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, situata presso AMEL s.r.l., è utilizzata per testare carichi resistivi. Questi producono calore, rendendo necessario il raffreddamento dell'ambiente, soprattutto nei mesi più caldi.

Il sistema embedded controlla la frequenza di una ventola di raffreddamento mediante un controllo PID:

1. La temperatura ambiente viene letta tramite un sensore.
2. Questa viene utilizzata come ingresso del sistema PID.
3. L'algoritmo PID calcola la tensione da inviare a un inverter, il quale determina la frequenza della ventola in base all'errore attuale (azione proporzionale) e a quello accumulato (azione integrativa).
4. Questo sistema di retroazione negativa viene applicato continuamente per mantenere costante la temperatura ambiente.

La comunicazione tra l'inverter e il sistema embedded avviene tramite il protocollo MODBUS RTU.

Il sistema embedded fornisce inoltre un'interfaccia per regolare la temperatura target dell'ambiente mediante un display touchscreen. La GUI è sviluppata utilizzando la libreria LVGL.

Il sistema embedded è collegato alla rete aziendale tramite Ethernet; attraverso un web server sarà possibile regolare la temperatura target anche da remoto.

La comunicazione tra il sistema embedded e il web server avviene mediante scrittura su file o IPC (Inter-Process Communication).

È ancora da valutare l'impiego di un database per registrare la temperatura nel tempo. In tal caso, il web server gestirà l'interazione con esso.

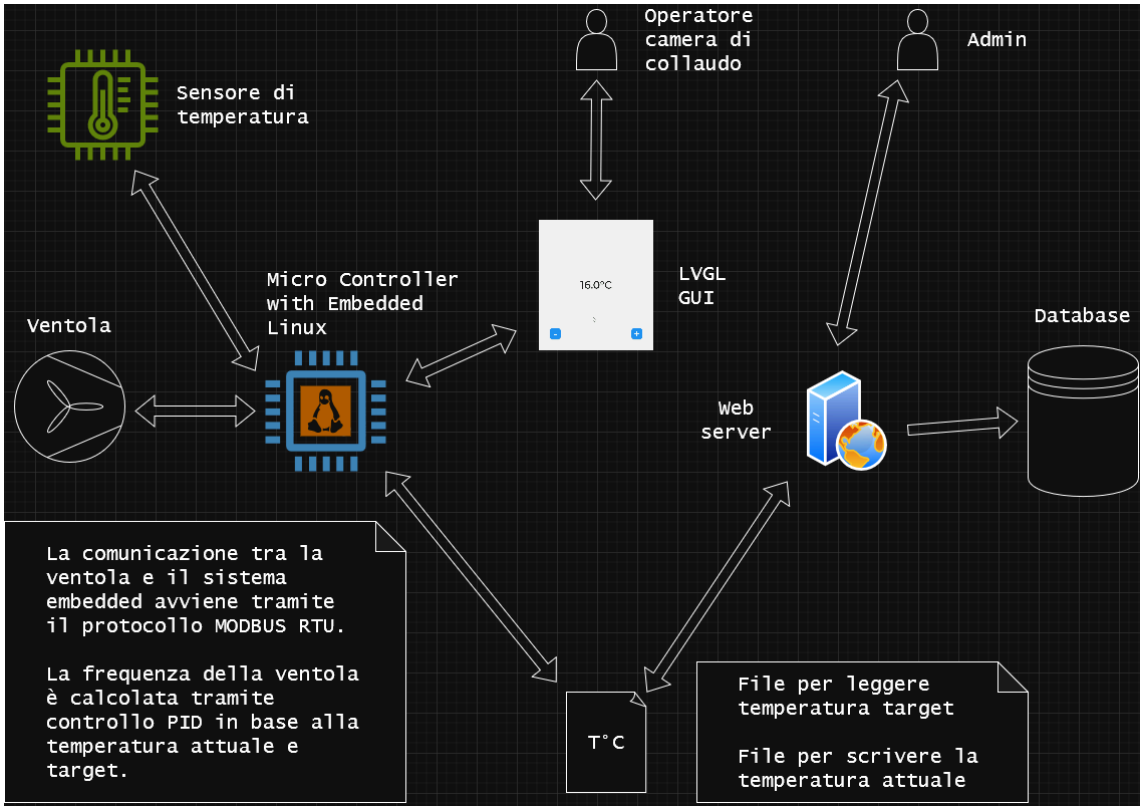

Figura 1 — Diagramma che illustea il sistema

1.2 Struttura del codice sorgente

Il codice sorgente del progetto è pubblicato su GitHub [1]. Questa repository è composta da più sottomoduli:

- tesi-gui [2]
- tesi-overleaf [3]
- tesi-modbus

Questo approccio è stato adottato per garantire modularità e organizzazione del software [4].

2

Control

2.1 Graphical User Interface

Per controllare la temperatura della camera di collaudo, l'operatore imposta la temperatura target mediante un display touchscreen «NOME DISPLAY». L'interfaccia grafica è sviluppata utilizzando la libreria LVGL [5] e si è utilizzato un template [6] contenente il porting su Linux fornito dagli sviluppatori della libreria.

