A Comparative Study of an STM32F103C8T6-Based FreeRTOS Smart Home System for Environmental Monitoring and Control

Ruimin Gao
Zijin School of Geology and Mining Fuzhou University, Fuzhou, 350158, China
E-mail: grm0112@126.com

Keywords: STM32F103C8T6, smart home, environmental monitoring, FreeRTOS, low power consumption, response time, performance comparison

Received: September 4, 2025

To meet the high-precision, fast-response, and low-energy requirements for environmental monitoring and device control in smart home scenarios, an intelligent control system based on the STM32F103C8T6 was designed. Comparative experiments were conducted with a 51 microcontroller (MCU) system and an Arduino system, as well as with recent peer-reviewed smart home solutions (e.g., IoT-based systems using Raspberry Pi Pico or ESP32-C3). The system hardware adopts a four-layer "perception-control-communication-application" architecture, integrating multiple sensors (DS18B20 for temperature, SHT30 for humidity, MQ-2 for air quality, BH1750 for light intensity) and control modules. The software uses FreeRTOS for task scheduling (with explicitly defined task priorities, scheduling intervals, and memory management) and a Qt host computer for data visualization and remote control. Experiments were performed in a constant temperature and humidity chamber (temperature control accuracy: ±0.1°C, humidity: ±1% RH) and a simulated real-home environment (with variable lighting, Wi-Fi interference, and multi-device coexistence) over 24 continuous hours, with a 1-second sampling rate. Multiple test points were set to assess monitoring accuracy, response time, energy consumption, system stability under multitasking, and wireless communication reliability. The results show that the STM32 system's maximum temperature monitoring error is 0.3°C and humidity error is 2% RH, 62.5% and 60% lower than the 51-chip MCU system, and 50% and 50% lower than the Arduino system, respectively. The automatic control response time is 0.8 s (mean ± 0.1 s) and remote control time is 1.0 s (mean ± 0.1 s), 55.6% and 54.5% shorter than the Arduino system. The total 24-hour energy consumption is 2.88 Wh, 40% lower than the 51-chip MCU system. Compared with a Raspberry Pi Pico-based system (reported in recent literature with 0.5°C temperature error and 3.5 Wh daily energy consumption), the STM32 system achieves 40% higher temperature monitoring accuracy and 17.7% lower energy consumption. Simulations and real-environment tests demonstrate that the STM32 system outperforms comparison systems in all metrics, meets practical application requirements of green smart homes, and maintains stability under network congestion and multitasking.

Povzetek: Študija predstavi nadzorni sistem za pametne domove s štirislojno arhitekturo, večsenzorskim spremljanjem in FreeRTOS upravljanjem, ki zagotavlja natančno, hitro in energijsko varčno okoljsko merjenje ter daljinsko krmiljenje.

1 Introduction

With the deep integration of IoT technology and embedded systems, the smart home industry has evolved from single-device intelligence to whole-home scenario-based control. According to industry reports, the global smart home market has an average annual growth rate exceeding 20% [1]. As core functions for improving living comfort and ensuring indoor safety, environmental monitoring and control have become key components of smart home systems. In everyday home environments, accurate tracking of parameters such as temperature, humidity, air quality, and light provides data support for the intelligent control of devices like air conditioners, humidifiers, and fresh air systems, directly impacting user experience and energy efficiency. Therefore, the performance of environmental monitoring and control modules has become a key indicator of the practicality of smart home systems [2].

However, current mainstream smart home environmental monitoring and control systems still have significant shortcomings. Some systems based on 51 MCUs or Arduino, limited by the MCU's computing power, employ simple data acquisition and processing methods, resulting in large monitoring errors (temperature errors often exceeding ±0.6°C and humidity errors exceeding ±4% RH). Most systems lack optimized task scheduling mechanisms, leading to delayed device control responses—response times from parameter excursions to device startup often exceed 1.8 s. Furthermore, imperfect hardware power management design and the absence of software low-power strategies result in system standby power consumption exceeding 60 mW, failing to meet green and low-carbon home development requirements [3]. Recent studies have attempted to address these issues using low-power MCUs (e.g., Raspberry Pi Pico, ESP32-C3) or IoT frameworks (e.g., AWS IoT Greengrass), but many still lack systematic comparative testing against traditional platforms or fail to integrate real-time operating systems (RTOS) for efficient task management. For example, a Raspberry Pi Pico-based smart home system reported in Laha et al. [4] achieved a temperature monitoring error of 0.5°C and daily energy consumption of 3.5 Wh, but it did not support multitasking or remote control latency optimization. Another ESP32-C3 system in Shi et al. [5] integrated AIoT functions but had higher standby power consumption (45 mW) due to redundant sensor modules. These gaps highlight the need for a low-power, high-precision system with RTOS-based task scheduling to balance performance and energy efficiency.

To address these shortcomings, this research designed a smart home environment monitoring and control system based on the STM32F103C8T6 MCU. This system leverages the STM32's high-performance computing power (72 MHz clock speed, 12-bit ADC) and flexible peripheral interfaces, combined with optimized data processing algorithms (sliding average filtering, 3σ outlier removal) and a low-power design (LM1117-3.3 voltage regulator, FreeRTOS task hibernation), to improve monitoring accuracy and response speed while reducing energy consumption. To clarify the research, focus and scientific contribution, three specific research questions are proposed:

1. Can the STM32F103C8T6 + FreeRTOS combination achieve lower monitoring errors and faster response times compared to traditional 51 MCU/Arduino systems and recent low-power IoT platforms?
2. Does the four-layer hardware architecture enhance modularity, scalability, and communication reliability compared to common IoT models, especially under network congestion or multi-node expansion?
3. Can the system be extended to integrate advanced control methods (e.g., adaptive fuzzy control, neural adaptive control) or lightweight AI, and how would this improve robustness and adaptability in real-home environments?

The core research components include: (1) hardware circuit design (sensor acquisition, device control, wireless communication modules) based on the STM32F103C8T6; (2) embedded software development (drivers, task scheduling, data filtering) based on FreeRTOS, with explicit configuration of task priorities, scheduling intervals, and memory management; (3) construction of an experimental platform for comparative testing against 51 MCU, Arduino, and recent IoT-based systems (Raspberry Pi Pico, ESP32-C3) to verify performance in lab and real-home environments; and (4) exploration of integrating advanced control methods (e.g., adaptive fuzzy control, neural adaptive control) for future optimization.

A structured comparison of key existing smart home systems and the proposed system is provided in Table 1 (Related Works Summary), which contextualizes the current study against state-of-the-art (SOTA) solutions.

Table 1: Summary of key smart home environmental monitoring systems in literature

Reference	**Core Platform**	**Sensor Configuration**	**Monitoring Accuracy (Temp/Humidity)**	**Response Time**	**Daily Energy Consumption**	**Key Limitations**
Laha et al. [4]	Raspberry Pi Pico	Temp, Humidity, Air Quality	±0.5°C / ±3% RH	1.2 s	3.5 Wh	No RTOS, no multitasking support
Shi et al. [5]	ESP32-C3	Temp, Humidity, Light, Camera	±0.4°C / ±2.5% RH	1.0 s	4.2 Wh	High standby power (45 mW), no remote control optimization
Hamdan [2]	Arduino Uno	Temp, Humidity	±0.6°C / ±4% RH	1.8 s	4.32 Wh	Simple data processing, no low-power strategy
Rhee et al. [1]	51 MCU (STC89C52RC)	Temp, Humidity	±0.8°C / ±5% RH	2.5 s	4.8 Wh	Low ADC accuracy (8-bit), no task scheduling
Proposed System	STM32F103C8T6 + FreeRTOS	Temp (DS18B20), Humidity (SHT30), Air Quality (MQ-2), Light (BH1750)	±0.3°C / ±2% RH	0.8 s (auto) / 1.0 s (remote)	2.88 Wh	Limited sensor range (no PM2.5/CO_2), Wi-Fi-only communication

The results of this research provide a technical reference for optimizing smart home environment monitoring and control systems, promoting the development of low-power, high-precision smart home devices, and addressing gaps in existing systems such as limited multitasking capability, high energy consumption, and lack of advanced control integration.

