Smart Temperature Monitoring System

Assignment#03 per il corso di Sistemi Embedded e IOT

Autore: Nicolò Morini
Contatto: nicolo.morini2@studio.unibo.it
Anno accademico: 2024/2025

1 Progettazione

Il sistema è suddiviso in quattro sotto-sistemi principali:

- **Window-controller subsystem** (utilizzando Arduino UNO)
- **Temperature-monitoring subsystem** (utilizzando ESP32-s3)
- **Control-Unit** (in Python, utilizzando Flask framework)
- **Dashboard-frontend** (HTML/CSS/JavaScript)

2 Window-controller subsystem

2.1 Architettura del sistema

Il Window Controller Subsystem è stato progettato utilizzando un'architettura basata su Finite State Machine (FSM) asincrona che gestisce le transizioni tra le modalità operative del sistema e coordina l'interazione tra i diversi componenti hardware.

La FSM implementa tre stati principali:

- **INIT**: Stato di inizializzazione del sistema
- **AUTOMATIC**: Modalità automatica con controllo da parte della Control Unit tramite Serial Line
- **MANUAL**: Modalità manuale con controllo locale tramite potenziometro

Grazie al design asincrono, garantisce risposte real-time. Infatti la FSM reagisce a eventi esterni senza bloccare l'esecuzione, gestisce in modo asincrono la comunicazione seriale e gli input dell'utente, senza introdurre delay bloccanti.

In questo sotto-sistema non è stata utilizzata una Task-based Architecture per i seguenti motivi:

- Il sistema ha requisiti di timing non particolarmente complicati, quindi introdurrebbe complessità non necessaria
- La FSM asincrona garantisce già responsività adeguata per l'applicazione

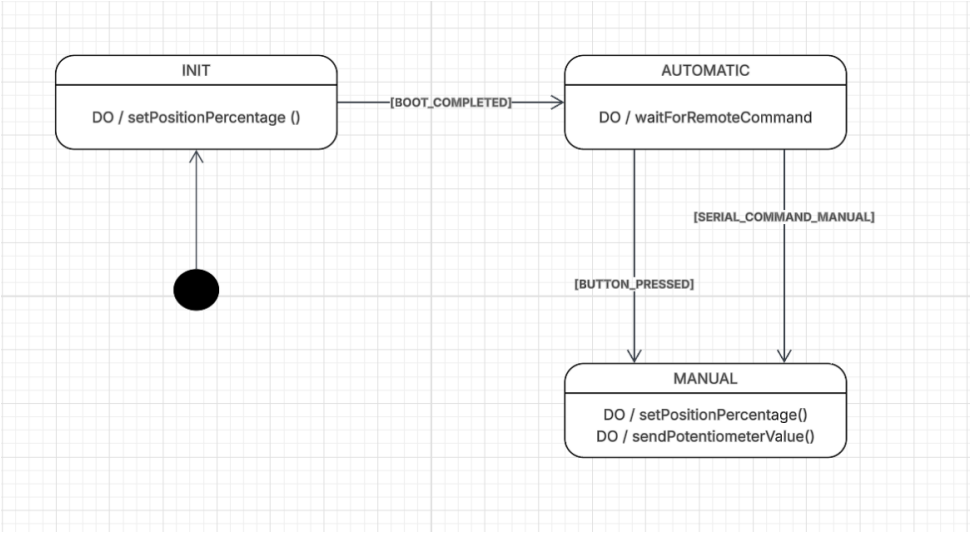

Figura 1: FSM del Window Controller Subsystem

2.2 Architettura del codice

L'implementazione adotta il pattern di progettazione "Strategy" per garantire modularità e manutenibilità del codice.

