Relazione Assignment-03

De Leonardis Federico - 0001071569

Monaco Andrea - 0001070845

Guerrini Alex - 0001090495

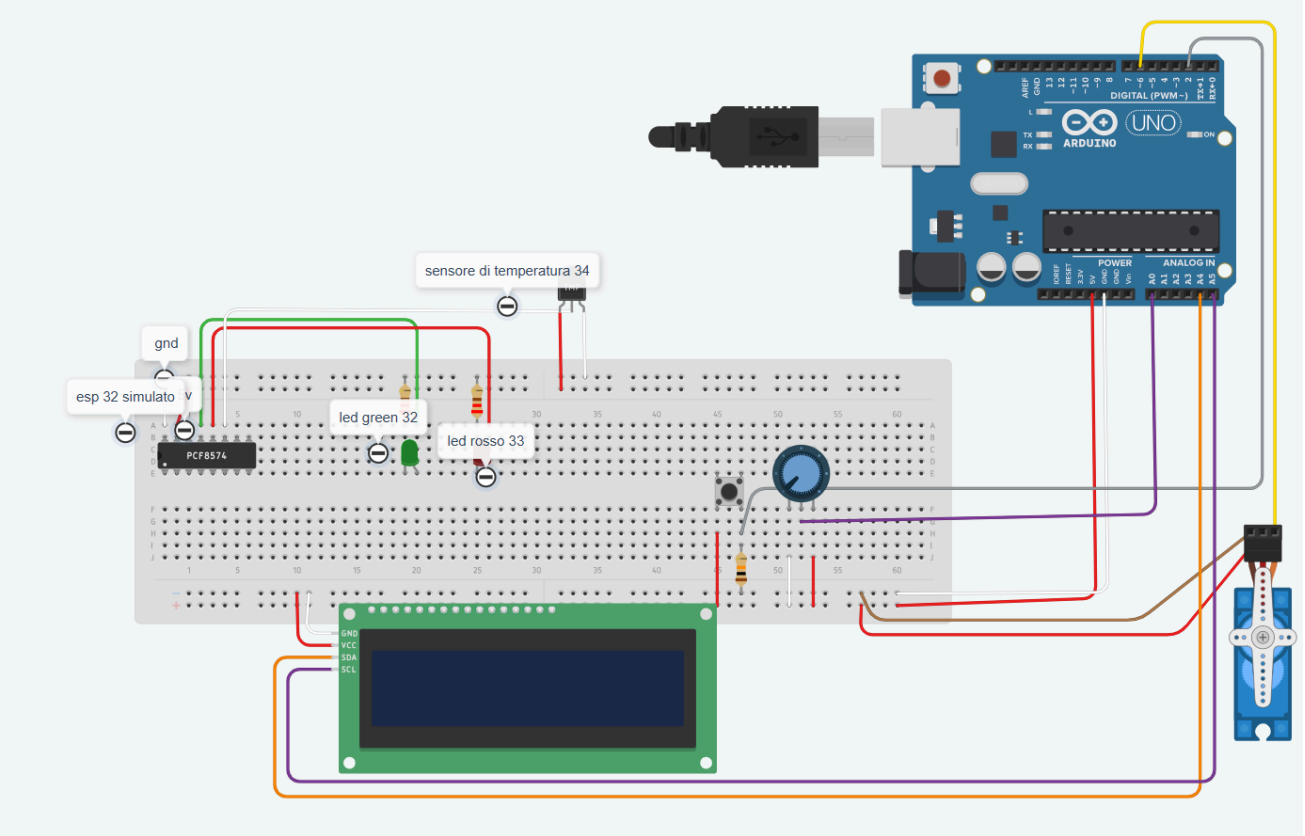
Video Presentazione:

Elaborato 3 IoT.mp4

Introduzione:

Per questo progetto l'obiettivo consisteva nel simulare il monitoraggio di un ambiente chiuso dotato di una finestra. Come si può vedere dal video di presentazione, è stato deciso di creare un modello fisico, costituito da una scatola in legno, per migliorare l'impatto visuale e l'aspetto testing.

Componenti Hardware:



Anche se sono stati utilizzati meno sensori rispetto all'assignment precedente, la sfida di questo progetto era creare una sinergia funzionante tra le varie unità operative. Di fatto abbiamo un'unità di controllo, che elabora e invia dati a tutte le altre.

