UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

DIPARTIMENTO DI SCIENZE TEORICHE E APPLICATE
CORSO DI STUDIO TRIENNALE IN INFORMATICA - F004

Sviluppo di un sistema embedded per il controllo della temperatura in una camera di collaudo

Relatore:
Prof. Carlo Dossi

Tesi di Laurea di:
Mattia Papaccioli - 747053

Tutor Aziendale:
Edoardo Scaglia

Azienda ospitante:
AMEL SRL

Anno Accademico:
2025/2026

Indice

1

Introduzione

1.1 Abstract

La camera di collaudo, installata presso la sede di AMEL s.r.l., rappresenta un'infrastruttura critica dedicata alla verifica e validazione di carichi resistivi. Durante le operazioni di test, i carichi resistivi generano una significativa quantità di energia termica, rendendo indispensabile l'implementazione di un sistema di controllo climatico efficiente, particolarmente durante i periodi estivi caratterizzati da elevate temperature ambientali.

Il presente progetto descrive lo sviluppo e l'implementazione di un sistema embedded progettato per regolare automaticamente la velocità di una ventola di raffreddamento mediante un algoritmo di controllo PID (Proporzionale-Integrale-Derivativo). Il funzionamento del sistema può essere articolato nelle seguenti fasi operative:

Monitoraggio della temperatura: La temperatura ambientale viene acquisita continuamente tramite sensori dedicati, fornendo il dato di ingresso fondamentale per il sistema di controllo.

Elaborazione PID: L'algoritmo PID processa il segnale di temperatura, calcolando la tensione ottimale da fornire all'inverter. Tale calcolo considera sia l'errore istantaneo rispetto al setpoint (azione proporzionale) sia l'accumulo storico degli errori (azione integrativa), garantendo una risposta precisa e stabile.

Controllo attuatore: L'inverter riceve il segnale di controllo e modula la frequenza di rotazione della ventola di raffreddamento, mantenendo costante la temperatura ambiente desiderata attraverso un sistema di retroazione negativa continua.

La comunicazione tra il sistema embedded e l'inverter avviene mediante il protocollo industriale MODBUS RTU, implementato tramite la libreria libmodbus [1], garantendo affidabilità e standardizzazione nello scambio dati.

Il sistema integra inoltre un'interfaccia utente grafica sviluppata con la libreria LVGL [2], accessibile tramite display touchscreen, che permette agli operatori di regolare

manualmente la temperatura target di funzionamento. Il dispositivo è connesso alla rete aziendale tramite interfaccia Ethernet e prevede l'implementazione di un web server per consentire il controllo remoto dei parametri operativi.

La comunicazione inter-processo tra i componenti del sistema è realizzata tramite meccanismi di scrittura su file e IPC (Inter-Process Communication), garantendo un'architettura modulare e scalabile.

È in fase di valutazione l'introduzione di un sistema di database per la registrazione storica dei dati di temperatura, con gestione delegata al web server per l'archiviazione e l'analisi delle tendenze temporali.

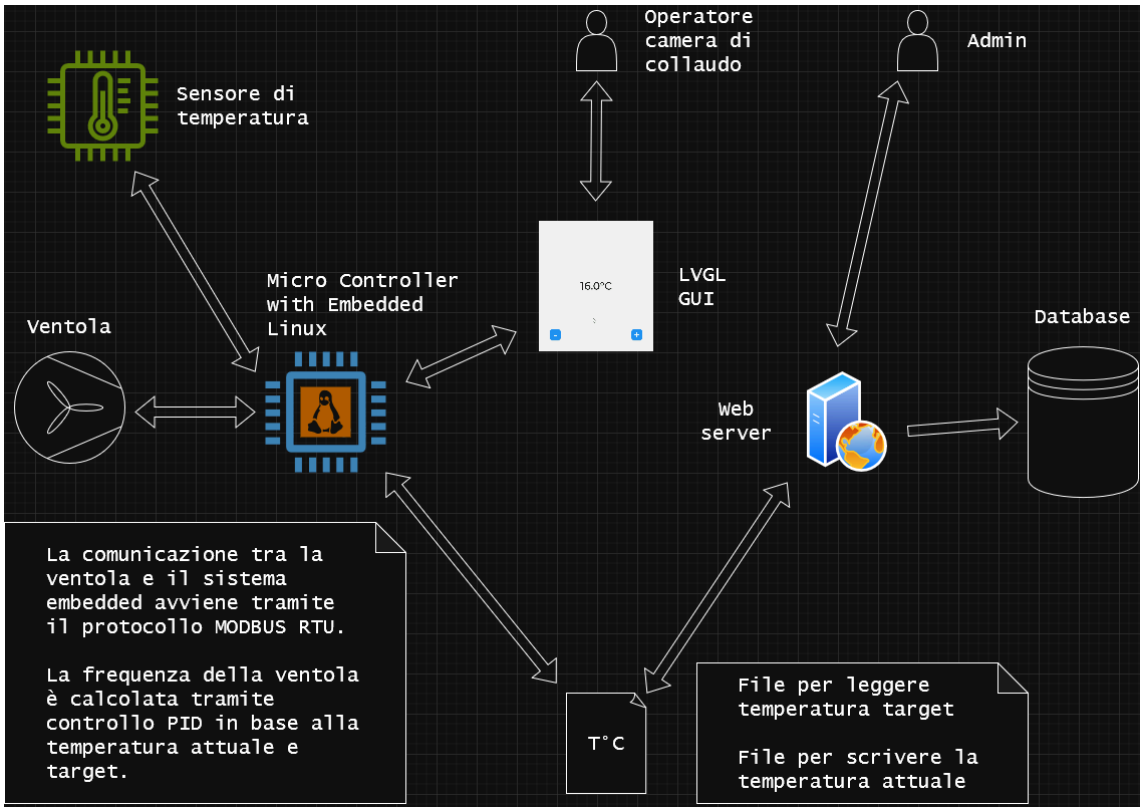

Figura 1 — Architettura generale del sistema di controllo climatico

1.2 Architettura modulare del sistema

Il codice sorgente del progetto è pubblicamente disponibile sulla piattaforma GitHub [3] e presenta un'architettura modulare progettata per massimizzare la manutenibilità, l'estensibilità e la riusabilità del codice. Tale approccio architetturale si discosta significativamente da soluzioni monolitiche, offrendo vantaggi strategici in termini di sviluppo, test e manutenzione.

Il sistema è strutturato nei seguenti moduli principali:

Modulo `common-control`: Costituisce il fondamento condiviso dell'intero sistema, fornendo un insieme di funzioni di utilità, definizioni di configurazione globale e librerie

comuni. Questo modulo facilita la standardizzazione delle comunicazioni tra i differenti componenti del sistema, in particolare tra l'interfaccia grafica e il backend di controllo.

