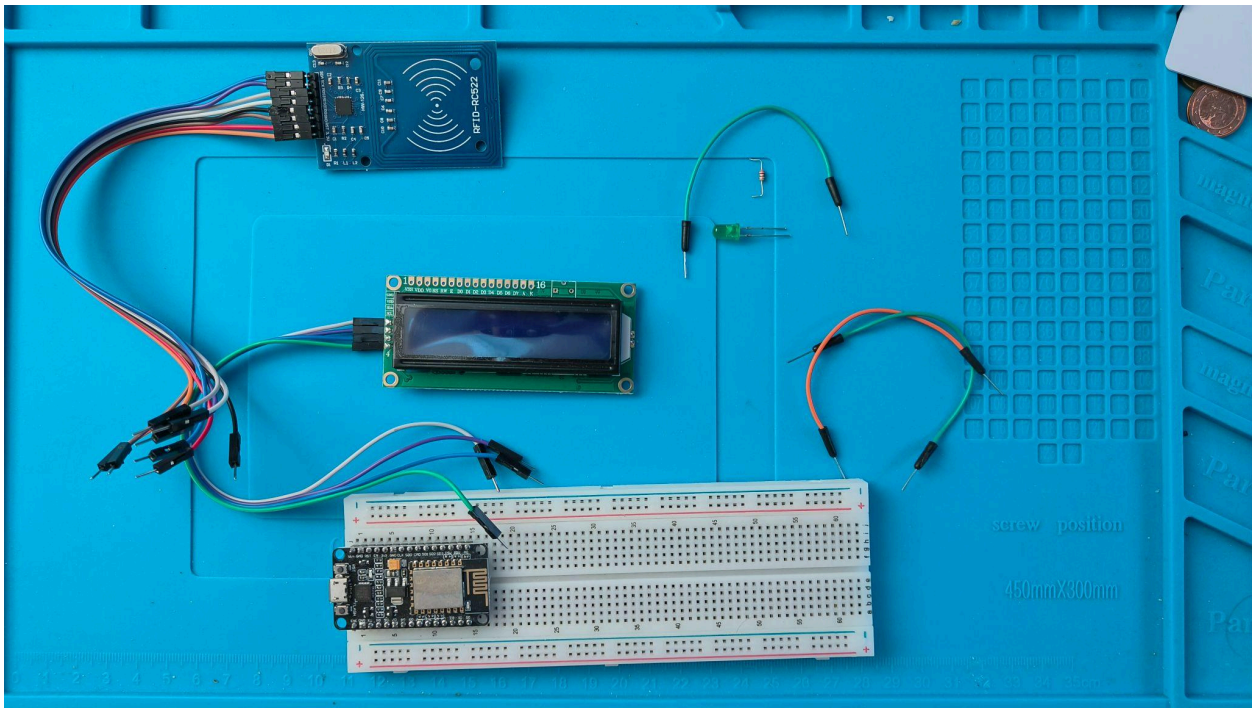

HR:0 Steps:0
Act: Still
Light: 1879
Noise: 1

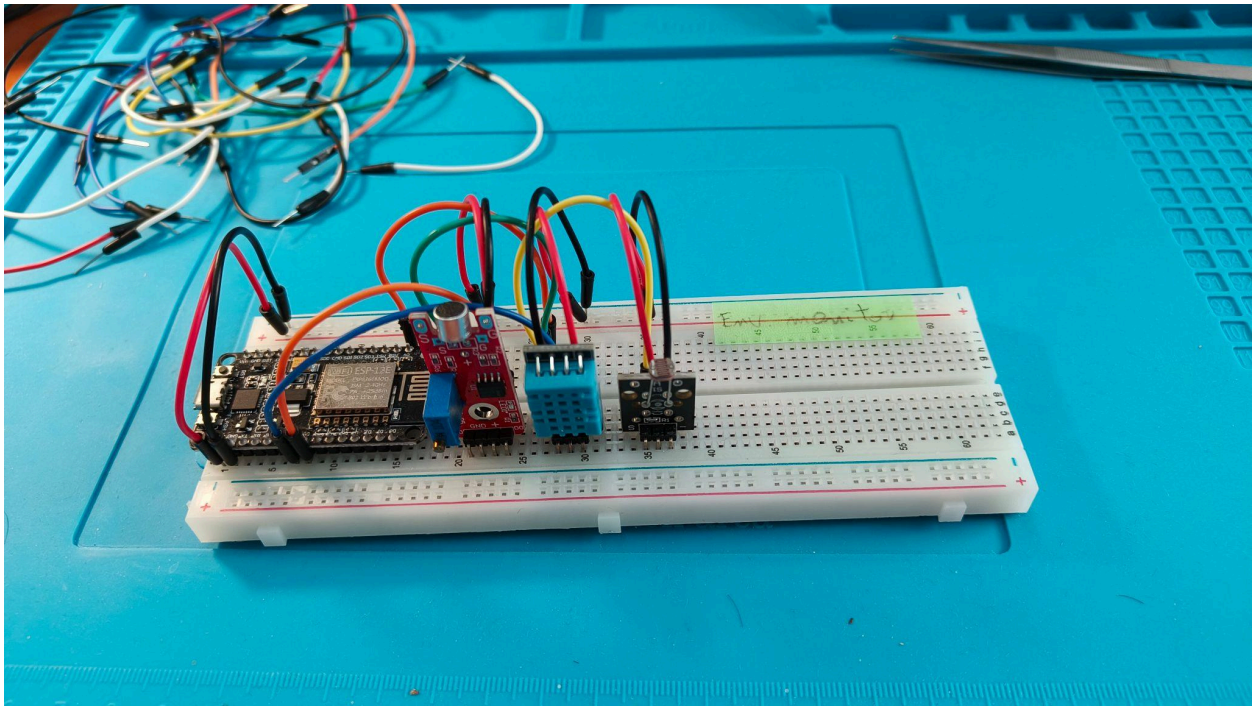

WORK
Time: 23:26

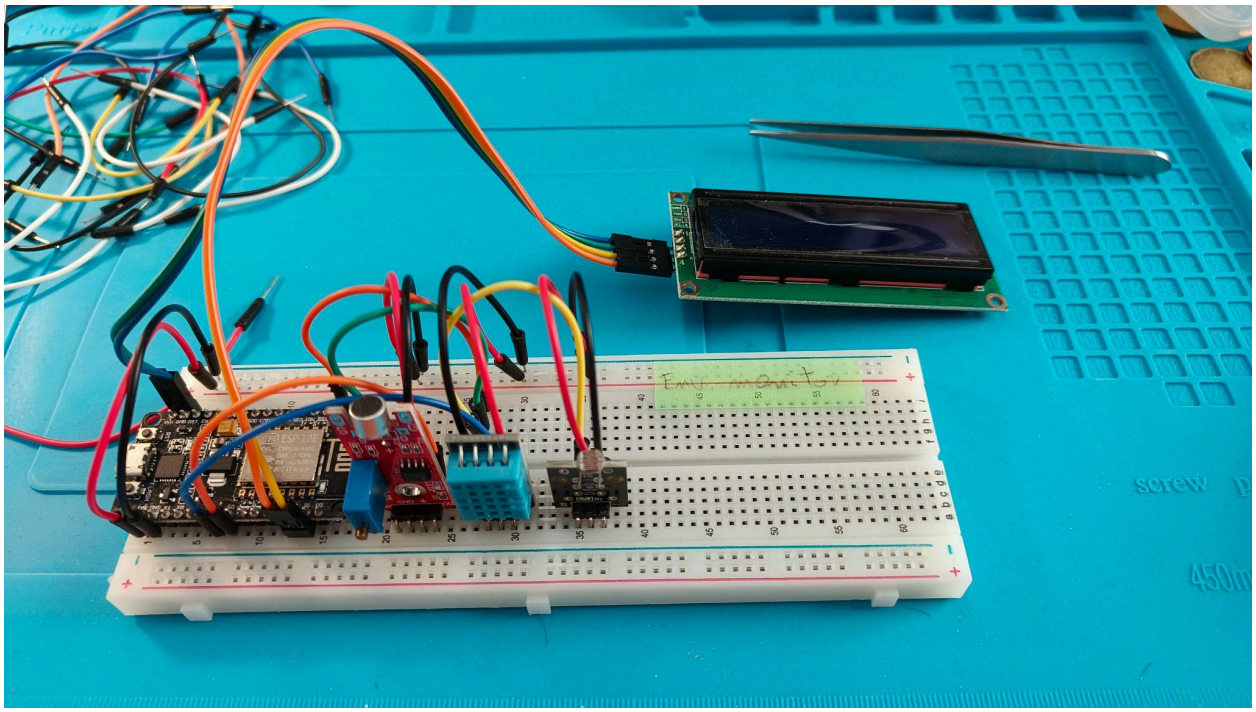

Temp: 28.4C
Hum: 27%