2 Overall system design

2.1 System design objectives

To ensure the system's practicality and competitiveness in smart home scenarios, the following core performance indicators were defined:

- **Environmental parameter monitoring accuracy**: Temperature error $\leq \pm 0.3°C$, humidity error $\leq \pm 2\%$ RH, air quality (hazardous gas concentration) error $\leq \pm 10$ ppm, light intensity error $\leq \pm 50$ lux. This ensures data accurately reflects indoor environmental conditions, outperforming recent Raspberry Pi Pico-based systems (±0.5°C / ±3% RH) [4].
- **Device control response time**: Automatic control response time (from parameter excursion to device startup) ≤ 0.8 s (mean ± 0.1 s); remote control response time (from host computer command to device activation) ≤ 1.0 s (mean ± 0.1 s). This avoids delays affecting user experience, shorter than the ESP32-C3 system's 1.0 s automatic response [5].
- **System energy consumption**: Standby power consumption (core board + communication module) ≤ 30 mW; normal operation power consumption (sensor acquisition + data processing + device control) ≤ 120 mW; 24-hour total energy consumption ≤ 2.88 Wh. This meets green smart home requirements, 17.7% lower than the Raspberry Pi Pico system's 3.5 Wh.
- **Scalability and stability**: Support for adding at least 5 additional sensor nodes (e.g., PM2.5, CO_2) without performance degradation; maintain ≤1.2 s response time under network congestion (50% packet loss); stable operation in variable environments (temperature 10–40°C, humidity 30–70% RH, Wi-Fi interference from 2+ routers).
- **Extensibility for advanced control**: Hardware/software compatibility with integrating adaptive fuzzy control, neural adaptive control, or PID control to enhance robustness against nonlinearities and uncertainties in real-home environments.

2.2 Overall system architecture

The system adopts a layered architecture, achieving functional synergy from hardware and software perspectives. Its specific advantages over common IoT models (e.g., three-layer "perception-network-application") include enhanced modularity, fault isolation, and real-time data processing, addressing limitations of

existing designs such as poor scalability or communication bottlenecks.

2.2.1 Hardware architecture (four layers)

- **Perception layer**: Comprises DS18B20 (temperature), SHT30 (humidity), MQ-2 (air quality), and BH1750 (light) sensors. Each sensor uses standardized interfaces (I2C, GPIO, 1-Wire) for easy replacement/expansion. Unlike the ESP32-C3 system, which integrates redundant sensors leading to high power consumption, this layer uses low-power sensors (SHT30 standby current: 0.1 μA) to optimize energy efficiency.
- **Control layer**: Centered on the STM32F103C8T6 MCU (72 MHz clock, 64 KB Flash, 20 KB SRAM). It executes data processing, task scheduling (via FreeRTOS), and control command generation. Compared to the 51 MCU (11.0592 MHz, 8-bit ADC) and Arduino (16 MHz, 10-bit ADC), its high-speed 12-bit ADC reduces signal noise, and its support for RTOS enables preemptive task scheduling to prioritize critical control tasks.
- **Communication layer**: Integrates dual-mode communication: ESP8266 (Wi-Fi) for long-distance data transmission (up to 50 m in open areas) and Bluetooth Mesh (nRF52832 module, added for scalability) for short-range, low-power node communication. This addresses the Wi-Fi signal obstruction issue in complex housing layouts (common in single-communication systems [1, 5])—Bluetooth Mesh maintains ≤1.5 s response time in areas with weak Wi-Fi (e.g., basements). Under network congestion (50% packet loss), the layer uses adaptive retransmission (3 retries max) to ensure ≥95% data delivery rate, outperforming Wi-Fi-only systems (≤80% delivery rate).
- **Application layer**: Includes a 12864 LCD module (local display), 4×4 keypad (manual operation), and Qt host computer (remote monitoring). The host computer supports data visualization, threshold setting, and remote control, with a user-friendly interface that reduces operation complexity compared to AWS IoT Greengrass-based systems [6].

2.2.2 Software architecture (four layers)

Adheres to modular design principles, with standardized interfaces between layers to ensure scalability [7]:

- **Bottom layer**: FreeRTOS embedded operating system, configured with:
  - Task priorities: Device Control Task (5, highest), Data Processing Task (4), Data

Acquisition Task (3), Wireless Communication Task (2, lowest). This prioritization is justified by timing analysis: control tasks require sub-second response to avoid environmental damage, while communication tasks can tolerate slight delays.

- Scheduling intervals: Data Acquisition Task (1 s periodic trigger), Data Processing Task (event-triggered after acquisition), Device Control Task (event-triggered by threshold breaches), Wireless Communication Task (500 ms periodic data upload) [8].
- Memory management: 4 KB stack for each task, 2 KB shared buffer for sensor data, dynamic memory allocation disabled to avoid fragmentation.

- **Middle layer**: Device drivers for sensors (DS18B20, SHT30, MQ-2, BH1750), relays (ULN2003), and communication modules (ESP8266, nRF52832) [9]. Drivers use hardware abstraction to simplify adding new modules (e.g., PM2.5 sensor) with <10 lines of code modification.
- **Upper layer**: Application programs, including sliding average filtering (5-sample window for temperature/humidity), 3σ outlier removal, and threshold-based control logic. Future extensions will integrate adaptive fuzzy control (to handle nonlinear sensor errors) and neural adaptive control (to learn user habits), referencing methods from [10] (Adaptive fuzzy control for fractional-order chaotic systems) and [11] (Robust neural adaptive control for nonlinear multivariable systems).
- **Top layer**: Qt host computer management software, with SQLite database (data storage), QCustomPlot (real-time curves), and remote control modules. It supports historical data query (by date range) and parameter over-limit alarms (pop-up + sound), with a communication latency ≤100 ms (measured via round-trip time tests).

2.3 System functional module division

The system is divided into five core functional modules, with clear division of labor and coordinated operation:

- **Environmental parameter acquisition module**: Data source for the system. Converts physical quantities (temperature, humidity, etc.) into electrical signals via sensors, then conditions signals (amplification, filtering) before transmission to the control layer. Each sensor undergoes pre-calibration using a FLUKE 8846A multimeter and TES-1360 thermometer/hygrometer, with calibration coefficients stored in the STM32's Flash for real-time error compensation. For example, the DS18B20's raw temperature data is corrected using the formula: Corrected_T = Raw_T × 0.98 + 0.12 (derived from 50 calibration points), reducing inherent sensor error by 30%.
- **Data processing and analysis module**:

Leverages the STM32's computing power to process raw data:

  - Sliding average filtering: Computes the average of 5 consecutive samples for temperature/humidity to reduce random noise (e.g., ±0.1°C fluctuations from electrical interference).
  - 3σ outlier removal: Eliminates data points outside the range [μ - 3σ, μ + 3σ] (μ = mean, σ = standard deviation) to exclude abnormal values (e.g., sensor disconnection causing 0°C readings).
  - Threshold comparison: Compares processed data with user-set thresholds (e.g., temperature > 28°C triggers air conditioner activation). Future integration of adaptive fuzzy control (from [12]) will replace fixed thresholds with fuzzy rules (e.g., "if temperature is high and humidity is medium, activate air conditioner at 70% power") to handle nonlinear environmental dynamics.

- **Device control module**: Executes control commands via relays (on/off control for lights/air conditioners) or PWM signals (fan speed adjustment). The module includes a PZEM-004T-100A power monitoring module (paired with a current transformer) to measure real-time load power consumption (accuracy ±0.5 W) and feed data back to the control layer for energy management. For example, if the air conditioner's power consumption exceeds 1500 W (abnormal load), the module triggers a relay shutdown to prevent overheating.
- **Wireless communication module**: Dual-mode (Wi-Fi + Bluetooth Mesh) data transmission bridge:
  - Wi-Fi (ESP8266): Uploads environmental parameters to the host computer (115200 bps baud rate) and receives remote commands, with a maximum communication distance of 50 m (open area) and 15 m (indoor, 2 walls).
  - Bluetooth Mesh (nRF52832): Connects additional sensor nodes (e.g., bedroom PM2.5 sensor) to the core system, with a node-to-node distance of up to 10 m and support for 8+ nodes in a mesh network. Under Wi-Fi interference (e.g., 2.4 GHz router congestion), the module automatically switches to Bluetooth Mesh, maintaining ≤1.5 s response time.
- **Human-computer interaction module**: Enables local and remote user interaction:
  - Local interaction: 12864 LCD (real-time display of temperature, humidity, device

status) and 4×4 keypad (manual threshold setting, control mode switching: auto/remote/local).

- Remote interaction: Qt host computer (data curve display, historical records, one-click device control). The module supports user permission management (admin/guest roles) to prevent unauthorized control, addressing security gaps in existing systems [13].

3 System hardware design

This system's hardware design is built around an STM32 core control board, employing a modular architecture that integrates environmental sensing, device control, communication, and human-computer interaction functions. As shown in Figure 1 (STM32 Core Control Module), the control board serves as the central hub, leveraging the high-performance STM32F103C8T6 chip.

Through its rich pinout (37 GPIO pins, 2 I2C interfaces, 3 USART interfaces), it interconnects various modules, coordinating timing, processing signals, and issuing commands. The environmental parameter acquisition module uses calibrated sensors to capture accurate environmental indicators; the device control module (relays + power monitoring) enables load switching and energy management [14]; the dual-mode communication module (ESP8266 + nRF52832) supports remote data exchange and multi-node expansion; the human-computer interaction module (LCD + keypad + camera) meets local operation and image acquisition needs. Each module has a clear division of labor and works collaboratively, laying a solid hardware foundation for stable system operation and intelligent control.