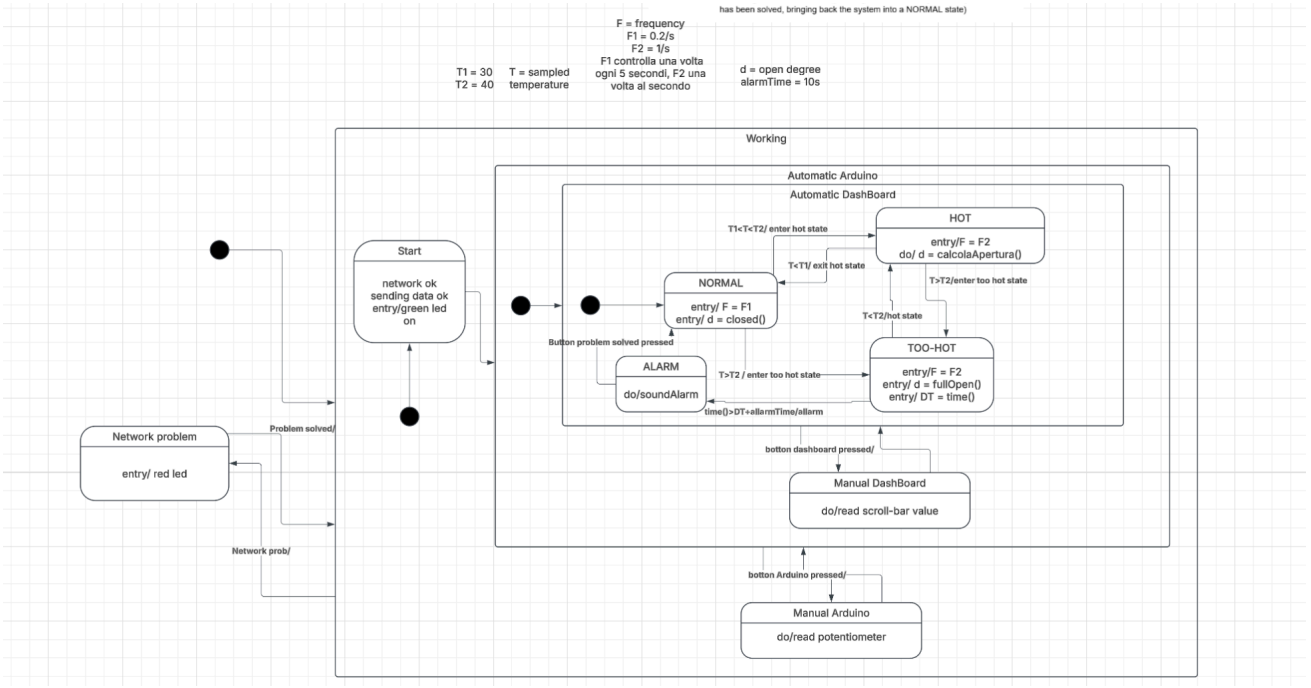
- Window Controller (arduino), dati i parametri di temperatura e apertura d'angolo, si occupa dell’apertura della finestra e della visualizzazione dei parametri tramite lcd.
- Temperature Monitoring (Esp32) si occupa di connettersi alla rete internet per creare una connessione mqtt. Su di essa verranno inviati i dati raccolti da un sensore di temperatura, e poi utilizzati dalla Control Unit
- DashBoard(front end), permette di visualizzare su un monitor un grafico con le ultime temperature, lo stato della macchina e la possibilità di modificare manualmente l’apertura della finestra

Architettura:

Il sistema è quindi gestito dalla control unit, che tramite un sistema a loop sincronizza gli altri componenti, seguendo questo flusso esecutivo:

1. Temperature Monitoring invia sul topic temperature il valore rilevato dal sensore e aggiorna il periodo di campionamento se questo è cambiato nel topic period
2. La control unit controlla se è stato inviato un nuovo dato nel topic temperature, se sì aggiorna lo stato in base al valore ottenuto, invia un nuovo dato sul topic period se esso è cambiato
3. La control unit aggiorna lo stato e l’angolo motore
4. Invia Stato, angolo motore e temperatura alla dashboard
5. La dashboard aggiorna il suo stato e angolo motore che viene ignorato nel caso lo stato dell’interfaccia sia settato in manuale
6. Viene restituito l’angolo motore e lo stato (che cambia se si esce dallo stato di allarme) alla CU, e viene gestito il sistema di allarme
7. Temperatura e angolo motore sono inviati tramite seriale al Window Controller, il quale deciderà o meno se aggiornare l’apertura della porta in base al suo stato

Diagramma degli stati: Visione generale



Task Control Unit:

Task Arduino

Nel contesto di questo progetto, Arduino è utilizzato come componente esecutivo, operando principalmente in modalità automatica. I dispositivi gestiti sono:

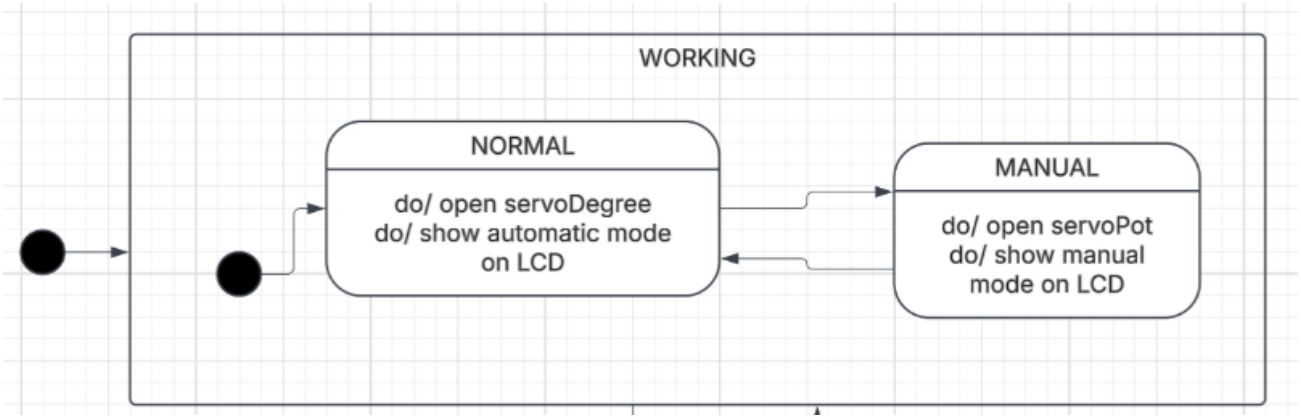
- **Button**: consente di cambiare lo stato da **Automatic** a **Manual** senza dover necessariamente utilizzare interrupt.
- **LCD**: fornisce metodi predefiniti per mostrare informazioni nell'interfaccia utente, come lo stato della macchina e la temperatura ricevuta dalla control unit.
- **ServoMotor**: controlla l'apertura della finestra con un angolo definito tra 0° e 90°.
- **Potentiometer**: in modalità manuale, legge il valore di un potenziometro convertendolo in gradi, valore che viene poi passato al servo motore.

Questi dispositivi sono gestiti dalla classe `WindowManagerMachine`, che include un’istanza di ciascuno di essi e fornisce metodi pratici per il loro utilizzo all'interno dei task.

Per questo componente esecutivo si è scelto di adottare un'**architettura basata su task**, anche se attualmente è presente un solo task. In fase di progettazione si è stabilito che gli stati del sistema fossero due, cioè NORMAL e MANUAL

Nonostante la semplicità della struttura, l'utilizzo di uno **scheduler** e di task migliora la gestione delle tempistiche, la sincronizzazione e rende il sistema più facilmente estendibile

per future implementazioni (ad esempio, l'aggiunta di un buzzer che segnali situazioni di allarme).



Il `servoTask` ha anche il compito di gestire la **comunicazione seriale**: ad ogni tick controlla se sono stati inviati nuovi messaggi tramite il canale seriale. Quando questi messaggi arrivano dalla control unit, il sistema aggiorna i valori di apertura della finestra e della temperatura.

Esp32:

Il componente ESP32 offre nuove potenzialità rispetto ad Arduino Uno:

- Processore a 32 bit e memoria RAM molto superiore.
- Possibilità di connettersi a WiFi e Bluetooth.
- Circa 34 GPIO configurabili.

In questo progetto è stata valorizzata particolarmente la sua capacità di connettersi alla rete, permettendo l'invio e la ricezione di dati dalla control unit tramite il protocollo MQTT.

I tipi di dispositivi implementati sono infatti solamente due:

- **Led**: gestisce l'accensione e lo spegnimento di un LED, utilizzato per determinare se la connessione al server MQTT è andata a buon fine.
- **SensorTemp**: gestisce il sensore di temperatura, restituendo il valore rilevato.

Entrambi questi dispositivi erano già presenti nel progetto precedente e vengono gestiti dalla classe `TempMonitoringMachine`, che verrà utilizzata direttamente nel loop principale.

Si è deciso di non adottare un'architettura basata su task in questo caso, poiché è presente una sola state machine con due stati soltanto (**CONNECTED, NOT_CONNECTED**). Questi stati servono per ripristinare la connessione MQTT nel caso in cui venga persa.