Modulo `temp-control`: Si occupa della gestione completa dell'interfaccia utente grafica, sviluppata mediante il framework LVGL. Questo modulo gestisce l'interazione con l'operatore, la visualizzazione dei dati sensoriali e l'impostazione dei parametri operativi, garantendo un'esperienza utente intuitiva e responsiva.

Modulo `pid-control`: Implementa il nucleo logico del sistema di regolazione, includendo l'algoritmo PID per il controllo della temperatura, l'acquisizione dei dati sensoriali e la comunicazione con l'inverter tramite protocollo MODBUS RTU [1].

La scelta di un'architettura modulare offre molteplici vantaggi: in primo luogo, consente l'isolamento delle responsabilità, facilitando il debugging e la manutenzione; in secondo luogo, permette la sostituzione o l'aggiornamento di singoli moduli senza impattare sull'intero sistema; infine, supporta lo sviluppo parallelo da parte di team diversi, accelerando i cicli di sviluppo e testing.

2

Architettura dei Moduli Software

2.1 Modulo `common-control`

Il modulo `common-control` rappresenta il componente fondamentale dell'architettura software, fornendo l'infrastruttura condivisa necessaria per il coordinamento degli altri moduli. Questo modulo contiene un insieme di funzioni di utilità, header di configurazione globale e script di sistema essenziali per l'inizializzazione e l'avvio coordinato dell'intero sistema.

L'architettura prevede l'utilizzo della directory di sistema `/opt/amel/` come repository centrale per tutti gli script, i file di configurazione e i dati condivisi tra i differenti moduli, garantendo una gestione centralizzata e standardizzata delle risorse di sistema.

2.1.1 Sistema di Comunicazione Inter-modulo

Per garantire un efficace coordinamento tra i moduli `temp-control` e `pid-control`, è stata implementata un'architettura di comunicazione basata su meccanismi di segnalazione e scrittura su file. Questo approccio combina la semplicità dello scambio dati tramite file system con l'efficienza della comunicazione asincrona tramite segnali di sistema.

Il flusso comunicativo segue un schema ben definito: quando un operatore modifica la temperatura target attraverso l'interfaccia touchscreen, il nuovo valore viene immediatamente persistito nel file `/opt/amel/target-temperature`, rendendolo disponibile per il processo di controllo.

In modo simmetrico, il processo PID, periodicamente acuisce i dati dai sensori di temperatura DS18B20 [4] e scrive i valori rilevati nei file `/opt/amel/current-temperature/sX`, dove X rappresenta l'identificativo univoco del sensore sul bus 1-Wire.

Immediatamente dopo l'aggiornamento dei file, il processo `pid-control` invia un segnale al modulo `temp-control`, il quale provvede a leggere i dati aggiornati e a refreshare la visualizzazione delle temperature sensoriali sull'interfaccia grafica.

2.1.2 Libreria di Logging `logging.h`

Per garantire un monitoraggio efficace e un'analisi approfondita del comportamento del sistema, è stata implementata una libreria di logging in C che offre funzionalità avanzate di registrazione eventi. La libreria supporta una gerarchia completa di livelli di logging, inclusi `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR` e `FATAL`, permettendo una granularità fine nella selezione dei messaggi da registrare.

Il sistema offre flessibilità configurabile nella gestione dell'output, consentendo di scegliere tra scrittura su console, registrazione nel syslog di sistema, o entrambe. È possibile definire dinamicamente il livello minimo di filtro, ottimizzando il volume di informazioni registrate in base alle esigenze operative.

La libreria implementa meccanismi di sincronizzazione tramite mutex per prevenire race condition e garantire la consistenza dei log in ambienti multi-threading. È stata inoltre introdotta un'opzione configurabile per la precisione temporale dei timestamp, particolarmente utile durante le fasi di debug per analizzare la regolarità e la precisione del ciclo di controllo PID.

2.1.3 Sistema di Templating Bash tramite Preprocessore C

Per eliminare ridondanze e garantire la consistenza configurativa attraverso l'intero sistema, è stato implementato un approccio innovativo che sfrutta il preprocessore del linguaggio C per la generazione di script Bash. Questa soluzione consente di centralizzare tutte le definizioni di configurazione nell'header `include/config.h`, utilizzandolo come sorgente unica di verità per la generazione degli script di sistema.

Il processo di generazione avviene mediante l'utilizzo di GCC con la flag `-E`, che attiva esclusivamente la fase di preprocessing, evitando le successive fasi di compilazione, assemblaggio e linking. Gli script risultanti, `init.sh` e `run.sh`, vengono generati automaticamente a partire dai template che incorporano le costanti definite nell'header di configurazione.

Questo approccio offre significativi vantaggi in termini di manutenibilità: qualsiasi modifica ai parametri di configurazione richiede l'intervento su un unico file, con propagazione automatica delle modifiche a tutti gli script derivati, eliminando così potenziali errori di inconsistenza e riducendo significativamente lo sforzo di manutenzione.

2.1.4 Sequenza di Avvio del Sistema

La procedura di avvio del sistema è strutturata in due fasi distinte per garantire una corretta inizializzazione e un avvio ordinato dei componenti. Durante la prima esecuzione del dispositivo embedded, è necessario eseguire lo script `/opt/amel/init.sh`, che provvede a creare le directory di sistema necessarie per il funzionamento dei sensori di temperatura e a configurare i meccanismi di controllo degli accessi.

Per l'avvio operativo del sistema, lo script `/opt/amel/run.sh` coordina il lancio sequenziale dei componenti principali. La sequenza di avvio segue un ordine preciso:

1. **Inizializzazione dell'interfaccia grafica**: Viene avviato prima il

processo principale di `temp-control`, che si pone in stato di attesa dei segnali provenienti dal modulo di controllo PID.

2. **Avvio del controllore PID**: Successivamente viene lanciato il processo

`pid-control`, che esegue l'inizializzazione del bus 1-Wire, rileva il numero di sensori disponibili e persiste questa informazione nel file `/opt/amel/number-sensors`.

3. **Sincronizzazione dei moduli**: Una volta completata l'inizializzazione,

il processo `pid-control` invia un segnale di notifica a `temp-control`, avviando così la fase operativa congiunta.

Durante il funzionamento, i due processi mantengono una comunicazione costante attraverso segnali di sistema generati tramite il comando `kill`, permettendo l'aggiornamento in tempo reale sia dei setpoint di temperatura target che dei valori effettivi misurati.

2.2 Modulo `temp-control`

Il modulo `temp-control` costituisce l'interfaccia primaria di interazione tra l'operatore e il sistema di controllo climatico. Attraverso un display touchscreen LCD dedicato, l'utente può monitorare lo stato del sistema e regolare dinamicamente la temperatura target desiderata per la camera di collaudo.