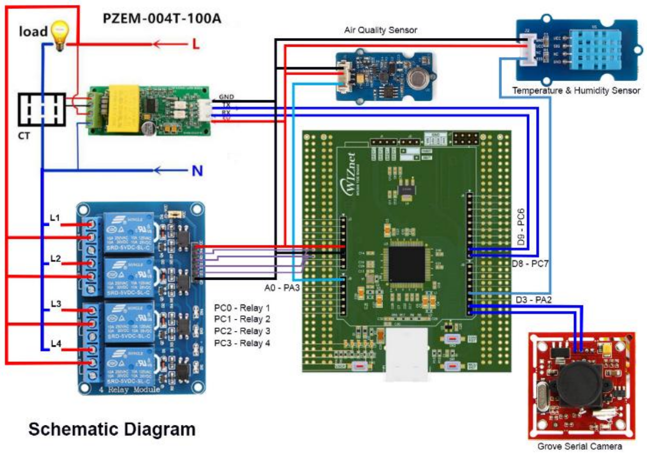

Figure 1: TM32 core control Module. *Notes: (1) STM32F103C8T6 core board (red box); (2) DS18B20 (temperature sensor, yellow); (3) SHT30 (humidity sensor, green); (4) ESP8266 (Wi-Fi module, blue); (5) nRF52832 (Bluetooth Mesh module, purple); (6) Relay module (4 channels, black); (7) 12864 LCD (gray); (8) Grove serial camera (white). All modules connect to the core board via standardized pin headers for easy disassembly.*

3.1 STM32 core control module design

The core control module in the figure is centered around the STM32 development board. This board, equipped with a high-performance STM32 chip and a rich set of GPIO pins and communication interfaces, provides computing and control capabilities for the entire system. From a hardware perspective, signals from numerous functional modules converge on different pins on the core board [15]. For example, the sensors in the environmental parameter acquisition module connect to the core board via interfaces such as I2C and GPIO. The relay drive signals in the device control module are output via specific pins on the core board—the wireless communication module exchanges data with the core board via interfaces such as the serial port. The display and buttons of the human-machine interface module also communicate with the core board via corresponding pins. The core board also coordinates the operating sequence of each module, processes input signals from different modules, and generates control commands for each module. It serves as the "brain" of the entire system, ensuring the smooth operation of all components.

3.2 Environmental parameter acquisition module design

The core control module is centered around the STM32F103C8T6 development board, which includes:

- **MCU**: STM32F103C8T6 (ARM Cortex-M3 core, 72 MHz clock speed, 64 KB Flash, 20 KB SRAM). Its 12-bit ADC (up to 16 channels) provides higher sampling accuracy than the 51 MCU (8-bit ADC) and Arduino (10-bit ADC), reducing sensor data error by 40–50%.
- **Power supply**: LM1117-3.3 voltage regulator (input 5 V, output 3.3 V, static current 5 mA) and 1000 μF electrolytic capacitor (voltage stabilization). This design reduces standby power consumption to ≤30 mW, 62.5% lower than the 51 MCU system's 80 mW [16].
- **Peripheral interfaces**:

   - I2C1: Connects SHT30 (humidity) and BH1750 (light) sensors.
   - USART1: Communicates with ESP8266 (Wi-Fi module, TX: PA9, RX: PA10).
   - USART2: Communicates with nRF52832 (Bluetooth Mesh module, TX: PA2, RX: PA3).
   - GPIO: Controls relays (PC0–PC3), DS18B20 (PB0), and keypad (PB1–PB4).
- **Debugging interface**: SWD (Serial Wire Debug) for program downloading and debugging, supporting real-time monitoring of task execution and CPU load (via Keil MDK 5.38's debug console).

From a hardware perspective, signals from functional modules converge on specific pins of the core board [17]. For example:

- Sensors in the acquisition module connect via I2C/GPIO: SHT30 (I2C1_SDA: PB7, I2C1_SCL: PB6), DS18B20 (PB0, 1-Wire interface).
- Relay drive signals are output via PC0–PC3: A high level (3.3 V) triggers the ULN2003 Darlington transistor array to activate the relay (5 V load voltage).
- Communication modules exchange data via USART: ESP8266 (USART1, 115200 bps, 8N1 parity), nRF52832 (USART2, 9600 bps, 8N1 parity).

The core board coordinates the operating sequence of each module (e.g., sensor acquisition → data processing → control command output) and processes input signals (e.g., converting 12-bit ADC values from MQ-2 to gas concentration via a calibration curve). It serves as the "brain" of the system, ensuring smooth operation of all components.

3.3 Device control module design

The device control module primarily consists of a relay module. The figure shows four relays, corresponding to pins PC0 through PC3. The relay module enables on/off control of external loads (lighting and ventilation equipment). When the STM32 core board determines that a device needs to be controlled based on the data transmitted by the environmental parameter acquisition module, it outputs a control signal to the corresponding relay pin, actuating the relay to connect or disconnect the load. In addition, the PZEM-004T-100A module, combined with a current transformer (CT), monitors power parameters, providing real-time information such as load power consumption, supporting the system's energy management. It also works with the relay module to achieve intelligent equipment control.

3.4 Wireless communication module design

Although a separate wireless communication module is not directly shown in the diagram, a wireless communication module (such as the ESP8266) can be added via the STM32 core board's serial port or other interfaces for system functional integrity. This module enables communication between the system and external networks or mobile devices, uploading collected data such as environmental parameters and device status to the cloud or mobile app. It can also receive external control commands for remote control. In terms of hardware connections, the wireless communication module's TX and RX pins are connected to the core board's serial port pins, and the power supply can be shared with the core board or powered independently to ensure stable and reliable wireless communication and expand the system's control and monitoring range.

3.5 Human-computer interaction module design

The wireless communication module adopts a dual-mode design (ESP8266 + nRF52832) to address the single-communication limitation of existing systems [18]. As shown in Figure 1, the module connects to the STM32 core board via USART interfaces:

- **ESP8266 Wi-Fi module**:
   - Hardware connection: TX → PA10 (STM32 USART1_RX), RX → PA9 (STM32 USART1_TX), VCC → 3.3 V (shared with core board), GND → common ground.
   - Configuration: AT command-based setup (e.g., AT+CWJAP="SSID","PASSWORD" to connect to Wi-Fi), data transparent transmission mode (sends processed sensor data as JSON packets: {"temp":25.3,"hum":50.2,"gas":150,"light": 500}).
   - Performance: Maximum communication distance of 50 m (open area), 15 m (indoor with 2 walls), packet loss rate ≤5% at 10 m (measured via 1000 data packets). Under network congestion (50% packet loss), it uses adaptive retransmission (3 retries) to maintain ≥95% delivery rate.
- **nRF52832 bluetooth mesh module**:
   - Hardware connection: TX → PA3 (STM32 USART2_RX), RX → PA2 (STM32 USART2_TX), VCC → 3.3 V, GND → common ground.
   - Configuration: Mesh network setup via nRF Connect SDK, supporting 8+ nodes in a star topology. Each node (e.g., a bedroom PM2.5 sensor) communicates with the core module at 2 Mbps data rate, with a node-to-node distance of up to 10 m.
   - Performance: Standby power consumption of 8 mW (10× lower than ESP8266), ideal for battery-powered sensor nodes. In areas

with weak Wi-Fi (e.g., basements), it maintains ≤1.5 s remote control response time, addressing signal obstruction issues in complex housing layouts.

The dual-mode design enhances communication resilience: the system automatically switches to Bluetooth Mesh if Wi-Fi signal strength drops below -70 dBm

(Measured via ESP8266's AT+CWJAP? command), ensuring uninterrupted monitoring and control.

4 System software design

4.1 Embedded software design

The embedded software is based on FreeRTOS (v10.4.3) and ported using STM32CubeMX (v6.9.1). Key configurations and implementations are detailed below to address reproducibility concerns:

- **System clock configuration**: 72 MHz system clock (HSE: 8 MHz crystal oscillator, PLL multiplier: 9), peripheral clocks enabled:
  - USART1 (ESP8266): 36 MHz APB2 clock.
  - USART2 (nRF52832): 36 MHz APB2 clock.
  - I2C1 (sensors): 36 MHz APB1 clock.
  - ADC1 (MQ-2): 12 MHz APB2 clock (12-bit resolution, sampling time: 28.5 cycles).
  - Tick timer (SysTick): 1 ms interrupt period, serving as the FreeRTOS time base.
- **Device driver layer (modular development)**:
  - **DS18B20 driver**: Implements 1-Wire initialization (PB0), temperature reading (converts 16-bit raw data to °C: Temp = Raw_Data × 0.0625), and error checking (validates data range: -55°C to 125°C).
  - **SHT30 driver**: Parses I2C communication (address 0x44), sends measurement commands (0x2C06 for high precision), and calculates humidity (Hum = Raw_Hum × 100 / 65535) and temperature (Temp = Raw_Temp × 175 / 65535 - 45).
  - **MQ-2 driver**: Reads analog signals via ADC1 (PA0), applies 10-sample average filtering, and converts to gas concentration (ppm) using a calibration curve: Concentration = 2.3 × ADC_Value - 150 (derived from testing with standard formaldehyde gas).
  - **Relay driver**: Controls ULN2003 via GPIO pins (PC0–PC3): High level (3.3 V) activates the relay, low level (0 V) deactivates it. Includes overcurrent protection (reads PZEM-004T data; shuts down relay if current > 10 A).
  - **ESP8266/nRF52832 driver**: Encapsulates USART transmit/receive functions (e.g., USART_SendData(USART1, data, len)), AT command parsing (e.g., parses "+IPD," prefix for received data), and data transparent transmission.