Per ogni componente del sistema è stata implementata una gerarchia di astrazioni articolata su più livelli:

- **Interfaccia Astratta Base**: Definisce i contratti funzionali indipendenti dall'hardware specifico
- **Interfaccia Hardware specifica**: Estende l'interfaccia base con contratti specifici per i componenti utilizzati
- **Implementazione concreta**: Realizza le funzionalità utilizzando l'interfaccia hardware specifica

L'architettura si articola sui seguenti componenti principali:

- **ServoMotor**: Controllo del motore servo per il posizionamento della finestra
- **UserInputSource**: Gestione input utente (pulsante modalità, potenziometro)
- **ControlUnitLink**: Comunicazione seriale bidirezionale con la Control Unit
- **LcdView**: Interfaccia display LCD per feedback visivo
- **ISystemFSM**: Logica della FSM

3 Temperature monitoring subsystem

3.1 Architettura del sistema

Il Temperature monitoring subsystem è stato progettato utilizzando un'architettura uguale a quella impiegata per il sotto-sistema precedente, ossia utilizzando una Finite State Machine asincrona event-driven.

La FSM implementa otto stati principali:

- **INITIALIZING**: Stato di inizializzazione del sistema
- **WIFI_CONNECTED**: Connessione WiFi stabilita (stato di transizione)
- **MQTT_CONNECTING**: Tentativo di connessione MQTT in corso
- **OPERATIONAL**: Funzionamento normale con connessioni attive
- **SAMPLING_TEMPERATURE**: Acquisizione dati dal sensore TMP36
- **SENDING_DATA**: Trasmissione dati via MQTT
- **NETWORK_ERROR**: Errore di connettività di rete
- **WAIT_RECONNECT**: Attesa prima del tentativo di riconnessione

Anche in questo caso il design asincrono garantisce responsività in tempo reale attraverso la risposta agli eventi (ad esempio di connessione/disconnessione) e alla gestione asincrona delle connessioni WiFi e MQTT (utilizzando Mosquitto come broker).

Grazie alla libreria PubSubClient inoltre, è stato implementato un meccanismo di callback per il quale la ricezione dei messaggi MQTT avviene tramite funzioni di callback chiamate automaticamente quando arrivano nuovi messaggi, invece di controllare manualmente nel loop se ci sono messaggi in arrivo (evito così l'utilizzo di un polling sincrono che mi causerebbe maggiori attese).

In questo caso, l'utilizzo di ESP32-s3, mette a disposizione FreeRTOS preinstallato, ossia un sistema operativo real-time che gestisce automaticamente la creazione di task per determinate operazioni quali:

- Task principale che esegue il loop()
- Task di sistema per WiFi
- Task di librerie per MQTT

Quindi anche in questo caso l'utilizzo di una architettura Task-Based introdurrebbe inutile complicatezza del codice, dato che la FSM asincrona è sufficiente per gestire il sotto-sistema (aiutato in questo caso anche dalla presenza del sistema operativo).