- Nello stato **CONNECTED**, viene inviata la temperatura rilevata e ricevuto il tempo di campionamento per la misurazione successiva.
- Se il compito generale dell'ESP32 fosse stato più complesso, un'architettura basata su task sarebbe stata sicuramente presa in considerazione.

Classe MQTTConnection

Si è deciso di creare una classe apposita per la gestione del collegamento al server MQTT. Questa classe prende in input:

- Il nome e la password della connessione WiFi.
- L'indirizzo del broker MQTT (in questo caso è stato utilizzato Mosquitto).
- Un puntatore contenente i nomi dei relativi topic.

Sarebbe stato possibile utilizzare un unico topic sia per l'invio della temperatura sia per la ricezione del periodo di campionamento, effettuando controlli sui prefissi delle stringhe inviate. Tuttavia, si è preferito mantenere due topic separati per migliorare la coerenza logica, la pulizia del codice e la riusabilità in caso di aggiunta di nuovi topic.

Durante l'inizializzazione di un'istanza della classe, avviene prima di tutto la connessione alla rete WiFi, includendo la libreria `WiFi.h`. Viene impostata la modalità **stazione (WIFI_STA)** e si tenta di connettersi alla rete utilizzando il nome e la password, fino a quando l'operazione non va a buon fine.

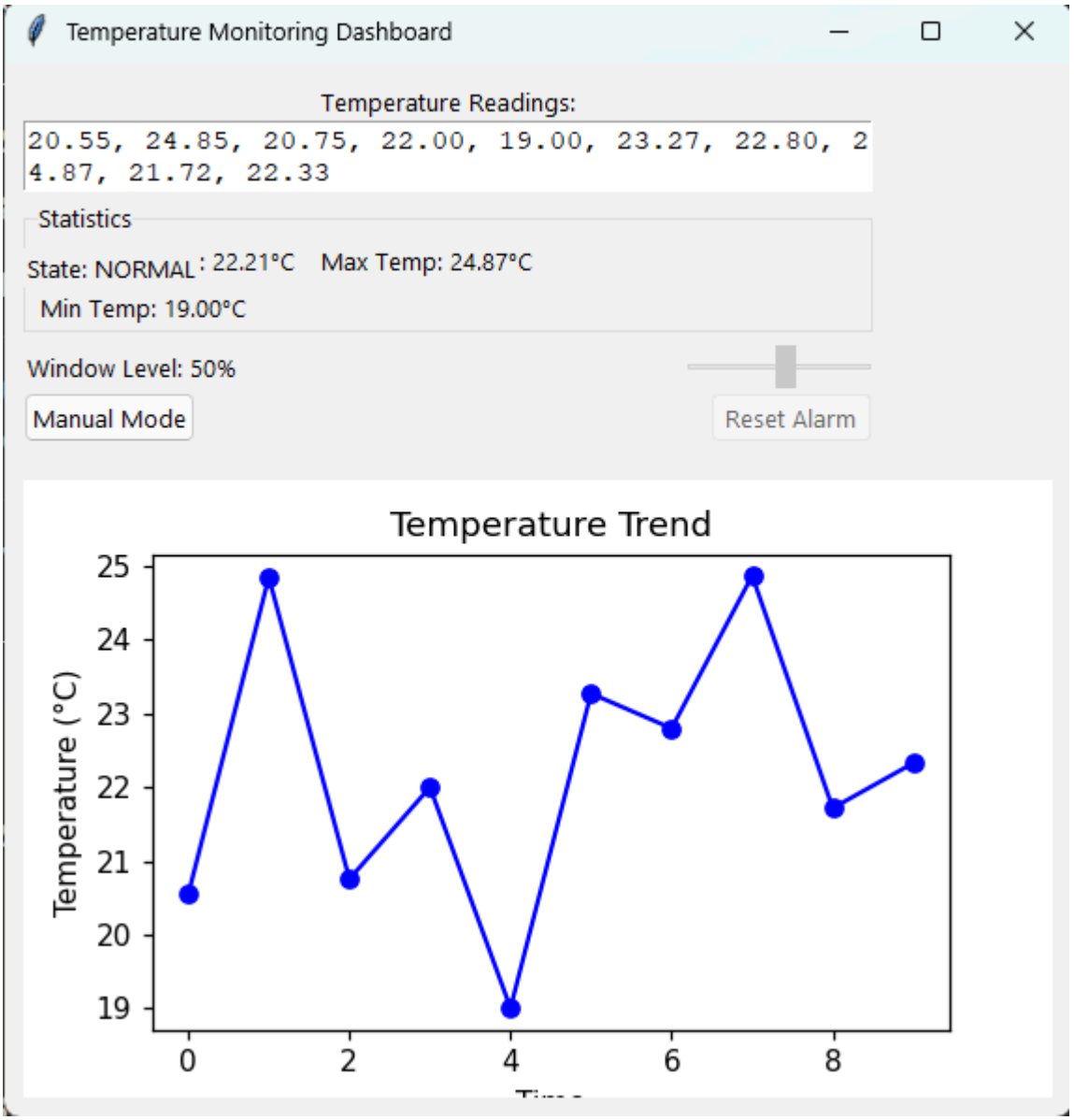
Successivamente, viene configurato l'`espClient` per connettersi al server MQTT. Si tenta quindi una connessione assegnando al client un ID con suffisso randomico, fino a ottenere una connessione riuscita. Il client si iscrive ai due topic disponibili (*IOT-Progetto-03-per* e *IOT-Progetto-03-temp*) e viene impostata una funzione di callback.