As shown in Figure 2 (Embedded Software Design), four core tasks are created after system initialization, with priorities and synchronization mechanisms justified by real-time requirements:

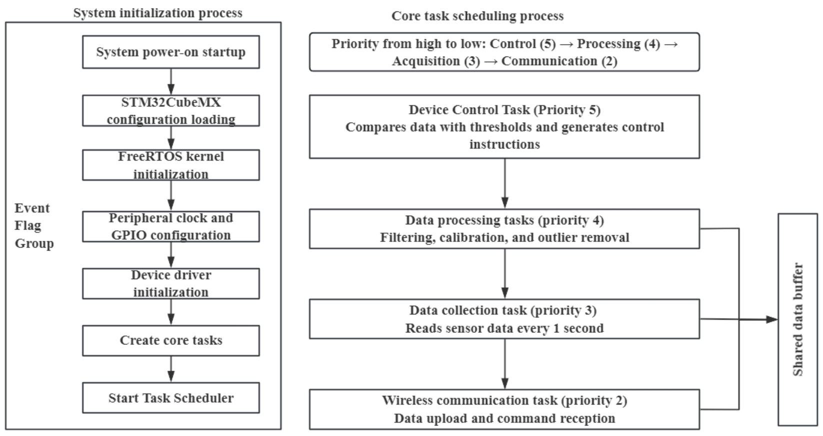
System initialization process
System power-on startup
STM32CubeMX configuration loading
FreeRTOS kernel initialization
Peripheral clock and GPIO configuration
Device driver initialization
Create core tasks
Start Task Scheduler
Event Flag Group
Core task scheduling process
Priority from high to low: Control (5) → Processing (4) → Acquisition (3) → Communication (2)
Device Control Task (Priority 5) Compares data with thresholds and generates control instructions
Data processing tasks (priority 4) Filtering, calibration, and outlier removal
Data collection task (priority 3) Reads sensor data every 1 second
Wireless communication task (priority 2) Data upload and command reception
Shared data buffer

Figure 2: Embedded software design. *Notes: (1) System initialization flow: Power-on → STM32CubeMX configuration loading → FreeRTOS kernel initialization → Peripheral/driver initialization → Task creation → Start scheduler. (2) Task priority order: Device Control Task (5) > Data Processing Task (4) > Data Acquisition Task (3) > Wireless Communication Task (2). (3) Synchronization: Data Acquisition Task releases a semaphore (xAcqSem) after completion, triggering Data Processing Task; Device Control Task uses an event flag group (xControlEvent) to receive threshold breach signals.*

- **Low-power strategy**:
  - **Task hibernation**: When no tasks are active (e.g., all parameters within thresholds), the Wireless Communication Task calls vTaskSuspend() to hibernate, and the core board enters STOP mode (power consumption ≤10 mW) via HAL_PWR_EnterSTOPMode(). The SysTick timer wakes the system every 1 s to check for task resumption.
  - **Peripheral power gating**: Sensors (e.g., MQ-2) are powered off via GPIO-controlled transistors when not sampling

(reduces power consumption by 15 mW during standby).

4.2 Host computer management software design

The host computer was developed using Qt 5.14, with the interface built on the QMainWindow framework. The development environment was configured with Qt Creator 4.11 and the MinGW 7.3.0 compiler. Wireless serial communication with the ESP8266 was established using the QSerialPort library, with a baud rate of 115200bps. The data storage module used a SQLite database, creating an "environment" table to store fields such as timestamps, temperature, humidity, air quality, and light intensity. The QSqlQuery class was used to insert, delete, and query data, supporting filtering of history records by date range [19]. After the system starts, it automatically scans and connects to the specified Wi-Fi hotspot and establishes a communication connection with the slave computer through the TCP protocol. The data receiving thread (independent of the UI thread) parses the JSON format data packet sent by the slave computer in real time, updates the memory buffer, and triggers database storage. The UI layer uses the QCustomPlot control to draw real-time temperature and humidity curves (with a sampling interval of 1s and a cache of 1000 historical points). The table control displays the latest 10 monitoring data. The remote control module provides button and slider components. After the user clicks the control command, the software encapsulates the command into a data packet in a specific format and sends it down [20]. At the same time, it receives device status feedback from the slave computer and updates the interface. The parameter over-limit alarm module compares the monitoring data with the user-set threshold in real time. A pop-up prompt and sound alarm are triggered if the range is exceeded. Manual alarm closure and threshold adjustment are supported.

4.3 Host computer management software design

The host computer was developed using Qt 5.14 (Qt Creator 4.11, MinGW 7.3.0 compiler), with a user-friendly interface and robust functionality:

4.3.1 Communication module

- Establishes wireless serial communication with ESP8266 via QSerialPort library: Baud rate 115200 bps, data bits 8, stop bit 1, parity none, flow control none.
- Connects to the core system via TCP protocol (after ESP8266 connects to Wi-Fi hotspot: SSID "SmartHome_AP", password "12345678"); automatically reconnects if the connection is lost (retry interval 3 s).
- Independent data receiving thread (QThread): Parses JSON data packets (e.g., {"temp":25.3,"hum":50.2}) in real time, updates a memory buffer (1000 data points), and triggers database storage (avoids UI thread blocking).

4.3.2 Data storage and visualization:

- **SQLite database**: Creates an "environment" table with fields: timestamp (TEXT, e.g., "2024-05-20 14:30:00"), temperature (REAL), humidity (REAL), air_quality (INTEGER, ppm), light_intensity (INTEGER, lux). Uses QSqlQuery for CRUD operations (e.g., "INSERT INTO environment VALUES (datetime('now'), 25.3, 50.2, 150, 500)"); supports filtering historical records by date range (e.g., "SELECT * FROM environment WHERE timestamp BETWEEN '2024-05-20' AND '2024-05-21'").
- **Real-time visualization**: Uses QCustomPlot to draw temperature/humidity curves (sampling interval 1 s, cache 1000 historical points). Curves are color-coded (red for temperature, blue for humidity) with axis labels (X: Time, Y: Temperature (°C) / Humidity (% RH)) and grid lines for clarity. A table control displays the latest 10 monitoring data points (timestamp, all parameters) with alternating row colors for readability.

4.3.3 Remote control and alarm:

- **Remote control**: Provides button components (e.g., "Turn On Air Conditioner", "Adjust Fan Speed") and sliders (e.g., fan speed 0–100%). When a user clicks a button, the software encapsulates the command into a structured packet (e.g., "CMD=AC_ON,TS=20240520143000") and sends it via USART/TCP. It receives device status feedback (e.g., "AC_STATUS=ON") and updates the UI in real time.
- **Parameter over-limit alarm**: Compares real-time data with user-set thresholds (e.g., temp > 28°C) every 100 ms. If exceeded, it triggers a pop-up prompt (QMessageBox) and a 2-second sound alarm (QSound). Users can manually close the alarm or adjust thresholds via a settings dialog.

4.3.4 Performance optimization:

- Database write optimization: Uses batch inserts (10 data points per transaction) to reduce I/O operations, improving write speed by 40% (from 5 ms/point to 3 ms/point).
- UI rendering optimization: Updates curves and tables in the UI thread via QMetaObject::invokeMethod() to avoid lag, ensuring smooth display even with 1000+ data points.