2.1.1 Funzioni di callback nel ciclo principale della GUI LVGL

```
// TODO CHANGEME READ THIS FROM A FILE
static float target_temperature = read_temperature_file();
lv_obj_t * screen;
lv_obj_t * target_temperature_label;

const int padding_button = 50;
const int height_button  = 50;
const int width_button   = 50;

static void increment_temperature(lv_event_t * e)
{
    if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
        return;
    }
    target_temperature++;
    lv_label_set_text_fmt(
        target_temperature_label, "%.1f°C", target_temperature
    );
}

static void decrement_temperature(lv_event_t * e)
{
    if(lv_event_get_code(e) != LV_EVENT_CLICKED) {
        return;
    }
    target_temperature--;
    lv_label_set_text_fmt(
        target_temperature_label, "%.1f°C", target_temperature
    );
}
```

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

2.1.2 Albero dei widget nel ciclo principale

```
screen = lv_scr_act();
lv_obj_t * increment_temperature_button = lv_btn_create(screen);
lv_obj_align(
    increment_temperature_button, LV_ALIGN_BOTTOM_RIGHT,
    -padding_button, -padding_button);
lv_obj_set_height(increment_temperature_button, height_button);
lv_obj_set_width(increment_temperature_button, width_button);
lv_obj_add_event_cb(
    increment_temperature_button, increment_temperature,
    LV_EVENT_ALL, NULL
);

lv_obj_t * increment_temperature_label = lv_label_create(
    increment_temperature_button
);
lv_label_set_text(increment_temperature_label, "+");
lv_obj_set_style_text_font(
    increment_temperature_label, &lv_font_montserrat_48, 0
);
lv_obj_center(increment_temperature_label);

lv_obj_t * decrement_temperature_button = lv_btn_create(screen);
lv_obj_align(
    decrement_temperature_button, LV_ALIGN_BOTTOM_LEFT,
    padding_button, -padding_button);
lv_obj_set_height(decrement_temperature_button, height_button);
lv_obj_set_width(decrement_temperature_button, width_button);
lv_obj_add_event_cb(
    decrement_temperature_button, decrement_temperature,
    LV_EVENT_ALL, NULL
);

lv_obj_t * decrement_temperature_label = lv_label_create(
    decrement_temperature_button
    );
lv_label_set_text(decrement_temperature_label, "-");
lv_obj_set_style_text_font(
    decrement_temperature_label, &lv_font_montserrat_48, 0
);
lv_obj_center(decrement_temperature_label);

target_temperature_label = lv_label_create(screen);
lv_label_set_text_fmt(
    target_temperature_label, "%.1f°C", target_temperature
);
lv_obj_set_style_text_font(
    target_temperature_label, &lv_font_montserrat_48, 0
);
lv_obj_align(target_temperature_label, LV_ALIGN_CENTER, 0, 0);
```

Codice 2 — Creazione dei widget nell'interfaccia grafica

I backend utilizzati da LVGL per l'I/O sono libevdev e il framebuffer device. Sono stati scelti per la loro semplicità e il ridotto utilizzo di risorse.

Libevdev [7] è una libreria che gestisce gli eventi di input: riceve i tocchi dal touchscreen e li passa all'interfaccia grafica.

Il framebuffer device è semplicemente il file `/dev/fb0`, scritto dalla GUI, che contiene il colore di ciascun pixel dello schermo.

2.2 Compilazione della GUI

Per la compilazione dell'applicazione è necessaria una toolchain adatta all'architettura ARM. Nel nostro caso, ci affidiamo al compilatore e alle librerie fornite da Buildroot.