L'interfaccia grafica è stata sviluppata utilizzando la libreria LVGL [2], un framework versatile e ottimizzato per sistemi embedded. Per l'integrazione con l'ecosistema Linux, è stato utilizzato il template [5], che fornisce il porting ufficiale della libreria per ambienti Linux, garantendo compatibilità e prestazioni ottimali sulla piattaforma target.

2.2.1 Interfaccia Utente Principale

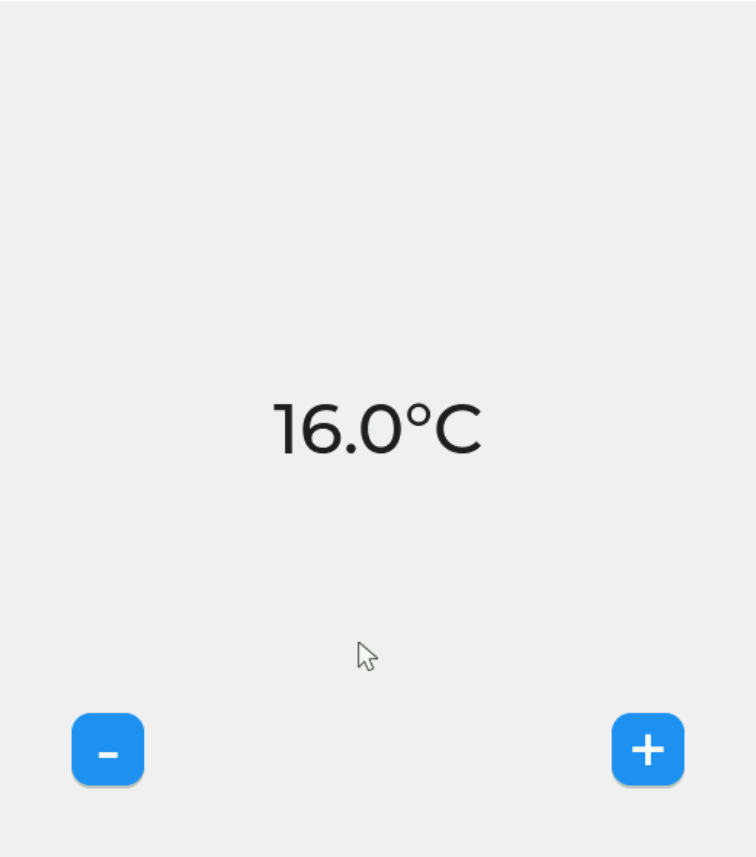

Figura 2 — Interfaccia grafica utente per il controllo della temperatura ambientale

L'interfaccia utente principale presenta un design pulito e funzionale, ottimizzato per l'operatività in ambienti industriali. L'layout è organizzato per garantire un'immediata comprensione dello stato del sistema e un controllo intuitivo dei parametri operativi:

- **Visualizzazione del setpoint**: Viene mostrato in modo prominente il valore della temperatura target attualmente configurata.
- **Monitoraggio sensoriale**: Vengono visualizzati in tempo reale i valori di temperatura rilevati da tutti i sensori collegati sul bus 1-Wire, permettendo un monitoraggio completo della distribuzione termica all'interno della camera di collaudo.
- **Controlli di regolazione**: Due pulsanti dedicati permettono di incrementare o decrementare la temperatura target con passo unitario, fornendo un controllo preciso e immediato del setpoint desiderato.

```c
void set_target_temperature(float t)
{
    LOG_DEBUG("debug callback -> %.1f\n", target_temperature);
    target_temperature += t;
    lv_label_set_text_fmt(target_temperature_label,
    target_temperature_format, target_temperature);
    write_float_to_file(TARGET_TEMPERATURE_FILE, target_temperature);
    kill(PID_control_PID, SIGUSR1);
}

static void increment_temperature(lv_event_t - e)
{
    if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
        return;
    }
    set_target_temperature(1);
}

static void decrement_temperature(lv_event_t - e)
{
    if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
        return;
    }
    set_target_temperature(-1);
}
```

Codice 1 — Funzioni di callback per i bottoni di incremento e decremento della temperatura target

Quando l'operatore interagisce con i pulsanti di regolazione, il sistema attiva una catena di operazioni coordinate che garantiscono l'immediatezza della risposta e la consistenza dei dati. La pressione di un pulsante determina l'esecuzione di una funzione di callback che provvede simultaneamente a:

1. Aggiornare l'etichetta visuale sul display per riflettere immediatamente

la modifica del setpoint;

2. Persistere il nuovo valore nel file di sistema dedicato, garantendo la

comunicazione al modulo PID;

3. Notificare il processo di controllo tramite segnale per attivare

l'adeguamento del sistema di raffreddamento.

Questo approccio garantisce una 用户体验 fluida e un controllo reattivo del sistema.

2.2.2 Sistema di Input/Output

L'infrastruttura I/O dell'interfaccia LVGL si basa su due componenti fondamentali, selezionati per la loro efficienza e compatibilità con ambienti embedded a risorse limitate:

- **Libevdev [6]**: Questa libreria specializzata gestisce l'acquisizione e

l'elaborazione degli eventi di input provenienti dal touchscreen. Il suo ruolo consiste nel ricevere i segnali tattili, convertirli in eventi standardizzati e trasmetterli al motore grafico LVGL per l'elaborazione.

- **Framebuffer device**: Il dispositivo `/dev/fb0` rappresenta l'interfaccia

hardware di output grafico. La GUI scrive direttamente su questo file system device, il quale contiene la mappa dei pixel dello schermo, permettendo un rendering diretto e ottimizzato dell'interfaccia visuale senza intermediari software aggiuntivi.

2.2.3 Processo di Cross-compilazione

La compilazione dell'applicazione GUI per la piattaforma embedded richiede l'utilizzo di una toolchain specializzata per l'architettura ARM. Il sistema si avvale della toolchain fornita da Buildroot, che include compilatori, linker e librerie ottimizzate per il target specifico.

Configurazione del Toolchain

```
set(CMAKE_SYSTEM_NAME Linux)
set(CMAKE_SYSTEM_PROCESSOR arm)

set(tools ~/buildroot/output/host/bin/arm-buildroot-linux-gnueabihf-)
set(CMAKE_C_COMPILER ${tools}gcc)
set(CMAKE_CXX_COMPILER ${tools}g++)

set(EVDEV_INCLUDE_DIRS ~/buildroot/output/staging/usr/include/libevdev/)
set(EVDEV_LIBRARIES ~/buildroot/output/staging/usr/lib/libevdev.so)

set(BUILD_SHARED_LIBS ON)
```

Codice 2 — cross_compile_setup.cmake - Configurazione del sistema di cross-compilazione

Il processo di generazione del sistema di build viene avviato tramite il comando:

```
cmake -DCMAKE_TOOLCHAIN_FILE=./cross_compile_setup.cmake -B build -S .
```

Questo comando istruisce CMake a generare i Makefile necessari per la cross-compilazione, specificando la toolchain ARM e le dipendenze richieste. La successiva esecuzione di `make -C build -j` [7] avvia il processo di compilazione parallelo.