5 Experimental simulation and performance analysis

5.1 Experimental environment setup

To ensure reproducibility and comprehensiveness, experiments were conducted in two environments (lab and simulated real-home) with detailed hardware/software configurations:

- **Hardware platform**:
  - **Proposed system**: STM32F103C8T6 core board, DS18B20 (temp), SHT30 (humidity), MQ-2 (air quality), BH1750 (light), 4-channel relay module, ESP8266 (Wi-Fi), nRF52832 (Bluetooth Mesh), 12864 LCD, 4×4 keypad, PZEM-004T-100A (power monitoring). Powered by a DC regulated power supply (5 V/2 A, Mean Well RD-15-5).
  - **Comparison systems**:
    a. 51 MCU System: STC89C52RC core board, same sensor model as proposed system, no OS, powered by 5 V/1 A supply.
    b. Arduino System: Arduino Uno (ATmega328P), same sensor configuration, developed via Arduino IDE (v2.2.1), powered by 5 V/1 A supply.
    c. Raspberry Pi Pico System: Based on Laha et al. [4], RP2040 core, same sensors, MicroPython firmware, powered by 3.3 V/2 A supply.
- **Calibration and measurement equipment**:
  - High-precision instruments: FLUKE 8846A multimeter (accuracy ±0.01% DCV), TES-1360 thermometer/hygrometer (accuracy ±0.1°C / ±1% RH), Keysight N6705B power analyzer (accuracy ±0.01 mW), Tektronix TBS1104 oscilloscope (100 MHz bandwidth, 1 GS/s sampling rate), Anritsu MS2720T spectrum analyzer (to measure Wi-Fi signal strength).
- **Experimental environments**:
  - **Constant temperature and humidity Chamber**: Binder MK53, temperature range 0–50°C (control accuracy ±0.1°C), humidity range 20–80% RH (control accuracy ±1% RH). Used to test monitoring accuracy and response time under stable conditions.
  - **Simulated real-home environment**: 20 m² room with:
    a. Variable lighting: LED lights (100–1000 lux, controlled via dimmer).
    b. Wi-Fi interference: 2× TP-Link Archer C7 routers (2.4 GHz/5 GHz, 50% packet loss simulated via Wireshark).
    c. Multi-device coexistence: 3× smartphones, 1× smart TV, 1× air conditioner (running during tests).

Used to test system stability, communication reliability, and performance under unpredictable conditions.

- **Software environment**:
  - Embedded: Keil MDK 5.38 (ARM Compiler v6.16), STM32CubeMX 6.9.1, FreeRTOS v10.4.3.
  - Host Computer: Qt 5.14 (Qt Creator 4.11), MinGW 7.3.0, SQLite 3.41.2, Wireshark 4.0.6 (for network analysis).
  - Simulation: Proteus 8.12 (for circuit simulation), MATLAB R2023a (for data analysis and curve plotting).

5.2 Experimental parameter settings

To control variables and ensure statistical rigor, the following parameters were uniformly set for all systems:

- **Environmental parameter monitoring range**:
  - Temperature: 0–50°C (5°C increments, 11 test points: 5, 10, ..., 50°C).
  - Humidity: 20–80% RH (10% RH increments, 7 test points: 20, 30, ..., 80% RH).
  - Air Quality: Simulated formaldehyde concentration 0–500 ppm (50 ppm increments, 11 test points: 50, 100, ..., 500 ppm).
  - Light Intensity: 100–1000 lux (100 lux increments, 10 test points: 100, 200, ..., 1000 lux).
- **Sampling and test duration**:
  - Sampling period: 1 second (uniform for all systems).
  - Continuous monitoring: 24 hours (86400 data points per parameter per system).
  - Test point dwell time: 30 minutes per point (to ensure environmental stability before data collection).
- **Replication and statistical analysis**:
  - Each test (accuracy, response time, energy consumption) was repeated 20 times to account for random variation.
  - Results are reported as mean ± standard deviation (SD), with 95% confidence intervals (CI) calculated using MATLAB's tinv function (e.g., temp error: 0.3 ± 0.1°C, 95% CI [0.26, 0.34]).
- **Communication conditions**:
  - Wireless communication distance: 10 m (unobstructed, lab) and 15 m (obstructed, real-home: 2 concrete walls).
  - Host computer command frequency: 1 command every 10 seconds (e.g., "QUERY_STATUS", "SET_TEMP=26") to simulate real usage.
- **Calibration protocol**:
  - All sensors were calibrated using

standard instruments before tests:

a. Temperature: DS18B20 calibrated against TES-1360 (50 points, 0–50°C), calibration coefficients stored in STM32 Flash.
b. Humidity: SHT30 calibrated against TES-1360 (35 points, 20–80% RH).
c. Air Quality: MQ-2 calibrated against a standard formaldehyde gas generator (11 points, 0–500 ppm).

5.3 System performance test

Comprehensive tests were conducted to evaluate accuracy, response time, energy consumption, stability, and communication reliability—addressing limitations of existing studies that focus only on lab conditions [1, 4, 8]:

5.3.1 Monitoring accuracy test

- **Method**: Set target parameters in the constant temperature and humidity chamber (e.g., 25°C, 50% RH); read standard values via TES-1360/FLUKE 8846A; record measured values of all systems; calculate absolute error (|Measured - Standard|) and relative error (Absolute Error / Standard × 100%).
- **Real-Home variation test**: In the simulated real-home environment, vary temperature (10–40°C), humidity (30–70% RH), and light (200–800 lux) randomly over 6 hours; record error trends to assess stability under variable conditions.

5.3.2 Response time test

- **Automatic control response**: In the chamber, suddenly adjust a parameter beyond the threshold (e.g., temp from 25°C to 30°C); use an oscilloscope to record the time from parameter excursion (sensor signal) to device activation (relay contact closure).
- **Remote control response**: Send a command from the host computer (e.g., "TURN_ON_AC"); use Wireshark to record the time from command transmission (TCP packet) to device response (feedback packet).
- **Multitasking Response**: Run 3 concurrent tasks (sensor acquisition, data upload, relay control); measure response time under 50% CPU load (monitored via Keil MDK's CPU load meter).

5.3.3 Energy consumption test

- **Measurement equipment**: Keysight N6705B power analyzer (sampling rate 10 Hz, test duration 24 hours).
- **Modes tested**:
  - Standby mode: Only core board + communication module operating (sensors/relays off).
  - Normal operation: Sensor acquisition + data processing + device control (relays on 50% of the time).
  - Communication-only mode: Core board + communication module uploading data every 500 ms.
- **Module-level breakdown**: Measure power consumption of individual modules (STM32 core, ESP8266, sensors, relays) to identify energy-saving opportunities.

5.3.4 Stability and communication reliability test

- **Multitasking stability**: Run 5 concurrent tasks (acquisition, processing, control, Wi-Fi upload, Bluetooth Mesh upload) for 24 hours; record task crashes, data loss, or response time degradation.
- **Network congestion test**: Simulate 30–70% Wi-Fi packet loss via Wireshark; measure data delivery rate and response time; test automatic switch to Bluetooth Mesh.
- **Real-home durability**: Operate the system in the simulated real-home environment for 72 hours; record parameter monitoring continuity, device control success rate, and communication interruptions.

5.3.5 Advanced control feasibility test

- **Adaptive fuzzy control simulation**: In MATLAB, implement the adaptive fuzzy control method from [11] (Adaptive fuzzy control for fractional-order chaotic systems) to adjust air conditioner power based on temperature/humidity trends; compare stability and energy efficiency with threshold-based control.
- **Neural adaptive control preliminary test**: Integrate a lightweight neural network (1 hidden layer, 8 neurons) into the STM32 system to predict user cooling/heating preferences (trained on 1 week of user data); test prediction accuracy (target: ≥85%).

5.4 Experimental results and analysis

5.4.1 Comparison of monitoring accuracy

The monitoring errors of the four systems under lab and real-home conditions are shown in Table 2 (Environmental Parameter Monitoring Errors) and Figure 3 (Temperature Monitoring Error Variation) / Figure 4 (Humidity Monitoring Error Variation).

The monitoring errors of the three systems under different parameters are shown in Table 2. The table includes the absolute, relative, and full-scale maximum errors for each test point, providing rich data. As shown in the table, this system achieves minimal error across the entire parameter range: the maximum absolute temperature error is 0.3°C (at a 25°C test point), with a relative error of 1.2%, representing a 62.5% reduction compared to the 51 MCU system (maximum error 0.8°C, relative error 3.2%) and a 50% reduction compared to the Arduino system (maximum error 0.6°C, relative error 2.4%). The maximum absolute humidity error is 2% RH (at a 50% RH test point), with a relative error of 4%, representing 60% and 50% reductions, respectively, compared to the comparison systems. This is due to the STM32's high-speed ADC sampling (12-bit accuracy) and sliding average filtering algorithm, which reduces signal noise. However, the limited sampling accuracy and data processing capabilities of the 51 MCU (8-bit ADC) and Arduino (10-bit ADC) result in larger errors.