Questa funzione di callback viene eseguita ogni volta che il client MQTT riceve un messaggio, ovvero durante l'esecuzione della funzione client.loop nel **main**, quando il sistema si trova nello stato **CONNECTED**. In sintesi, questo metodo controlla l'arrivo di messaggi nei due topic e aggiorna il periodo di campionamento quando un messaggio viene ricevuto su IOT-Progetto-03-per.

Python DashBoard:

La dashboard ha il compito di permettere di comandare e osservare la temperatura e l’apertura del motore con un'interfaccia grafica.

img dashboard:



DashboardApp (Tkinter + Matplotlib) - Dashboard.py

Questa classe crea la dashboard grafica per visualizzare le temperature lette e l'apertura del motore, utilizzando la libreria `Tkinter` per l'interfaccia grafica in sè e `Matplotlib` per il grafico.

Il dashboard include:

- Un display delle letture correnti della temperatura.
- Un grafico in tempo reale che mostra l'andamento delle temperature.
- Una sezione che mostra temperatura media, massima e minima.
- Un controllo per impostare il "window level" in caso di modalità manuale e che ne mostra l'apertura in modalità automatica.

HttpServer.py
Questa classe gestisce un server HTTP con Flask. Il server Riceve un JSON contenente il valore di stato ("HOT", "COLD", ecc.), la temperatura e il livello di finestra. Questi dati vengono poi usati per aggiornare lo stato e la temperatura nella dashboard. Poi risponde inviando, eventualmente, i nuovi window level e stato.

app.py
Il file app.py integra l'interfaccia grafica di Tkinter con il server Flask. All'avvio:

- Viene creata l'istanza di `DashboardApp` per avviare la GUI.
- Un server Flask viene avviato in un thread separato per gestire le richieste HTTP senza bloccare l'interfaccia grafica.

Quando il client invia i dati al server, questi vengono processati dal server Flask e riflessi sulla dashboard in tempo reale.

Control Unit:

Il sistema di controllo presentato è progettato per monitorare e regolare la temperatura di un ambiente utilizzando vari componenti hardware e software. Il sistema comprende la gestione di un motore che regola il livello di apertura di una finestra in base alla temperatura ambiente e alla comunicazione tra diverse componenti tramite diversi protocolli come HTTP, MQTT e seriale.

Componenti Chiave del Sistema

- `ControlUnit`: La classe principale del sistema che gestisce la logica di controllo, monitoraggio della temperatura, aggiornamento dello stato e comunicazione con la dashboard e il motore.
  La classe gestisce il passaggio tra i vari stati (NORMAL, HOT, TOO_HOT, ALARM) in base alla temperatura.
  Implementa anche la logica per inviare i comandi al motore e comunicare con la dashboard.

- `MQTTPeriodSender`: Invia il periodo al broker MQTT solo al variare del valore stesso. Utilizzato per monitorare la frequenza con cui vengono inviati i dati dall'ESP.

- `MQTTTemperatureReceiver`: Si occupa della ricezione dei valori di temperatura dal broker MQTT. Ogni valore di temperatura ricevuto viene elaborato e utilizzato per aggiornare lo stato del sistema e l'angolo di apertura.

- `HttpPostRequest`: Gestisce la comunicazione seriale con il motore per inviare comandi per regolare il livello di apertura della finestra.

- `MotorImpl`: Gestisce il motore e l'angolo di apertura della finestra. Comunica con la porta seriale per inviare il comando relativo all'angolo da impostare.

- `SerialPortDetector`: Permette di rilevare automaticamente la porta seriale a cui è connesso il dispositivo Arduino (o compatibile), utilizzato per comunicare con il motore.

Link al repository GitHub: Assignment-03 Repository