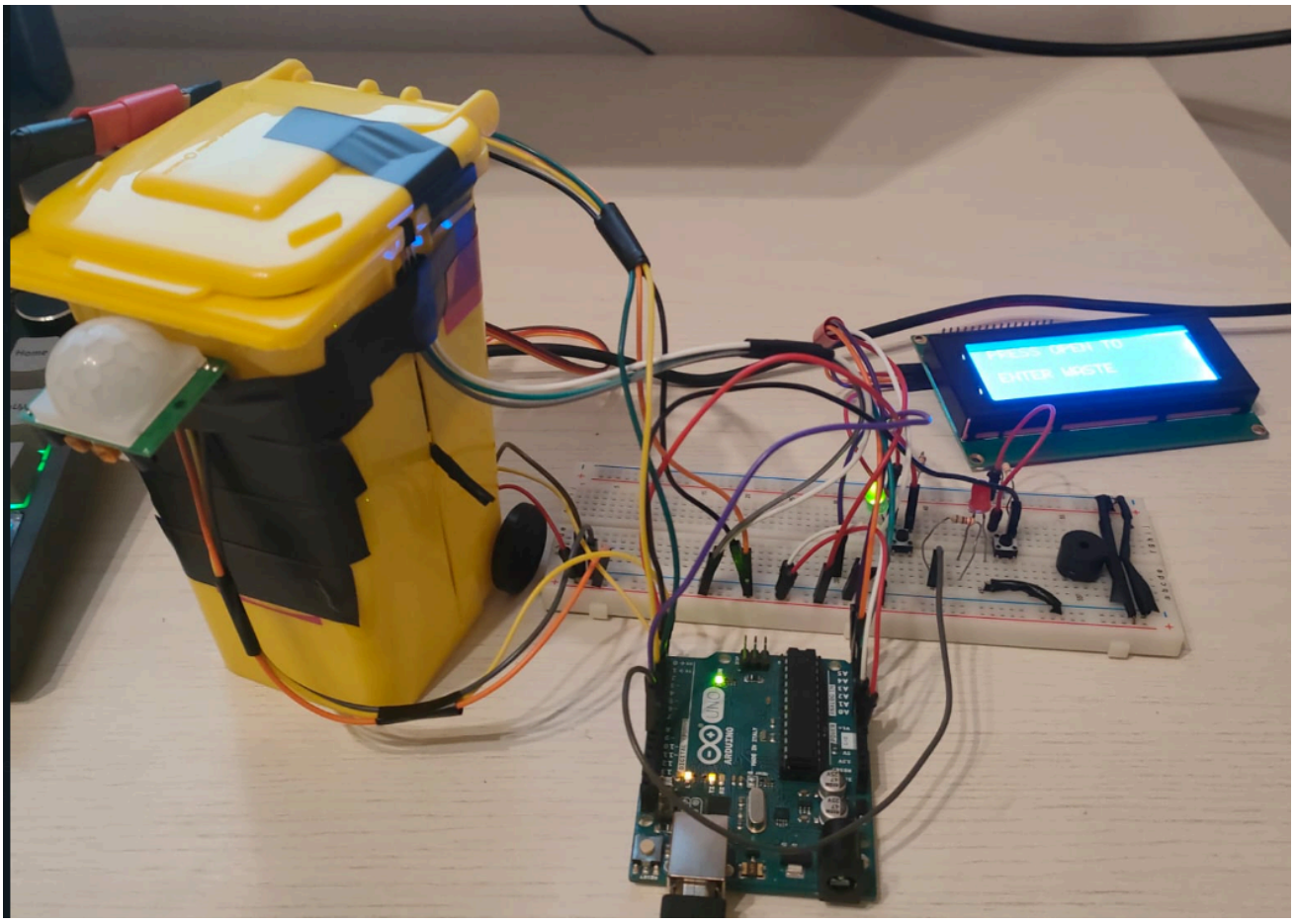
Relazione Assignment-02

De Leonardis Federico - 0001071569

Introduzione

Per questo progetto, dato il tema pratico e coinvolgente, ho deciso di realizzare una versione fisica del contenitore, rendendo l'esercitazione più immersiva e pratica da testare.