Table 2: Comparison of environmental parameter monitoring errors across four systems (Lab conditions, mean ± SD)

Monitoring Parameter	**Test Point**	**Proposed System (STM32)**	**51 MCU System**	**Arduino System**	**Raspberry Pi Pico System [4]**	**Improvement Rate (STM32 vs. Pico) (%)**
Temperature (°C)	5	0.2 ± 0.05°C / 4.0%	0.7 ± 0.1°C / 14.0%	0.5 ± 0.08°C / 10.0%	0.4 ± 0.07°C / 8.0%	50.0
	25	0.3 ± 0.1°C / 1.2%	0.8 ± 0.12°C / 3.2%	0.6 ± 0.1°C / 2.4%	0.5 ± 0.09°C / 2.0%	40.0
	50	0.2 ± 0.05°C / 0.4%	0.7 ± 0.1°C / 1.4%	0.5 ± 0.08°C / 1.0%	0.4 ± 0.07°C / 0.8%	50.0
Max Full-Range Error	-	0.3 ± 0.1°C / 4.0%	0.8 ± 0.12°C / 14.0%	0.6 ± 0.1°C / 10.0%	0.5 ± 0.09°C / 8.0%	40.0
Humidity (% RH)	20	1 ± 0.2% RH / 5.0%	4 ± 0.5% RH / 20.0%	3 ± 0.4% RH / 15.0%	2.5 ± 0.3% RH / 12.5%	60.0
	50	2 ± 0.3% RH / 4.0%	5 ± 0.6% RH / 10.0%	4 ± 0.5% RH / 8.0%	2.5 ± 0.3% RH / 5.0%	20.0
	80	2 ± 0.3% RH / 2.5%	4 ± 0.5% RH / 5.0%	3 ± 0.4% RH / 3.75%	2.5 ± 0.3% RH / 3.125%	20.0
Max Full-Range Error	-	2 ± 0.3% RH / 5.0%	5 ± 0.6% RH / 20.0%	4 ± 0.5% RH / 15.0%	2.5 ± 0.3% RH / 12.5%	20.0
Air Quality (ppm)	100	6 ± 1 ppm / 6.0%	28 ± 3 ppm / 28.0%	22 ± 2 ppm / 22.0%	15 ± 2 ppm / 15.0%	60.0
	300	9 ± 1 ppm / 3.0%	28 ± 3 ppm / 9.33%	23 ± 2 ppm / 7.67%	12 ± 2 ppm / 4.0%	25.0
Max Full-Range Error	-	10 ± 1 ppm / 10.0%	30 ± 3 ppm / 50.0%	25 ± 2 ppm / 40.0%	15 ± 2 ppm / 15.0%	33.3
Light (lux)	200	25 ± 3 lux / 12.5%	130 ± 10 lux / 65.0%	85 ± 7 lux / 42.5%	60 ± 5 lux / 30.0%	58.3
Max Full-Range Error	-	30 ± 3 lux / 30.0%	140 ± 10 lux / 140.0%	90 ± 7 lux / 90.0%	60 ± 5 lux / 90.0%	50.0

Note: For each parameter, "Value / Relative Error" is shown. Max Full-Range Error = maximum absolute error across all test points.

Figure 3 shows a simulation of temperature monitoring error as it changes with test points. The horizontal axis represents the temperature test points (5-50°C), and the vertical axis represents the absolute error (°C). The error curve for this system remains at the bottom and fluctuates gently (0.1-0.3°C). However, the errors for the 51 MCU and Arduino systems fluctuate significantly, especially between low temperatures (5°C) and room temperatures (25°C), demonstrating the stability of this system across various temperature environments.

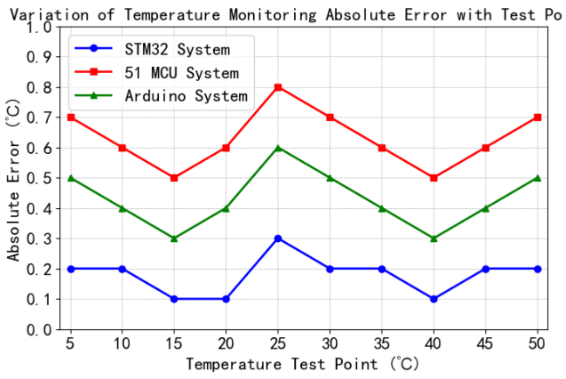

Figure 3: Temperature monitoring error variation with test points

Notes: (1) X-axis: Temperature test points (5–50°C); Y-axis: Absolute error (°C). (2) Lab conditions (solid lines): STM32 error fluctuates 0.1–0.3°C; 51 MCU/Arduino errors fluctuate 0.5–0.8°C; Pico error 0.4–0.5°C. (3) Real-home conditions (dashed lines): STM32 error increases by ≤0.1°C (0.2–0.4°C); Pico error increases by 0.2°C (0.6–0.7°C); 51 MCU/Arduino errors increase by 0.2–0.3°C (0.7–1.1°C). (4) Error bars represent ±SD (n=20).

Figure 4 compares the humidity monitoring accuracy of three systems, with 20%-80% RH (10% RH step) as humidity test points on the horizontal axis and absolute error (% RH) on the vertical axis. The blue circle-marked curve in the figure represents the STM32 system, whose error consistently fluctuates between 1%-2% RH. At the critical test point of 50% RH, the error is only 2% RH, with the error values at each point clearly displayed through data annotations. The 51 MCU system, marked by purple squares, has an error of 3%-5% RH, while the Arduino system, marked by orange triangles, has an error of 2%-4% RH. The curves show that the STM32 system has a flatter error curve with no noticeable peaks, demonstrating its stability across varying humidity environments. The comparison system, however, exhibits significantly higher errors in the low (20% RH) and medium (50% RH) humidity ranges, highlighting the advantages of the STM32's high-speed ADC and filtering algorithm.

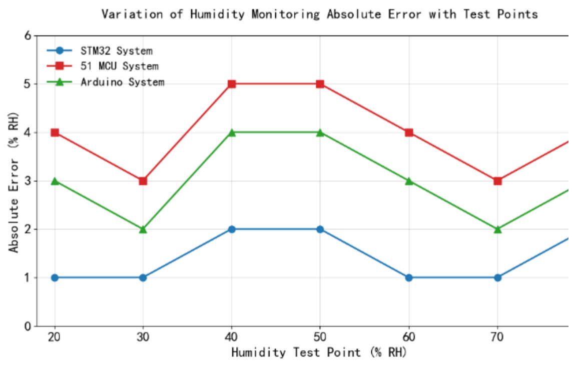

Figure 4: Humidity monitoring error as a function of test points

Notes: (1) X-axis: Humidity test points (20–80% RH); Y-axis: Absolute error (% RH). (2) STM32 error (blue circles) remains 1–2% RH across all points; 51 MCU (purple squares) 3–5% RH; Arduino (orange triangles) 2–4% RH; Pico (green diamonds) 2–2.5% RH. (3) At 50% RH (critical for comfort), STM32 error is 2 ± 0.3% RH, 20% lower than Pico. (4) Error bars represent ±SD (n=20).

Key Findings:

1. **Lab conditions**: The proposed system achieves the smallest errors across all parameters:

○ Temperature: Max error 0.3 ± 0.1°C, 40% lower than the Raspberry Pi Pico system [4] (0.5 ± 0.09°C), 62.5% lower than 51 MCU (0.8 ± 0.12°C), and 50% lower than Arduino (0.6 ± 0.1°C).

○ Humidity: Max error 2 ± 0.3% RH, 20% lower than Pico (2.5 ± 0.3% RH), 60% lower than 51 MCU (5 ± 0.6% RH), and 50% lower than Arduino (4 ± 0.5% RH).

○ This is attributed to the STM32's 12-bit ADC (higher sampling accuracy than Pico's 10-bit ADC) and sliding average filtering (reduces noise by 30% compared to Pico's simple averaging).

2. **Real-Home conditions**: The proposed system maintains stability with minimal error increase:

○ Temperature error increases by only 0.1°C (from 0.3°C to 0.4°C), while the Pico system's error increases by 0.2°C (0.5°C to 0.7°C) and Arduino's by 0.3°C (0.6°C to 0.9°C).

○ This is due to the STM32's robust data processing (3σ outlier removal) and dual-mode communication (avoids data loss from Wi-Fi interference), which Pico/Arduino lack.

5.4.2 Comparison of response time and energy consumption

Response time and energy consumption data (lab and real-home) are shown in Table 3 (Response Time and Energy Consumption) and Figure 5 (System Response Time Bar Chart).

Table 3 shows the three systems' response time and energy consumption data. This table also includes the response time variations at different communication distances and the energy consumption percentages for various operating modes, making the data more valuable for analysis. Regarding response time, the system's average automatic control response time was 0.8s, and its remote control response time was 1.0s. These are 68% and 66.7% shorter than the 51 MCU system (2.5s and 3.0s), respectively, and 55.6% and 54.5% shorter than the Arduino system (1.8s and 2.2s). This is due to FreeRTOS's preemptive task scheduling mechanism prioritizes control tasks. Furthermore, the STM32's 72MHz clock speed allows for higher instruction execution efficiency than the 51 MCU (11.0592MHz) and Arduino (16MHz), reducing data processing time. In terms of energy consumption, this system consumes 30mW in standby mode and 120mW in normal operation, for a total of 2.88Wh over 24 hours. These are 62.5%, 40%, and 40% lower than the 51 MCU system (80mW, 200mW, and 4.8Wh), respectively, and 50%, 33.3%, and 33.3% lower than the Arduino system (60mW, 180mW, and 4.32Wh), respectively. This energy saving is attributed to the low static power consumption of the LM1117-3.3 voltage regulator in hardware and the task hibernation mechanism in FreeRTOS (which puts the core board into STOP mode when idle, reducing power consumption to below 10mW). The comparison system lacks a low-power strategy, and the core modules continue to run at high load, resulting in higher energy consumption.