450mmX300mm

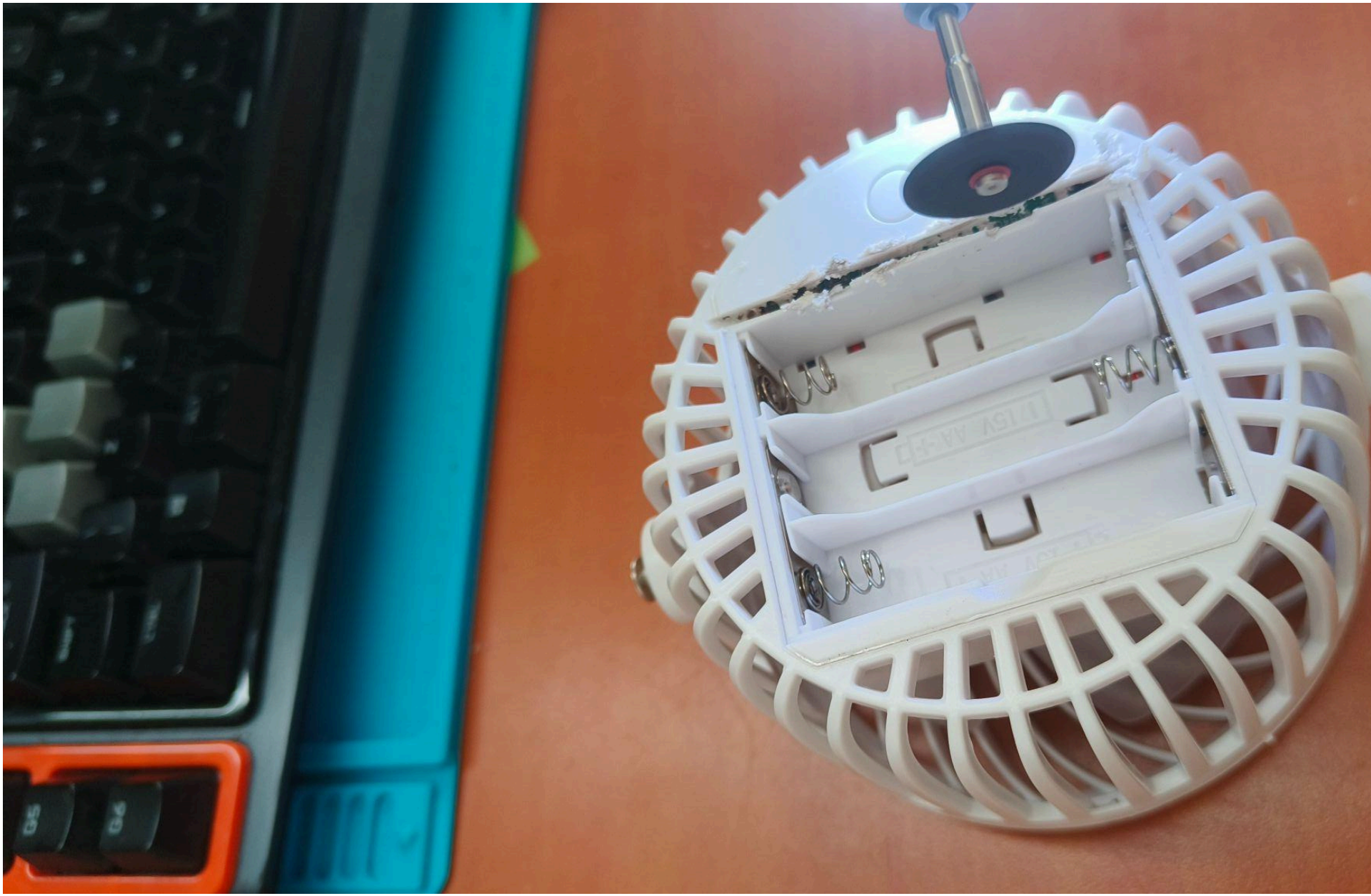

FOCUS TIME


msi
0.2–6.0
MM²

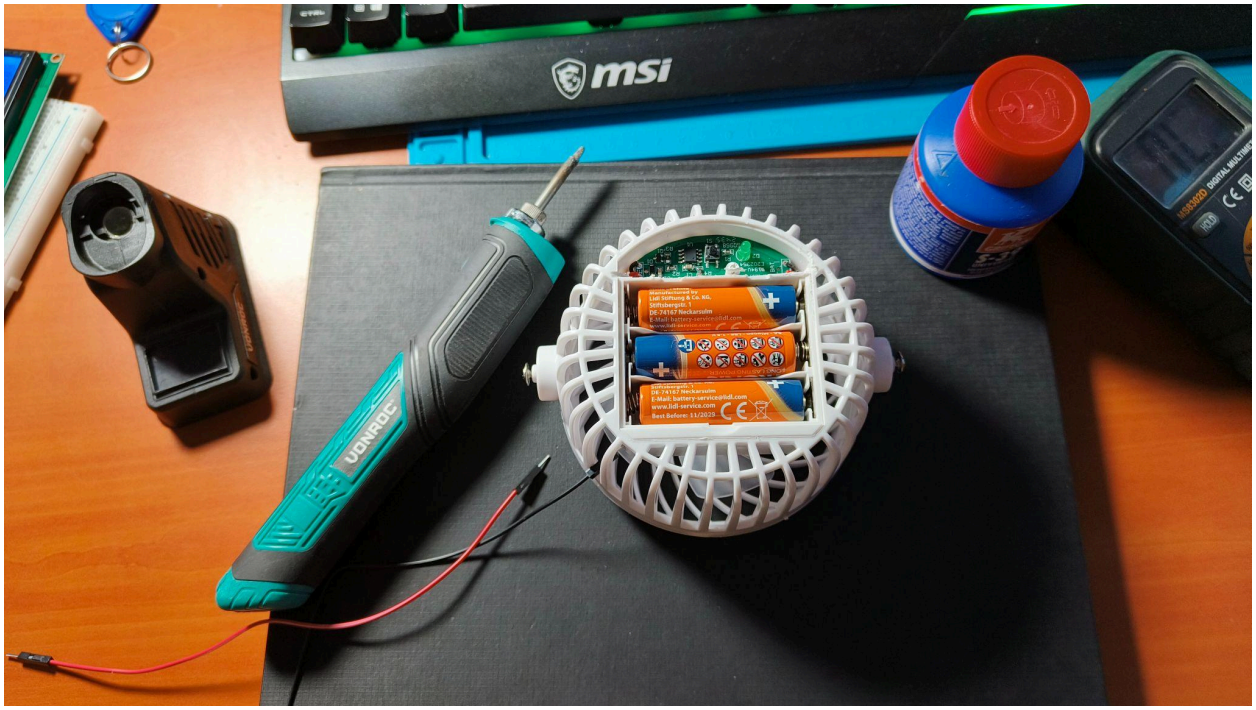

RFID-RC522
screw position
450mmX300mm



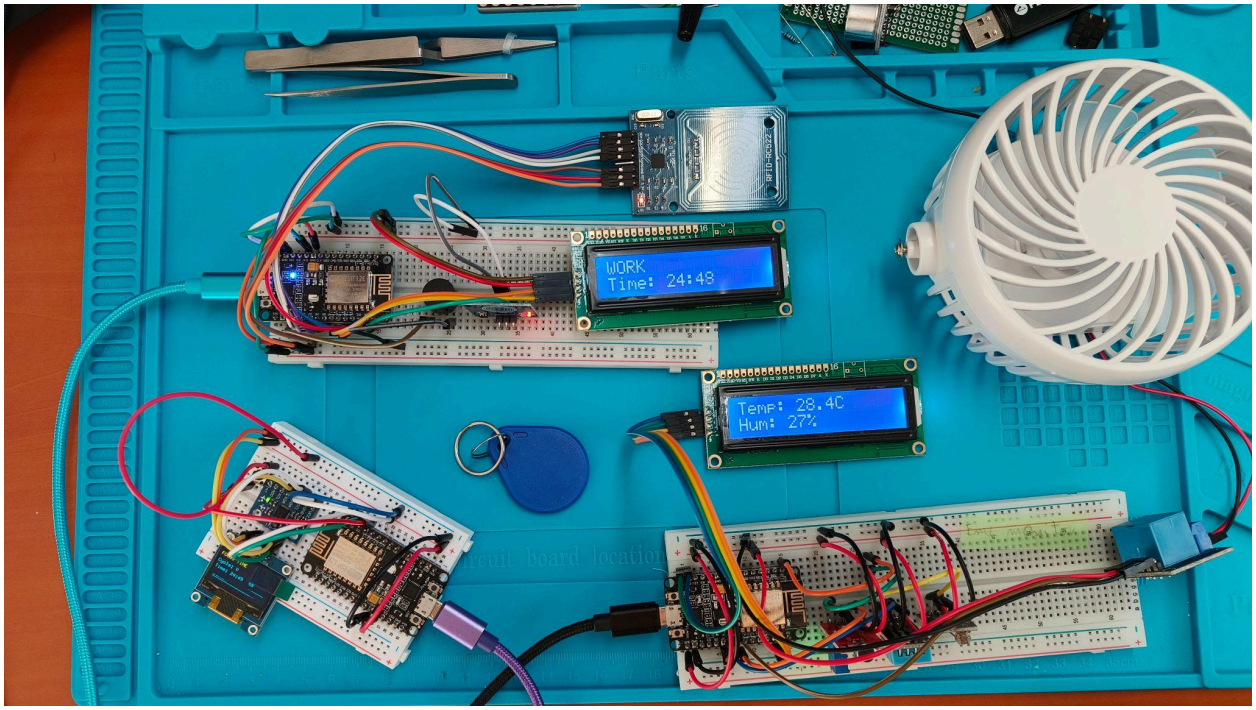