L'architettura del build prevede la compilazione di LVGL come libreria condivisa dinamica, mentre l'applicazione principale viene generata come eseguibile standalone. Questo approccio ottimizza l'utilizzo della memoria e facilita gli aggiornamenti futuri dei componenti.

2.2.4 Strategia di Branching

Per ottimizzare il flusso di sviluppo e garantire la corretta separazione tra ambiente di sviluppo e dispositivo target, è stata implementata una strategia di branching basata su due repository parallele. Questa architettura permette di mantenere distinti i profili di configurazione ottimizzati per i differenti contesti operativi.

Le due repository sono strutturalmente identiche, ad eccezione del file di configurazione `lv_conf.h`, che contiene i parametri specifici per ciascun ambiente:

- **Branch di sviluppo**: Configurata con backend X11 per l'esecuzione su

workstation di sviluppo. Include controlli di validazione (sanity checks) che facilitano il debug e la prevenzione di errori durante la fase di sviluppo, ma che introducono un overhead di performance non accettabile in produzione.

- **Branch target**: Ottimizzata per il dispositivo embedded, con sanity

checks disabilitati per massimizzare le prestazioni. Utilizza come backend il framebuffer device `/dev/fb0` per il rendering diretto sull'hardware target.

Per garantire l'integrità delle configurazioni specifiche, è stato implementato un meccanismo di protezione del file `lv_conf.h` tramite il file `.gitattributes` con la direttiva `lv_conf.h merge=ours`. Questa configurazione speciale istruisce Git a mantenere la versione del file presente nella branch di destinazione durante le operazioni di merge, prevenendo sovrascritture accidentali e garantendo la persistenza delle configurazioni specifiche per ogni ambiente.

2.3 Modulo `pid-control`

2.3.1 Gestione dei Sensori di Temperatura

Il sistema di monitoraggio termico si basa su due sensori di temperatura digitali DS18B20 [4], collegati in configurazione parallela su un singolo bus di comunicazione 1-Wire. Questa architettura permette di massimizzare l'efficienza cabling semplificando significativamente l'infrastruttura hardware.

Il microcontrollore opera come master del bus 1-Wire, avviando periodicamente sequenze di interrogazione per acquisire i dati di temperatura da ciascun sensore. Il processo principale del controllore PID esegue ciclicamente le operazioni di lettura dei sensori e calcola l'output di controllo basandosi sulla differenza tra la temperatura misurata e il setpoint desiderato.

La procedura di inizializzazione dei sensori segue un approccio ottimizzato per garantire robustezza ed efficienza:

1. **Enumerazione dei sensori**: Il sistema prima determina il numero esatto

di sensori presenti sul bus, allocando dinamicamente la memoria necessaria per memorizzare gli identificativi univoci (UUID) di ciascun dispositivo.

2. **Acquisizione degli identificativi**: Successivamente, il sistema legge e

memorizza gli UUID di tutti i sensori rilevati, creando una mappa persistente del bus.

Questa strategia, sebbene richieda un consumo di memoria leggermente superiore, offre significativi vantaggi in termini di efficienza operativa. L'alternativa di rilevare gli identificativi dei sensori ad ogni ciclo di lettura comporterebbe chiamate ripetute alla funzione `DS18X20_find_sensor`, generando un inutile consumo di cicli CPU e introducendo latenza nel processo di acquisizione dati. L'approccio adottato esegue la procedura di discovery una sola volta all'avvio, con un impatto trascurabile sulle performance globali del sistema.

```c
typedef struct
{
    char id[16];
    int16_t temperature;
    uint8_t uint_id[OW_ROMCODE_SIZE];
} sensor;

void init_onewire_sensors()
{
    uint8_t diff = OW_SEARCH_FIRST;

    while (diff != OW_LAST_DEVICE)
    {
        sensors_count++;
        DS18X20_find_sensor(&diff, id);
    }

    write_char_to_file(NUMBER_OF_SENSORS_FILE, sensors_count);
    LOG_INFO("Found %d sensors on the 1-wire bus", sensors_count);

    sensors = malloc(sizeof(sensor) - sensors_count);

    diff = OW_SEARCH_FIRST;
    int i = 0;
    while (diff != OW_LAST_DEVICE)
    {
        DS18X20_find_sensor(&diff, id);
        sensor s;
        for (int i = 0; i < OW_ROMCODE_SIZE; i++)
        {
            s.uint_id[i] = id[i];
        }
        sprintf(s.id, "%02hx%02hx%02hx%02hx%02hx%02hx%02hx%02hx",
        id[0], id[1],
              id[2], id[3], id[4], id[5], id[6], id[7]);
        s.temperature = 0;
        sprintf(s.file, CURRENT_TEMPERATURE_FILE "/s%d", i);
        LOG_INFO("sensor %d id %s", i, s.id);
        sensors[i] = s;
        i++;
    }
}
```

Codice 3 — `pid/src/main.c`

2.3.2 Comunicazione MODBUS RTU

Per l'interfacciamento con l'inverter che governa il funzionamento della ventola di raffreddamento, è stata implementata una soluzione basata sul protocollo di comunicazione industriale MODBUS RTU. Questo standard, ampiamente diffuso negli ambienti automazione industriale, garantisce affidabilità, interoperabilità e robustezza nella comunicazione seriale.

L'implementazione si avvale della libreria `libmodbus` [1], che fornisce un'interfaccia C standardizzata per la gestione delle comunicazioni MODBUS, semplificando significativamente lo sviluppo e garantendo la conformità con le specifiche del protocollo.

2.3.3 Algoritmo di Controllo PID

Il sistema di controllo PID (Proporzionale-Integrale-Derivativo) rappresenta il cuore logico del sistema di regolazione termica. Si tratta di un controller a retroazione negativa che elabora continuamente l'errore tra il valore misurato e il setpoint desiderato per generare un segnale di controllo ottimizzato.

Nel contesto specifico della camera di collaudo, l'algoritmo PID opera mantenendo stabile la temperatura ambiente. Il controller riceve in input due parametri fondamentali: la temperatura effettivamente rilevata dai sensori e il setpoint target impostato dall'operatore. Sulla base di questi dati, calcola il valore di tensione da trasmettere all'inverter, che a sua volta modula la frequenza di rotazione della ventola di raffreddamento.