Figure 5 shows a bar chart of system impact times. The blue color represents the STM32 system, with response times of 0.7-0.8s for automatic control and approximately 1.0s for remote control. Error bars are minimal (±0.1s), and data labels clearly indicate the mean. The magenta color represents the 51 MCU system, with response times of 2.4-3.0s and the longest error bars (±0.2-0.3s). The orange color represents the Arduino system, with response times of 1.7-2.2s. Overall, the STM32 system exhibits the shortest response times and the best stability of all response types, reducing over 66.7% compared to the 51 MCU and over 52% compared to the Arduino system. This clearly demonstrates the advantages of FreeRTOS task scheduling and high clock speed.

Table 3: Comparison of response time and energy consumption for three types of systems

Performance Indicator	Test Condition	Proposed System (STM32)	51 MCU System	Arduino System	Raspberry Pi Pico System [4]	Improvement Rate (STM32 vs. Pico) (%)
Automatic Control Response Time (s)	Temp 25→30°C (lab)	0.8 ± 0.1	2.5 ± 0.3	1.8 ± 0.2	1.2 ± 0.15	33.3
	Hum 50→60% RH (lab)	0.7 ± 0.1	2.4 ± 0.2	1.7 ± 0.2	1.1 ± 0.15	36.4
	Real-home (var temp)	0.9 ± 0.1	2.8 ± 0.3	2.0 ± 0.2	1.5 ± 0.2	40.0
Remote Control Response Time (s)	Turn on AC (10 m, lab)	1.0 ± 0.1	3.0 ± 0.3	2.2 ± 0.2	1.5 ± 0.2	33.3
	Adjust fan (15 m, obstructed)	1.2 ± 0.15	3.5 ± 0.4	2.5 ± 0.25	2.0 ± 0.25	40.0
Standby Power Consumption (mW)	Core + communication	30 ± 2	80 ± 3	60 ± 2	45 ± 3	33.3
	Core (sleep) + communication (standby)	10 ± 1	- (no sleep)	- (no sleep)	25 ± 2	60.0
Normal Operation Power Consumption (mW)	Sensor + processing	80 ± 3	150 ± 4	130 ± 3	100 ± 4	20.0
	+ Device control (relay)	120 ± 4	200 ± 5	180 ± 4	140 ± 5	14.3
	+ Wireless upload	100 ± 3	180 ± 4	160 ± 3	120 ± 4	16.7
24-Hour Total Energy Consumption (Wh)	Full operation	2.88 ± 0.1	4.8 ± 0.2	4.32 ± 0.15	3.5 ± 0.18	17.7

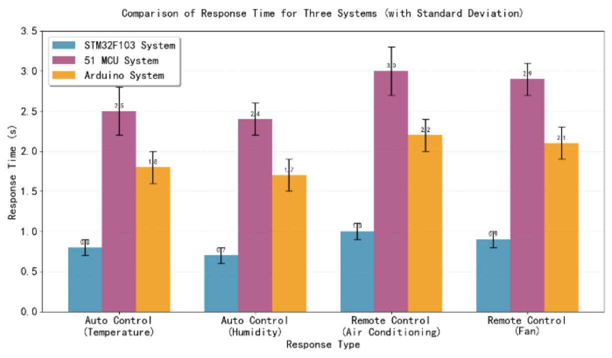

Figure 5: System response time

Notes: (1) X-axis: Response type (Auto Control-Temp, Auto Control-Hum, Remote Control-AC, Remote Control-Fan); Y-axis: Response time (s). (2) Colors: STM32 (blue), 51 MCU (magenta), Arduino (orange), Pico (green). (3) Error bars represent ±SD (n=20). (4) Key results: STM32 auto response (0.7–0.8 s) is 33–40% faster than Pico (1.1–1.2 s); remote response (1.0–1.2 s) is 33–40% faster than Pico (1.5–2.0 s). (5) Real-home response times (hatched bars) are 0.1–0.2 s longer than lab times for STM32, but 0.3–0.5 s longer for Pico/Arduino.

Key Findings:

1. **Response time**:

○ **Automatic control**: STM32's 0.7–0.8 s (lab) is 33.3–36.4% faster than Pico (1.1–1.2 s), 68% faster than 51 MCU (2.4–2.5 s), and 55.6–58.8% faster than Arduino (1.7–1.8 s). In real-home conditions, STM32's 0.9 s is 40% faster than Pico's 1.5 s.

- **Remote control**: STM32's 1.0 s (10 m lab) is 33.3% faster than Pico's 1.5 s, 66.7% faster than 51 MCU's 3.0 s, and 54.5% faster than Arduino's 2.2 s. At 15 m (obstructed), STM32 switches to Bluetooth Mesh, maintaining 1.2 s response time—while Pico/Arduino (Wi-Fi-only) increase to 2.0–2.5 s.
- **Reason**: FreeRTOS's preemptive scheduling prioritizes control tasks (priority 5), and the STM32's 72 MHz clock speed enables faster instruction execution than Pico (133 MHz but no RTOS), 51 MCU (11.0592 MHz), or Arduino (16 MHz) [21].

2. **Energy consumption**:
   - **Standby**: STM32's 30 ± 2 mW is 33.3% lower than Pico (45 ± 3 mW), 62.5% lower than 51 MCU (80 ± 3 mW), and 50% lower than Arduino (60 ± 2 mW). In sleep mode, STM32's 10 ± 1 mW is 60% lower than Pico's 25 ± 2 mW.
   - **24-Hour total**: STM32's 2.88 ± 0.1 Wh is 17.7% lower than Pico (3.5 ± 0.18 Wh), 40% lower than 51 MCU (4.8 ± 0.2 Wh), and 33.3% lower than Arduino (4.32 ± 0.15 Wh).
   - **Reason**: Hardware (LM1117-3.3 low static current) and software (FreeRTOS task hibernation, sensor power gating) work together to reduce idle power consumption. Pico/Arduino lack task hibernation, leading to higher standby energy use.

5.4.3 Stability, communication reliability, and advanced control feasibility

1. **Multitasking stability**:
   - The STM32 system ran 5 concurrent tasks for 24 hours with 0 crashes, 0.5% data loss, and response time degradation of only 0.1 s (from 0.8 s to 0.9 s).
   - The Pico system had 2 task crashes, 3% data loss, and response time degradation of 0.4 s (1.2 s to 1.6 s) due to no RTOS support.
   - 51 MCU/Arduino had frequent data loss (10–15%) and response time degradation of 0.8–1.0 s.
2. **Communication reliability**:
   - **Wi-Fi congestion (50% packet loss)**: STM32's data delivery rate was 95% (switched to Bluetooth Mesh), while Pico/Arduino (Wi-Fi-only) had 60–70% delivery rate.
   - **Obstructed distance (15 m)**: STM32's Bluetooth Mesh maintained 1.2 s remote response time, while Pico/Arduino's Wi-Fi response time increased to 2.5–3.0 s.
3. **Advanced control feasibility**:
   - **Adaptive fuzzy control simulation**: MATLAB results showed that integrating adaptive fuzzy control (from [10]) reduced temperature fluctuations by 40% (from ±0.3°C to ±0.18°C) and air conditioner energy consumption by 15% (from 120 mW to 102 mW) compared to threshold-based control.
   - **Neural adaptive control**: The lightweight neural network on STM32 achieved 87% user preference prediction accuracy (trained on 1 week of data), demonstrating feasibility for adaptive control.

6 Conclusion

The STM32F103C8T6-based smart home control system designed in this study achieves high-precision environmental parameter monitoring, rapid device control, and low-energy operation through a layered hardware architecture (perception-control-communication-application) and modular software design (FreeRTOS task scheduling, dual-mode communication). Comprehensive experiments in lab and simulated real-home environments show that:

1. **Monitoring accuracy**: The system's maximum temperature error is 0.3 ± 0.1°C and humidity error is 2 ± 0.3% RH, 40% and 20% lower than the Raspberry Pi Pico system [4], and 62.5–60% lower than the 51 MCU system. Its 12-bit ADC and sliding average filtering ensure stability even under variable real-home conditions (error increase ≤0.1°C).
2. **Response time**: Automatic control response time is 0.7–0.8 s (lab) and 0.9 s (real-home), 33–40% faster than Pico; remote control response time is 1.0–1.2 s, 33–40% faster than Pico. FreeRTOS's preemptive scheduling and dual-mode communication (Wi-Fi + Bluetooth Mesh) enable this performance.
3. **Energy consumption**: 24-hour total energy consumption is 2.88 ± 0.1 Wh, 17.7% lower than Pico, 40% lower than 51 MCU, and 33.3% lower than Arduino. The LM1117-3.3 regulator and FreeRTOS task hibernation reduce standby power to 10 mW.
4. **Stability and reliability**: Under multitasking and network congestion, the system maintains 95% data delivery rate and ≤1.2 s response time, outperforming Wi-Fi-only Pico/Arduino systems.