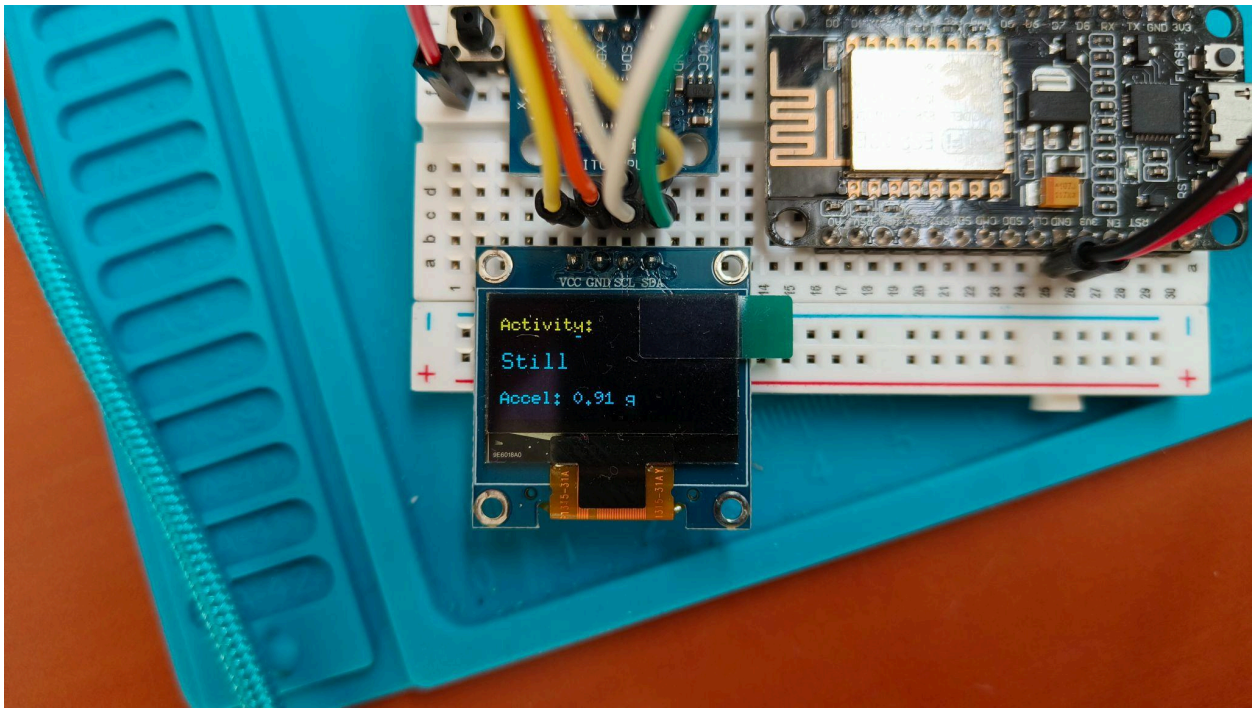



LONG LASTING POWER
AA • Mignon • LR6 • 1,5 V
Made in China
www.lidl-service.com
Made in China
Manufactured by
Lidl Stiftung & Co. KG,
Stiftsbergstr. 1
DE-74167 Neckarsulm
E-Mail: battery-service@lidl.com
www.lidl-service.com
Best Before: 11/2029
DCF-02/03
94V-0
E202354
R1
R2
R3
R4
C1
C2
D1
D2
Q1
U1
S1
GQ968


msi
VONROC

Manufactured by
Lidl Stiftung & Co. KG,
Stiftsbergstr. 1
DE-74167 Neckarsulm
E-Mail: battery-service@lidl.com
www.lidl-service.com
Best Before: 11/2029
BAT
D2


WORK
Time: 24:48
Temp: 28.4C
Hum: 27%


VCC GND SCL SDA
Activity:
Still
Accel: 0.91 g