La sintonizzazione dei parametri PID è stata eseguita mediante metodologie sperimentali che combinano l'analisi della risposta del sistema con criteri di stabilità e prestazione. Il coefficiente proporzionale (P) è stato calibrato per garantire una risposta rapida agli errori di temperatura, mentre il termine integrale (I) è stato ottimizzato per eliminare l'errore stazionario e garantire il raggiungimento del setpoint in condizioni steady-state.

Il termine derivativo (D) è stato volutamente escluso dalla configurazione finale. Questa decisione si basa sull'analisi delle caratteristiche dinamiche del sistema termico, che presenta una costante di tempo relativamente elevata e una risposta intrinsecamente lenta. In tali condizioni, l'azione derivativa introdurrebbe principalmente rumore di misurazione amplificato senza fornire contributi significativi alla qualità del controllo, oltre a potenzialmente causare instabilità nel comportamento del regolatore.

Gestione del Timing Basata su Monotonic Clock

La precisione temporale nel campionamento della temperatura riveste un ruolo critico per il corretto funzionamento dell'algoritmo PID. Per garantire una frequenza di campionamento costante e immune a variazioni sistemiche, è stata implementata una soluzione basata sull'interfaccia `<sys/timerfd.h>` della Standard C Library.

Questo approccio utilizza il sistema di monotonic clock del kernel Linux, che fornisce una misura del tempo immune a modifiche manuali dell'orologio di sistema. Le chiamate di sistema timerfd creano e gestiscono un timer ad alta precisione che genera eventi di scadenza a intervalli regolari, comunicati tramite un file descriptor.

Questo metodo offre significativi vantaggi rispetto alle soluzioni tradizionali basate su sleep o busy-waiting:

- Precisione temporale consistente e riproducibile
- Minimo overhead CPU in attesa degli eventi temporali

- Immunità a variazioni dell'orologio di sistema
- Integrazione nativa con i meccanismi di I/O multiplexing del kernel

Ottimizzazione della Priorità di Scheduling

Per massimizzare la determinismo del sistema di controllo e garantire la massima regolarità possibile nel ciclo di campionamento, è stato implementato un meccanismo di ottimizzazione della priorità di scheduling. Mediante l'utilizzo dell'header `<sched.h>`, al processo PID viene assegnata la priorità più elevata disponibile nel sistema di scheduling real-time.

Questa configurazione garantisce che il processo di controllo abbia precedenza sulla maggior parte delle altre attività di sistema, riducendo significativamente la probabilità di preemption da parte di altri processi e minimizzando la latenza nell'esecuzione del ciclo di controllo.

I benefici di questo approccio includono:
- Riduzione del jitter temporale nelle operazioni di campionamento
- Migliore prevedibilità dei tempi di risposta del controller
- Minore impatto delle attività di sistema sull'algoritmo PID
- Maggiore stabilità complessiva del sistema di controllo

Questa ottimizzazione è particolarmente critica considerando che il sistema opera su una piattaforma embedded con risorse limitate e dove la regolarità temporale del ciclo di controllo influenza direttamente la qualità della regolazione termica.

2.3.4 Sistema di Amministrazione e Monitoraggio

Il modulo `pid-control` integra funzionalità avanzate di amministrazione e monitoraggio che consentono una gestione completa del sistema operativo. Queste capacità permettono non solo di supervisionare il funzionamento corrente, ma anche di diagnosticare eventuali anomalie e ottimizzare le prestazioni nel tempo.

Il sistema di logging, integrato con il modulo `common-control`, fornisce una registrazione dettagliata di tutti gli eventi significativi, includendo letture sensoriali, calcoli di controllo, comunicazioni con l'inverter e stati di errore. Questi dati sono essenziali per l'analisi post-mortem e per l'identificazione di trend di comportamento a lungo termine.

Sono inoltre implementati meccanismi di monitoraggio real-time che permettono di verificare la salute del sistema, controllare la regolarità dei cicli di campionamento e validare la correttezza delle comunicazioni MODBUS. Queste funzionalità contribuiscono a garantire l'affidabilità e la robustezza del sistema di controllo climatico.

3

Sistema Embedded Basato su Linux

3.1 Architettura Hardware

La piattaforma hardware impiegata per il sistema embedded è stata sviluppata internamente da AMEL e si caratterizza per un'architettura modulare composta da due schede principali che integrano tutte le funzionalità necessarie per l'applicazione specifica.

3.1.1 Host-board Ganador (rev. 4)

La scheda host-board rappresenta l'infrastruttura di connettività e interfacciamento del sistema, integrando:

- **Memoria di massa**: Scheda SD utilizzata come supporto di archiviazione principale per il sistema operativo, le applicazioni e i dati di configurazione.
- **Connettività di rete**: Interfaccia Ethernet 10/100 Mbps per il collegamento alla rete aziendale e la comunicazione remota.
- **Interfaccia seriale**: Porta RS-232 per la comunicazione di sistema, debugging e configurazione in fase di sviluppo e manutenzione.
- **Display interattivo**: Schermo touchscreen per l'interfaccia utente locale e il controllo diretto del sistema da parte degli operatori.

3.1.2 System-on-Module Vulcano-A5

Il SoM Vulcano-A5 costituisce il cuore computazionale del sistema, integrando in un formato compatto tutte le risorse di elaborazione necessarie:

- **Processore centrale**: CPU Atmel ARM9 AT91SAM9X35 operante a frequenza di 400MHz, ottimizzata per applicazioni embedded a basso consumo.
- **Sistema di memoria**: 128 MB di memoria DDR2 SDRAM per l'esecuzione delle applicazioni e 256 MB di memoria flash NAND per la persistenza dei dati e del firmware.
- **Interfacce di espansione**: Connettore SODIMM200 per l'espansione delle funzionalità e l'integrazione con periferiche aggiuntive.
- **Controller di rete**: Controller Ethernet MAC integrato per la gestione della comunicazione di rete nativa.
- **Interfacce USB**: Due porte USB 2.0 Host e una porta USB 2.0 Host/Device per la connessione di periferiche esterne.
- **Comunicazione seriale**: Tre porte USART per la comunicazione con dispositivi seriali e sensori.
- **Controller grafico**: Controller TFT LCD con supporto per interfacce TTL e LVDS per la gestione del display a colori.

3.2 Costruzione del Sistema Operativo

Per la gestione e l'orchestrazione delle risorse hardware è stato impiegato il sistema operativo Linux [8], configurato specificamente per le esigenze dell'applicazione embedded. Data la natura limitata delle risorse hardware disponibili, è stato realizzato un kernel personalizzato contenente esclusivamente i moduli essenziali per il funzionamento del sistema, ottimizzando così le prestazioni e riducendo l'impatto sulla memoria.