Architettura

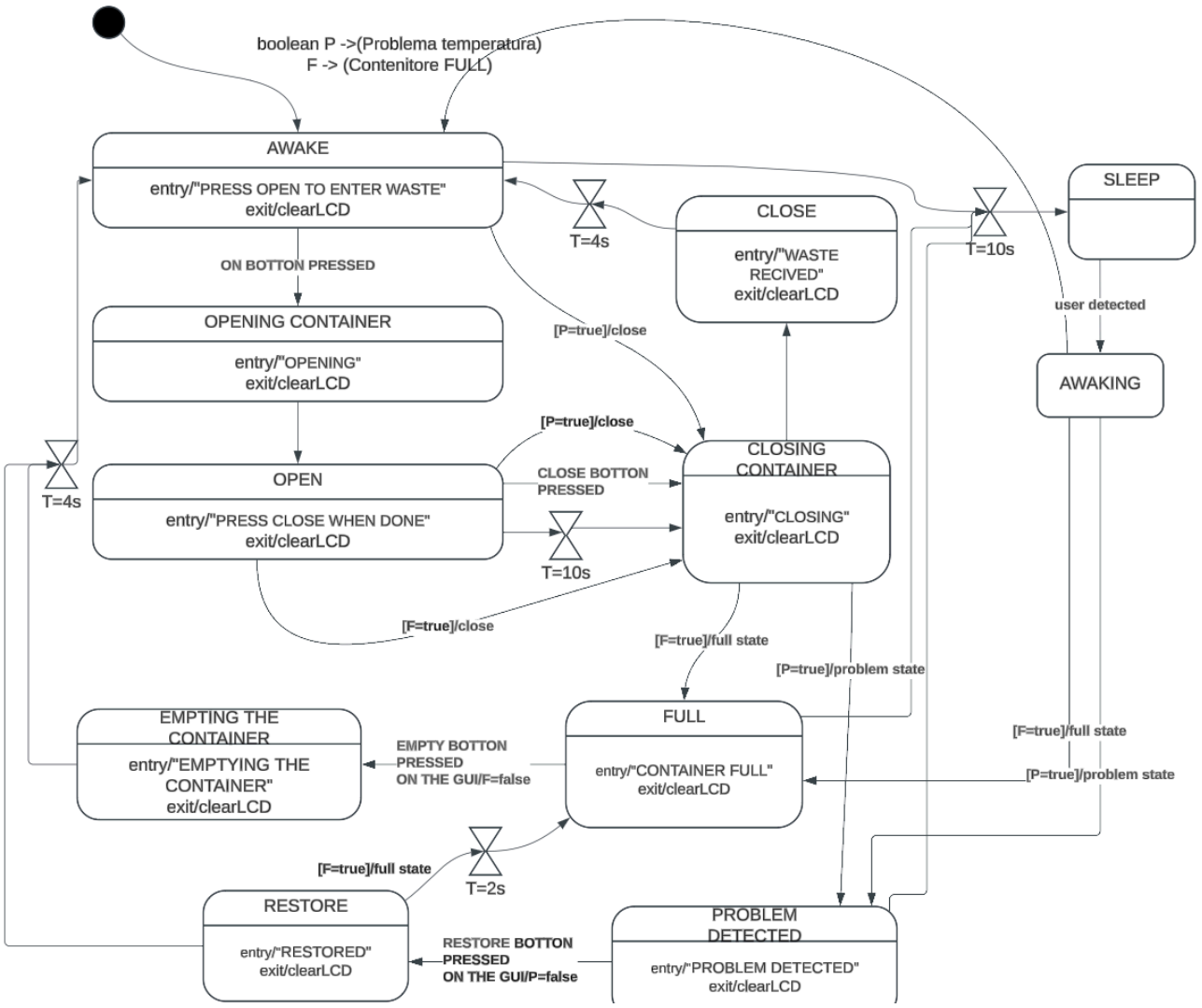
L'architettura software utilizzata si basa su una struttura task-based con macchine a stati finiti sincrone. In pratica, ciò significa che l'implementazione è partita dalla creazione dei *device*, ovvero classi che trasformano il comportamento dei sensori in oggetti facilmente utilizzabili dalle altre classi.

Sono stati creati *device* per quasi tutti i sensori: LCD, LED, sensore di temperatura, sonar, sensore di presenza (PIR) e servomotore.

Nota sul servomotore: è stata utilizzata la libreria *ServoTimer2*, che consente al servo di funzionare con un timer diverso da *Timer1*, il quale è stato invece utilizzato dallo *scheduler*.

Lo *scheduler* è un elemento fondamentale in questo tipo di architettura: il suo compito è organizzare i vari task in periodi prestabiliti, eseguendo in sequenza ciascun metodo `tick()`, che rappresenta una singola iterazione (o passo) di ciò che il task deve compiere in quello specifico stato.