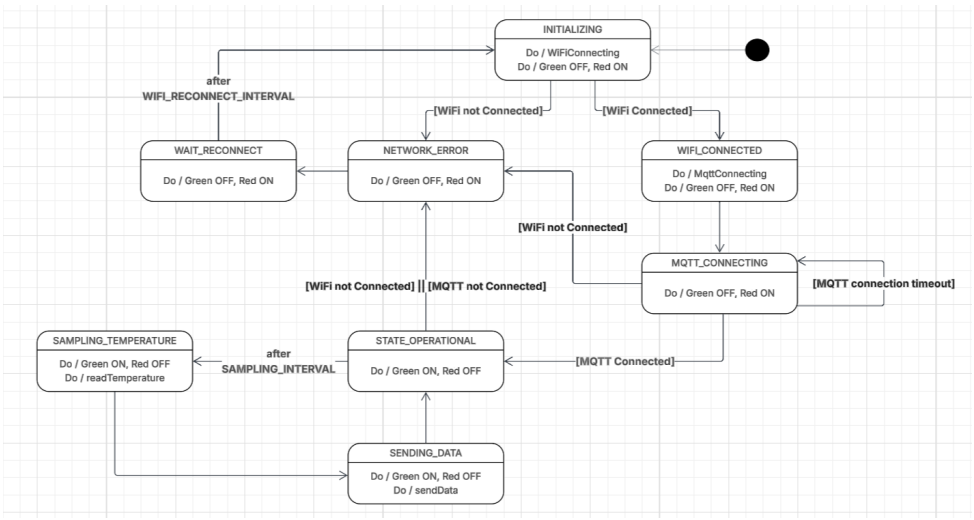

Figura 2: FSM del Temperature Monitoring Subsystem

3.2 Architettura del codice

Anche in questo sotto-sistema è stato utilizzato il pattern "Strategy" per l'architettura del codice.

I componenti principali in cui si articola l'architettura sono:

- **TemperatureManager**: Interfaccia per l'acquisizione dati dal sensore TMP36
- **LedStatus**: Gestione feedback visivo tramite LED di stato
- **WifiManager**: Gestione connessione WiFi con retry automatico
- **MqttManager**: Comunicazione bidirezionale con il broker MQTT
- **IFsmManager**: Logica della FSM

4 Control Unit Backend

La Control Unit rappresenta il centro di coordinamento dell'intero sistema, implementata come applicazione Python che integra comunicazione MQTT, controllo seriale e interfaccia web.

4.1 Utilizzo di Flask Framework

Per l'implementazione dell'interfaccia web è stato scelto Flask, un micro-framework Python che fornisce le funzionalità essenziali per lo sviluppo web senza complessità aggiuntive non necessarie.

Flask fornisce l'API REST che espone diversi endpoints per la comunicazione HTTP, restando quindi separata dalla logica applicativa.

Quando viene avviata l'applicazione Python, Flask funge da web server (tramite il development server integrato) che:

- Serve i file statici della dashboard (HTML, CSS, JS)
- Espone gli endpoint API per la comunicazione con il frontend
- Gestisce le richieste HTTP sia per l'interfaccia utente sia per l'API

5 Dashboard Frontend

La dashboard è implementata come Single Page Application utilizzando tecnologie web standard (HTML5, CSS3, JavaScript).

Viene fornita un'interfaccia grafica completa per il monitoraggio e controllo del sistema.

La dashboard utilizza comunicazione asincrona tramite:

- **Polling HTTP**: Richieste periodiche a `/api/status` per aggiornamenti real-time
- **REST API calls**: Invio comandi tramite POST requests agli endpoint Flask
- **JSON data exchange**: Scambio dati in formato JSON

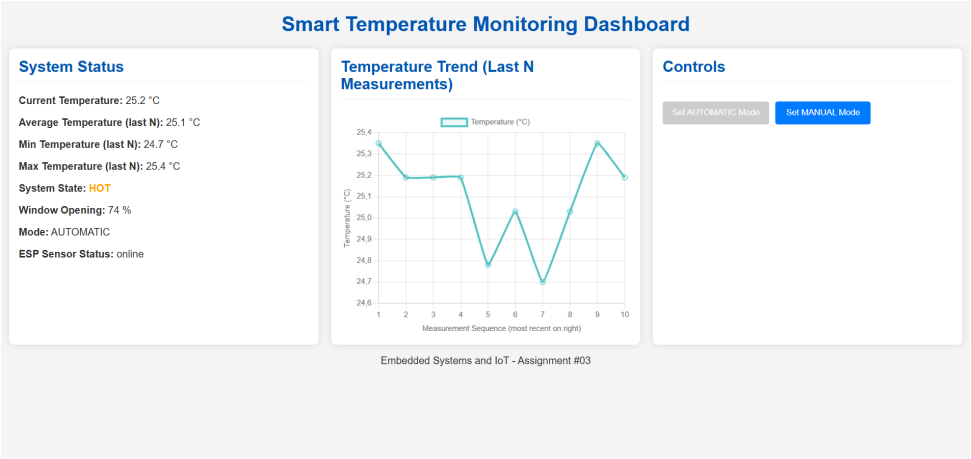

Figura 3: Dashboard screenshot