L'intero processo di costruzione del sistema è stato orchestrato mediante Buildroot [9], un framework specializzato per lo sviluppo di sistemi embedded Linux. Buildroot fornisce un'infrastruttura completa basata su Makefile che automatizza tutte le fasi critiche del processo:

- **Cross-compilazione**: Compilazione incrociata di tutte le librerie e applicazioni per l'architettura ARM target
- **Gestione dipendenze**: Risoluzione automatica delle dipendenze tra pacchetti
- **Creazione del filesystem**: Generazione di un filesystem radice completo e ottimizzato
- **Preparazione immagine**: Assemblaggio di un'immagine disco pronta per

essere scritta sulla scheda SD

Per integrare l'interfaccia grafica sviluppata con LVGL nell'ecosistema Buildroot, è stato necessario creare un pacchetto personalizzato che gestisca la compilazione e l'installazione della libreria grafica e delle sue dipendenze.

3.2.1 Sequenza di Avvio del Sistema

Il processo di boot segue una sequenza strutturata che garantisce un avvio ordinato e affidabile del sistema:

1. **Bootloader primario**: All'accensione del sistema, il bootloader primario

AT91bootstrap [10], residente nella memoria non volatile del SoC, viene eseguito per primo, eseguendo le inizializzazioni hardware fondamentali.

2. **Bootloader secondario**: AT91bootstrap a sua volta lancia Barebox [11],

un bootloader secondario flessibile che fornisce funzionalità avanzate di configurazione e debugging, oltre a gestire il caricamento del kernel Linux in memoria.

3. **Avvio del kernel**: Barebox trasferisce il controllo al kernel Linux,

che prosegue con l'inizializzazione dei driver, il montaggio del filesystem e l'avvio dei processi di sistema e delle applicazioni utente.

3.3 Personalizzazione di Buildroot

L'estensibilità di Buildroot permette di integrare pacchetti personalizzati nell'ecosistema di build mediante una procedura standardizzata. Per aggiungere un nuovo pacchetto al sistema è necessario creare una directory dedicata all'interno della cartella `package`, contenente due file fondamentali:

- **`Config.in`**: File di configurazione che definisce le opzioni compilative

del pacchetto, le dipendenze necessarie e le variabili di configurazione che l'utente può modificare tramite l'interfaccia di configurazione di Buildroot.

- **`<nomepacchetto>.mk`**: Makefile che contiene le istruzioni specifiche

per il download, la compilazione, l'installazione e la pulizia del pacchetto.

Questi due componenti collaborano per fornire a Buildroot tutte le informazioni necessarie per gestire il pacchetto nel ciclo di vita completo: risoluzione delle dipendenze, download dei sorgenti, cross-compilazione per l'architettura target e installazione nel filesystem finale del dispositivo embedded.

3.3.1 Pacchetto `amel-common-control`

Il pacchetto `amel-common-control` costituisce il fondamento condiviso del sistema, fornendo le librerie comuni, gli header di configurazione e gli script di sistema necessari al coordinamento dei diversi moduli applicativi.

```
 AMEL_COMMON_CONTROL_VERSION = v0.9
 AMEL_COMMON_CONTROL_SITE =
 git@git.amelchem.com:mpapaccioli/common-control.git
 AMEL_COMMON_CONTROL_SITE_METHOD = git

 define AMEL_COMMON_CONTROL_BUILD_CMDS
$(TARGET_MAKE_ENV) TOOLS=$(TARGET_CROSS) STAGE=$(STAGING_DIR) \
cmake -S $(@D) -B $(@D)/build -D CMAKE_BUILD_TYPE=Release -D  \
CMAKE_INSTALL_PREFIX=/usr             \
-DCMAKE_TOOLCHAIN_FILE=$(@D)/user_cross_compile_setup.cmake   \
-DBUILD_SHARED_LIBS=ON
$(TARGET_MAKE_ENV) cmake --build $(@D)/build
make -C $(@D)/source/init.sh source/run.sh
 endef

 define AMEL_COMMON_CONTROL_INSTALL_TARGET_CMDS
$(TARGET_MAKE_ENV) DESTDIR=$(TARGET_DIR) cmake --install $(@D)/build
mkdir -p $(STAGING_DIR)/usr/include/common-control
mkdir -p $(STAGING_DIR)/usr/lib

cp -r $(@D)/build/libcommon-control.so- $(STAGING_DIR)/usr/lib/
cp -r $(@D)/include/common-control.h
$(STAGING_DIR)/usr/include/common-control
cp -r $(@D)/include/logging.h $(STAGING_DIR)/usr/include/common-control

$(INSTALL) -d $(TARGET_DIR)/opt/amel
$(INSTALL) -m 0755 $(@D)/source/init.sh $(TARGET_DIR)/opt/amel/init.sh
$(INSTALL) -m 0755 $(@D)/source/run.sh $(TARGET_DIR)/opt/amel/run.sh
 endef

 $(eval $(generic-package))
```

Codice 4 — File di configurazione per il pacchetto amel-common-control.mk

Il processo di build prevede la configurazione tramite CMake con toolchain di cross-compilazione, la compilazione della libreria condivisa e la generazione degli script di sistema. La fase di installazione copia sia i file binari e header nell'ambiente di staging per lo sviluppo, sia gli eseguibili e script nella directory target del dispositivo finale.

3.3.2 Pacchetto `amel-temp-control`

Il pacchetto `amel-temp-control` gestisce l'interfaccia grafica utente sviluppata con LVGL, fornendo il sistema di interazione tra l'operatore e il controllo della temperatura.

```
AMEL_TEMP_CONTROL_VERSION = 44c17c6f2c492f1f3c7d8a6767df390c8d13eb9c
AMEL_TEMP_CONTROL_SITE = git@git.amelchem.com:mpapaccioli/temp-control.git
AMEL_TEMP_CONTROL_SITE_METHOD = git

AMEL_TEMP_CONTROL_DEPENDENCIES = libevdev
AMEL_TEMP_CONTROL_GIT_SUBMODULES = YES

define AMEL_TEMP_CONTROL_BUILD_CMDS
cmake -DCMAKE_TOOLCHAIN_FILE=$(@D)/user_cross_compile_setup.cmake \
  -B $(@D)/build -S $(@D)
make -C $(@D)/build -j @cmake

endef

define AMEL_TEMP_CONTROL_INSTALL_TARGET_CMDS
$(INSTALL) -d $(TARGET_DIR)/opt/amel-temp-control/
cp $(@D)/build/bin/lvglsim $(TARGET_DIR)/opt/amel-temp-control/main
cp -r $(@D)/build/lvgl/lib/- $(TARGET_DIR)/usr/lib

endef

$(eval $(generic-package))
```

Codice 5 — File di configurazione per il pacchetto amel-temp-control.mk

Il processo di costruzione del pacchetto segue una sequenza precisa:

1. **Acquisizione sorgenti**: Viene clonata la repository contenente il codice sorgente dell'interfaccia grafica al commit specificato, con inizializzazione automatica dei sottomoduli Git per includere le dipendenze di LVGL.
2. **Gestione dipendenze**: Buildroot verifica automaticamente la presenza della dipendenza `libevdev`, necessaria per la gestione degli eventi di input dal touchscreen.
3. **Cross-compilazione**: Vengono compilate sia la libreria LVGL che l'applicazione principale utilizzando la toolchain ARM fornita da Buildroot, garantendo l'ottimizzazione per l'architettura target.
4. **Installazione target**: Gli eseguibili compilati e le librerie necessarie vengono installati nelle directory appropriate del filesystem target, rendendo l'applicazione pronta per l'esecuzione sul dispositivo embedded.