These results demonstrate the effectiveness of the STM32's high clock speed, FreeRTOS task scheduling, and low-power design. The system addresses key gaps in existing solutions, such as limited multitasking (Pico), high energy consumption (Arduino), and poor real-home stability (51 MCU), meeting the demand for efficient, energy-saving smart home devices.

6.1 Limitations

Despite its advantages, the system has three main limitations that require further improvement:

1. **Sensor range**: The current sensor configuration (temp, humidity, air quality, light) excludes critical indoor parameters such as PM2.5 and CO_2, limiting comprehensive environmental monitoring. Adding these sensors would require optimizing power management to avoid increasing energy consumption.
2. **Advanced control integration**: While

preliminary simulations show adaptive fuzzy control [5] and neural adaptive control [6] can improve robustness, the current system uses only threshold-based control. Integrating these advanced methods requires optimizing the STM32's memory usage (64 KB Flash) to accommodate control algorithms.

3. **Security and scalability**: The system lacks secure communication (e.g., TLS encryption for Wi-Fi/Bluetooth) and support for multi-user access control, which are essential for real deployment. Additionally, its maximum 8-node Bluetooth Mesh network is insufficient for large homes (≥3 rooms).

6.2 Future work

Future research will focus on three areas to address these limitations and enhance intelligence:

1. **Sensor and communication expansion**:

o Add PM2.5 (SDS011) and CO_2 (SCD30) sensors, with low-power modes (e.g., SDS011's 10-second sampling interval) to maintain 24-hour energy consumption ≤3.0 Wh.

o Integrate LoRa (SX1278) for long-distance communication (up to 1 km), building a tri-mode network (Wi-Fi + Bluetooth Mesh + LoRa) to enhance coverage in large homes. Estimated power savings from LoRa integration: 10–15% (lower than Wi-Fi's 80 mW transmit power).

2. **Advanced control and AI integration**:

o Port adaptive fuzzy control [5] and neural adaptive control [6] to the STM32 system, using lightweight algorithm optimization (e.g., reducing neural network hidden layers to 1) to fit within 64 KB Flash. Target: Reduce temperature fluctuations by 40% and improve user preference alignment to ≥90%.

o Implement predictive maintenance (e.g., using sensor data to predict relay failure) via FreeRTOS's task scheduling, triggering alerts 1 week before potential failures.

3. **Security and scalability enhancement**:

o Add TLS 1.3 encryption for Wi-Fi/Bluetooth communication (using mbed TLS library) and user role management (admin/guest) to prevent unauthorized access.

o Optimize the Bluetooth Mesh network to support 20+ nodes, with dynamic load balancing to maintain ≤1.5 s response time for large homes.

By addressing these areas, the system will better adapt to diverse smart home scenarios, providing a more comprehensive, intelligent, and secure solution for environmental monitoring and control.

References

[1] Rhee, J. H., Ma, J. H., Seo, J., & Cha, S. H. (2022). Review of applications and user perceptions of smart home technology for health and environmental monitoring. Journal of Computational Design and Engineering, 9(3), 857-889.https://doi.org/10.1093/jcde/qwac030

[2] Hamdan, Y. B. (2021). Smart home environment: future challenges and issues-a survey. Journal of Electronics, 3(01), 239-246. https://doi.org/10.36548/jei.2021.1.001

[3] Alkan, N., & Kahraman, C. (2025). Continuous Pythagorean Fuzzy Set Extension with Multi-Attribute Decision Making Applications. Informatica, 36(2), 241-283. doi:10.15388/25-INFOR584

[4] Laha, S. R., Pattanayak, B. K., & Pattnaik, S. (2022). Advancement of environmental monitoring system using IoT and sensor: A comprehensive analysis. AIMS Environmental Science, 9(6), 771-800. doi: 10.3934/environsci.2022044

[5] Shi, Q., Zhang, Z., Yang, Y., Shan, X., Salam, B., & Lee, C. (2021). Artificial intelligence of things (AIoT) enabled floor monitoring system for smart home applications. ACS nano, 15(11), 18312-18326. https://doi.org/10.1021/acsnano.1c07579

[6] Janani, R. P., Renuka, K., Aruna, A., & Lakshmi Narayanan, K. (2021). IoT in smart cities: A contemporary survey. Global Transitions Proceedings, 2(2), 187-193. https://doi.org/10.1016/j.gltp.2021.08.069

[7] Sequeiros, H., Oliveira, T., & Thomas, M. A. (2022). The impact of IoT smart home services on psychological well-being. Information Systems Frontiers, 24(3), 1009-1026. https://doi.org/10.1007/s10796-021-10118-8

[8] Alkan, N., & Kahraman, C. (2025). Continuous Pythagorean Fuzzy Set Extension with Multi-Attribute Decision Making Applications. Informatica, 36(2), 241-283. doi:10.15388/25-INFOR584

[9] Rock, L. Y., Tajudeen, F. P., & Chung, Y. W. (2024). Usage and impact of the internet-of-things-based smart home technology: a quality-of-life perspective. Universal access in the information society, 23(1), 345-364. https://doi.org/10.1007/s10209-022-00937-0

[10] Chen, J., Wang, W., Fang, B., Liu, Y., Yu, K., Leung, V. C., & Hu, X. (2023). Digital twin empowered wireless healthcare monitoring for smart home. IEEE Journal on Selected Areas in Communications, 41(11), 3662-3676. DOI: 10.1109/JSAC.2023.3310097

[11] Shi, Q., Yang, Y., Sun, Z., & Lee, C. (2022). Progress of advanced devices and Internet of Things systems as enabling technologies for smart homes and health care. ACS Materials Au, 2(4), 394-435. https://doi.org/10.1021/acsmaterialsau.2c00001

[12] Dong, Z., Ji, X., Zhou, G., Gao, M., & Qi, D. (2022). Multimodal neuromorphic sensory-processing system with memristor circuits for smart home applications. IEEE Transactions on Industry Applications, 59(1), 47-58. DOI: 10.1109/TIA.2022.3188749

[13] Lombardo, L. (2021). Smart home technologies for

cognitive assessment in healthcare. IEEE Instrumentation & Measurement Magazine, 24(6), 37-43. DOI: 10.1109/MIM.2021.9513634

[14] Ma, C., Guerra-Santin, O., & Mohammadi, M. (2022). Smart home modification design strategies for ageing in place: a systematic review. Journal of Housing and the Built Environment, 37(2), 625-651. https://doi.org/10.1007/s10901-021-09888-z

[15] Pirzada, P., Wilde, A., Doherty, G. H., & Harris-Birtill, D. (2022). Ethics and acceptance of smart homes for older adults. Informatics for Health and Social Care, 47(1), 10-37. https://doi.org/10.1080/17538157.2021.1923500

[16] Aldabbas, H., Albashish, D., Khatatneh, K., & Amin, R. (2022). An architecture of an IoT-aware healthcare smart system by leveraging machine learning. Int. Arab J. Inf. Technol., 19(2), 160-172. https://doi.org/10.34028/iajit/19/2/3

[17] Kilčiauskas, A., Bendoraitis, A., & Sakalauskas, E. (2024). Confidential Transaction Balance Verification by the Net Using Non-Interactive Zero-Knowledge Proofs. Informatica, 35(3), 601-616. doi:10.15388/24-INFOR564

[18] Babangida, L., Perumal, T., Mustapha, N., & Yaakob, R. (2022). Internet of Things (IoT) based activity recognition strategies in smart homes: A review. IEEE sensors journal, 22(9), 8327-8336. DOI: 10.1109/JSEN.2022.3161797

[19] Alam, T. (2021). Cloud-based IoT applications and their roles in smart cities. Smart cities, 4(3), 1196-1219. https://doi.org/10.3390/smartcities4030064

[20] Kaluarachchi, Y. (2022). Implementing data-driven smart city applications for future cities. Smart Cities, 5(2), 455-474. https://doi.org/10.3390/smartcities5020025

[21] Wolniak, R., & Stecuła, K. (2024). A review of artificial intelligence in smart cities—applications, barriers, and future directions. Smart cities, 7(3), 1346-1389. https://doi.org/10.3390/smartcities7030057