2.2.1 `cross_compile_setup.cmake`

```
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 3 — cross_compile_setup.cmake

Il comando `cmake -DCMAKE_TOOLCHAIN_FILE=./cross_compile_setup.cmake -B build -S .` genera i Makefile necessari per la cross-compilazione, che vengono poi eseguiti con `make -C build -j`.

LVGL viene compilata come libreria condivisa, mentre l'applicazione come eseguibile.

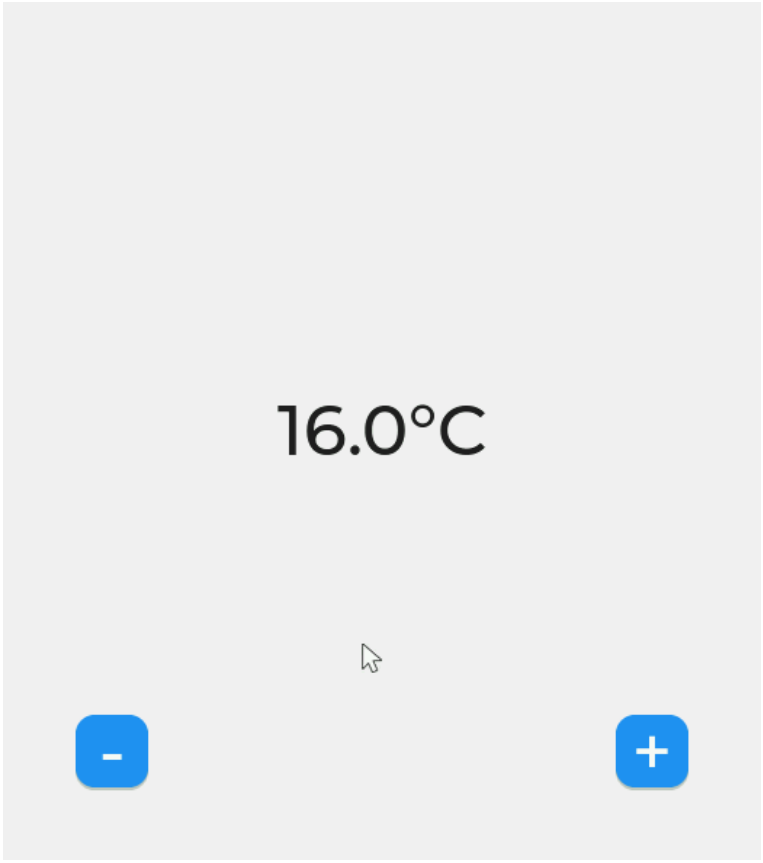

Figura 2 — Interfaccia grafica per il controllo della temperatura

2.3 Admin Control

2.4 Logging and Monitoring

3

Embedded Linux

3.1 Hardware

La scheda utilizzata per il sistema embedded è sviluppata da AMEL e comprende:

- una host-board Ganador (rev. 4);
- un system-on-module Vulcano-A5;
- una CPU ARM;
- una memoria SD utilizzata come disco fisso;
- un'interfaccia Ethernet;
- un'interfaccia seriale;
- un display touchscreen.

3.2 Costruzione del sistema

Per orchestrare il sistema è stato utilizzato Linux. È stato creato un kernel personalizzato con soli i moduli essenziali per il funzionamento, data la limitatezza delle risorse hardware. Lo strumento utilizzato per completare l'opera è Buildroot [8], che consiste essenzialmente in una serie di Makefile per installare e cross-compilare tutte le librerie e i pacchetti necessari alla costruzione e all'esecuzione del sistema. Inoltre, si occupa di creare il filesystem e di prepararlo in un'immagine pronta per essere scritta sulla scheda SD del sistema embedded.

Per aggiungere l'interfaccia grafica sviluppata con LVGL è stato necessario creare un nuovo pacchetto in Buildroot.

All'avvio del sistema, il bootloader del chip attiva AT91bootstrap, che a sua volta avvia Barebox, il quale carica il kernel in memoria.

3.3 Personalizzazione di buildroot

Per aggiungere un pacchetto a Buildroot è necessario inserire una nuova voce nella cartella `package`, comprendente un file `Config.in` e un file `.mk`.

Questi due file contengono le istruzioni che consentono a Buildroot di risolvere le dipendenze, scaricare e installare il pacchetto nel filesystem del dispositivo target.

3.3.1 Il pacchetto amel-temp-control

```
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

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 4 — amel-temp-control.mk

Inizialmente, viene clonata la repository temp-control, contenente la GUI, al commit specificato, inizializzando i sottomoduli e verificando la presenza della dipendenza `libevdev`.

Successivamente, vengono cross-compilate la libreria LVGL e l'applicazione con interfaccia grafica utilizzando il compilatore ARM fornito da Buildroot.

Infine, i binari della libreria e l'eseguibile dell'applicazione vengono installati sulla macchina target.

3.3.2 Il pacchetto amel-pid

```
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 5 — amel-pid-control.mk

Analogamente, è stato creato il pacchetto `amel-pid-control` per cross-compilare e installare la libreria PID sviluppata in C++.

3.3.3 Modifica pacchetti networking

Dopo aver ripetuto questo processo per tutti i pacchetti desiderati, il filesystem e l'immagine del kernel vengono assemblati in un file `sdcard.img` pronto per essere scritto su una scheda SD e avviato sul dispositivo embedded.

3.4 File I/O

3.5 PID Control

3.5.1 Sensore di temperatura

I sensori di temperatura utilizzati sono due DS18B20 collegati in parallelo su un bus 1-Wire.

Il binario `pid` legge periodicamente la temperatura dai sensori e calcola il valore di output del controller PID in base alla temperatura misurata e al setpoint desiderato.

3.6 MODBUS RTU

Per comunicare con l'inverter che controlla la ventola di raffreddamento, è stato utilizzato il protocollo MODBUS RTU tramite l'apposita libreria `libmodbus`.

4

Conclusione

A Riferimenti

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

[2] M. Papaccioli, «GUI». [Online]. Disponibile su: https://github.com/sbOogway/tesi-gui

[3] M. Papaccioli, «LaTeX source code». [Online]. Disponibile su: https://github.com/sbOogway/tesi-overleaf

[4] G. contributors, «7.11 Git Tools - Submodules». [Online]. Disponibile su: https://git-scm.com/book/en/v2/Git-Tools-Submodules

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

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

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

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

B Glossario