3.3.3 Pacchetto `amel-pid`

Il pacchetto `amel-pid` gestisce la compilazione e l'installazione del modulo di controllo PID, che rappresenta il cuore logico del sistema di regolazione termica.

```
    AMEL_PID_VERSION = v0.0.3
    AMEL_PID_SITE = git@git.amelchem.com:mpapaccioli/pid.git
    AMEL_PID_SITE_METHOD = git

    AMEL_PID_DEPENDENCIES = libmodbus

    define AMEL_PID_BUILD_CMDS
make -C $(@D) CC=$(TARGET_CC)
    endef

    define AMEL_PID_INSTALL_TARGET_CMDS
$(INSTALL) -d $(TARGET_DIR)/opt/amel-pid/

cp $(@D)/pid $(TARGET_DIR)/opt/amel-pid/pid

    endef

    $(eval $(generic-package))
```

Codice 6 — File di configurazione per il pacchetto amel-pid-control.mk

Il pacchetto si basa su un processo di build più semplice rispetto agli altri componenti, utilizzando direttamente il compilatore target fornito da Buildroot. Le fasi principali includono:

1. **Download sorgenti**: Acquisizione del codice sorgente dalla repository

interna di AMEL.

2. **Verifica dipendenze**: Controllo automatico della disponibilità della

libreria `libmodbus` per la comunicazione MODBUS.

3. **Compilazione nativa**: Utilizzo del toolchain di Buildroot per la

cross-compilazione dell'eseguibile PID.

4. **Installazione sistema**: Copia dell'eseguibile compilato nella directory

dedicata sul filesystem target.

Questo approccio consente di integrare il modulo di controllo PID nell'ecosistema Buildroot, garantendo la corretta gestione delle dipendenze e l'integrazione con il resto del sistema software.

3.3.4 Configurazione della Rete e Accesso Remoto

Per garantire la piena gestibilità del sistema embedded, è stata implementata un'infrastruttura di rete completa che include connettività remota e accesso a risorse esterne.

La configurazione di rete prevede:

- **Interfaccia virtuale**: Creazione di un'interfaccia di rete virtuale `eth0`

configurata con indirizzo IP statico, garantendo un punto di accesso stabile e prevedibile al dispositivo.

- **Accesso remoto sicuro**: Installazione e configurazione del server SSH

[12] per consentire la connessione remota sicura al dispositivo. La configurazione permette il login come utente `root` tramite autenticazione password, semplificando le operazioni di amministrazione e manutenzione.

Le personalizzazioni necessarie sono state implementate mediante la modifica degli script di sistema:
- Il file `/etc/init.d/S50network` è stato modificato per configurare

automaticamente l'interfaccia di rete con l'indirizzo IP statico durante l'avvio.
- Il file `/etc/ssh/sshd_config` è stato adattato per permettere l'accesso

root tramite password.

3.3.5 Connettività Internet e Sincronizzazione Temporale

Per garantire l'accesso a risorse di rete esterne, è stata configurata una rete bridge che permette al dispositivo embedded di condividere la connessione Internet del PC host. La configurazione prevede:

- **Gateway di default**: Il PC che crea la rete bridge è configurato come

gateway predefinito per il dispositivo embedded.
- **DNS server**: È stato configurato il server DNS di Google `8.8.8.8`

per la risoluzione dei nomi di dominio.
- **Sincronizzazione temporale**: L'accesso a Internet è essenziale

per il corretto funzionamento del servizio NTP [13], che garantisce la sincronizzazione dell'orologio di sistema con server di tempo accurati. Questa funzionalità è fondamentale per l'integrità dei timestamp dei log e la corretta funzionalità dei meccanismi di scheduling del sistema.

Una volta completate tutte le configurazioni dei pacchetti necessari, Buildroot assembla il filesystem completo e l'immagine del kernel in un unico file `sdcard.img`, pronto per essere scritto su scheda SD e avviato sul dispositivo embedded.

3.3.6 Modulo Kernel CP210x per Comunicazione Seriale

Per abilitare la comunicazione seriale attraverso la porta RS-232 integrata nella host-board Ganador, è stato necessario includere nel kernel il modulo `cp210x` specifico per i controller USB-seriale Silicon Labs.

Questo driver è fondamentale per il corretto funzionamento dell'interfaccia di comunicazione con l'inverter tramite protocollo MODBUS RTU. L'integrazione del modulo è stata realizzata mediante il sistema di configurazione del kernel di Buildroot:

- **Configurazione kernel**: Esecuzione del comando `make linux-menuconfig`

per accedere all'interfaccia di configurazione del kernel.

- **Selezione modulo**: Navigazione nel menu `Kernel modules -> USB Serial Converter support -> USB CP210x family of UART Bridge Controllers` e

attivazione del modulo corrispondente.

Questo approccio garantisce che il modulo sia compilato come parte del kernel personalizzato e caricato automaticamente all'avvio del sistema, rendendo disponibile l'interfaccia seriale per le applicazioni utente senza necessità di interventi manuali.

3.3.7 Servizio NTP per Sincronizzazione Temporale

Per garantire la precisione e l'affidabilità dell'orologio di sistema, essenziale per il corretto funzionamento delle applicazioni e per l'integrità dei log temporali, è stato incluso il pacchetto `ntpd` come client Network Time Protocol.

Il servizio NTP [13] viene automaticamente installato tramite Buildroot e configurato per sincronizzare periodicamente l'orologio locale con server di tempo accurati accessibili tramite Internet. Questa funzionalità è particolarmente importante per:

- **Consistenza temporale**: Garantire che tutti gli eventi di sistema siano

timestampati in modo coerente.

- **Debugging e analisi**: Facilitare l'analisi dei log e la correlazione

degli eventi temporali.

- **Integrazione sistemica**: Assicurare la compatibilità con protocolli e

servizi che dipendono da un orologio di sistema accurato.

La configurazione predefinita utilizza server NTP pubblici, garantendo una sincronizzazione affidabile senza richiedere infrastrutture temporali dedicate.

3.4 Personalizzazioni del Kernel e Bootloader

Durante le fasi intensive di sviluppo e testing, la frequenza dei cicli di rebuild del sistema ha reso necessarie alcune personalizzazioni specifiche dei componenti fondamentali del sistema operativo, in particolare il kernel Linux e il bootloader Barebox.

3.4.1 Patch del Kernel Linux

È stata creata una clonazione dedicata della repository del kernel Linux personalizzata da AMEL, implementando le seguenti modifiche:

- **Abilitazione predefinita modulo CP210x**: Modifica del file

`/drivers/usb/serial/Kconfig` per abilitare il modulo del driver seriale come configurazione predefinita. Questa modifica elimina la necessità di riabilitare il

manualmente il modulo ad ogni rebuild, accelerando significativamente i cicli di sviluppo e testing.

3.4.2 Patch del Bootloader Barebox

Analogamente, è stato modificato il bootloader Barebox [11] per ottimizzare la sequenza di avvio:

- **Priorità dispositivo di boot**: Modifica dell'ordine di ricerca dei

dispositivi di avvio, dando priorità al dispositivo `mmc2` (scheda SD) rispetto alla memoria flash NAND interna. Questa configurazione è particolarmente utile durante le fasi di sviluppo, quando frequenti aggiornamenti del sistema vengono eseguiti tramite scheda SD.

Queste personalizzazioni sono state implementate per migliorare l'efficienza del flusso di sviluppo, riducendo i tempi di setup tra i cicli di build e test, e garantendo una maggiore ripetibilità dei processi di avvio e configurazione del sistema.

4

Conclusione

Il progetto di sviluppo del sistema embedded per il controllo climatico della camera di collaudo AMEL rappresenta un caso significativo di integrazione tra hardware specializzato, software di controllo real-time e sistemi operativi embedded. La realizzazione di questa soluzione ha permesso di consolidare competenze trasversali che spaziano dall'elettronica di potenza all'ingegneria del software, dall'automazione industriale alla sistemistica Linux.

4.1 Risultati Ottenuti

Il sistema sviluppato ha raggiunto con successo tutti gli obiettivi prefissati, dimostrando affidabilità operativa e precisione nel mantenimento delle condizioni termiche ottimali per le procedure di test. L'implementazione di un algoritmo PID opportunamente sintonizzato ha garantito un controllo stabile e reattivo, capace di gestire efficacemente le variazioni termiche prodotte dai carichi resistivi durante i cicli di collaudo.

L'architettura modulare adottata si è rivelata una scelta strategica vincente, permettendo non solo un efficiente processo di sviluppo, ma anche garantendo manutenibilità ed evoluzione futura del sistema. La separazione delle responsabilità tra i moduli ha facilitato il debugging, l'ottimizzazione delle performance e l'integrazione di nuove funzionalità.

4.2 Aspetti Tecnici Salienti

Dal punto di vista tecnico, il progetto ha evidenziato l'efficacia dell'ecosistema Buildroot per lo sviluppo di sistemi embedded Linux personalizzati. La capacità di creare pacchetti custom ha permesso di integrare perfettamente le librerie LVGL e i moduli di controllo sviluppati internamente, dimostrando la flessibilità del framework nell'adattarsi a requisiti specifici.

La gestione delle risorse hardware limitate è stata affrontata attraverso un'attenta ottimizzazione del kernel, la selezione mirata dei driver necessari e l'implementazione di strategie di scheduling real-time per il processo di controllo critico. Queste ottimizzazioni hanno permesso di raggiungere prestazioni adeguate nonostante i vincoli di memoria e potenza computazionali tipici delle piattaforme embedded.

4.3 Sviluppi Futuri

Il sistema attuale rappresenta una base solida per ulteriori evoluzioni. L'implementazione del web server consentirà estensioni significative in termini di monitoraggio remoto, configurazione avanzata e integrazione con sistemi di supervisione più ampi. L'introduzione di un database per la registrazione storica dei dati termici aprirà possibilità di analisi predittiva e ottimizzazione dei processi di collaudo.

Ulteriori direzioni di sviluppo potrebbero includere l'implementazione di algoritmi di controllo più avanzati, l'integrazione con sensori addizionali per il monitoraggio di altri parametri ambientali, e lo sviluppo di interfacce utente più sofisticate basate su tecnologie web moderne.

4.4 Considerazioni Finali

Questo progetto dimostra come l'integrazione di tecnologie open-source, standard industriali consolidati e competenze di sviluppo embedded possa portare alla realizzazione di soluzioni complesse ed efficaci per problemi industriali reali. L'approccio metodologico adottato, basato su un'architettura modulare, sull'ottimizzazione delle risorse e sull'attenzione alla manutenibilità, costituisce un modello replicabile per lo sviluppo di sistemi embedded in contesti industriali simili.

L'esperienza acquisita attraverso questo progetto rappresenta un patrimonio tecnico significativo, applicabile a futuri sviluppi nell'ambito dell'automazione industriale e dell'embedded computing, e testimonia la crescente importanza delle competenze interdisciplinari nello sviluppo di sistemi tecnologici complessi.

A Riferimenti

[1] S. Raimbault, «libmodbus - A Modbus library for Linux, Mac OS X, FreeBSD, QNX and Windows». [Online]. Disponibile su: https://github.com/stephane/libmodbus

[2] L. Contributors, «LVGL». [Online]. Disponibile su: https://github.com/lvgl/lvgl

[3] M. Papaccioli, «Tesi». [Online]. Disponibile su: https://github.com/sbOogway/tesi

[4] M. Integrated, «DS18B20 Programmable Resolution 1-Wire Digital Thermometer». [Online]. Disponibile su: https://www.maximintegrated.com/en/products/sensors/DS18B20.html

[5] L. Contributors, «Linux template for LVGL». [Online]. Disponibile su: https://github.com/lvgl/lv_port_linux

[6] [Online]. Disponibile su: https://www.freedesktop.org/wiki/Software/libevdev/

[7] K. Inc., «CMake - Cross Platform Make». [Online]. Disponibile su: https://cmake.org/

[8] L. Torvalds e L. kernel developers, «The Linux Kernel Archives». [Online]. Disponibile su: https://www.kernel.org/

[9] B. contributors, [Online]. Disponibile su: https://buildroot.org/

[10] M. T. Inc., «AT91Bootstrap». [Online]. Disponibile su: https://github.com/linux4sam/at91bootstrap

[11] B. developers, «Barebox - bootloader for embedded systems». [Online]. Disponibile su: https://barebox.org/

[12] O. developers, «OpenSSH». [Online]. Disponibile su: https://www.openssh.com/

[13] N. T. Foundation, «Network Time Protocol - NTP». [Online]. Disponibile su: https://www.ntp.org/