Tasks: Visione generale



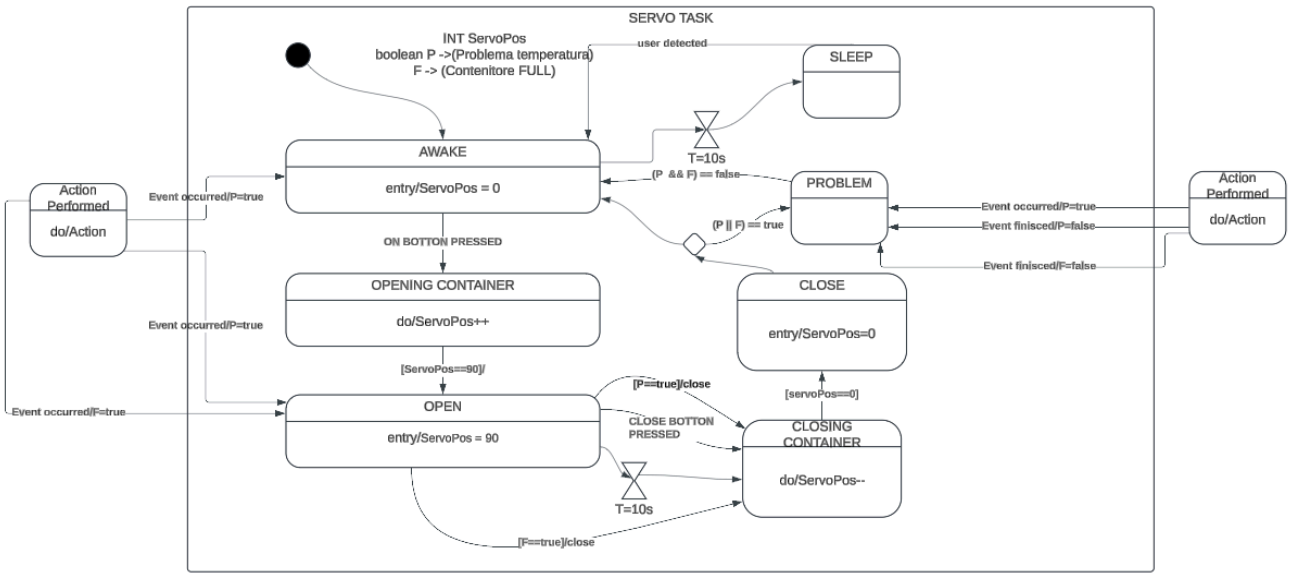
Il diagramma mostra cosa fanno i vari task nel loro insieme, evidenziando i diversi stati in cui la macchina può trovarsi dopo vari passaggi e condizioni.

Descrizione dettagliata degli stati:

- **AWAKE**: Stato base in cui la macchina accetta la maggior parte degli input dai *device*. Da qui è possibile:
    - Aprire la porta del contenitore
    - Entrare in *sleep* dopo 10 secondi di inattività (se il PIR non rileva cambiamenti)
    - Passare agli stati **FULL** o **PROBLEM DETECTED** in caso di necessità di reset tramite console.
- **OPENING CONTAINER**: Transizione verso lo stato **OPEN**.
- **OPEN**: La porta del contenitore è aperta. Dopo 10 secondi, se il pulsante non è premuto, la porta si chiude. La chiusura anticipata avviene se il contenitore è pieno o se la temperatura è anomala.
- **CLOSING CONTAINER**: Transizione verso **CLOSE** o, in caso di anomalie, verso **FULL** o **RESTORE**.
- **CLOSE**: Notifica all'utente la chiusura. Dopo 4 secondi, il bidone sarà nuovamente apribile.
- **FULL**: Stato in cui il contenitore è pieno oltre una certa soglia. Può uscirne solo premendo il pulsante *EMPTY CONTAINER* sull'interfaccia grafica. Dopo 10 secondi di inattività, entra in *sleep*.
- **PROBLEM DETECTED**: Stato simile a **FULL**, ma legato al controllo della temperatura interna.
- **EMPTY CONTAINER** e **RESTORE**: Permettono il ritorno allo stato **AWAKE** dopo 4 secondi dalla pressione dei relativi pulsanti della GUI.
- **SLEEP**: Stato di risparmio energetico. Si entra in questo stato dopo 10 secondi di inattività negli stati **AWAKE**, **FULL** e **PROBLEM DETECTED**. Rilevata la presenza di un utente, il sistema ritorna allo stato precedente.

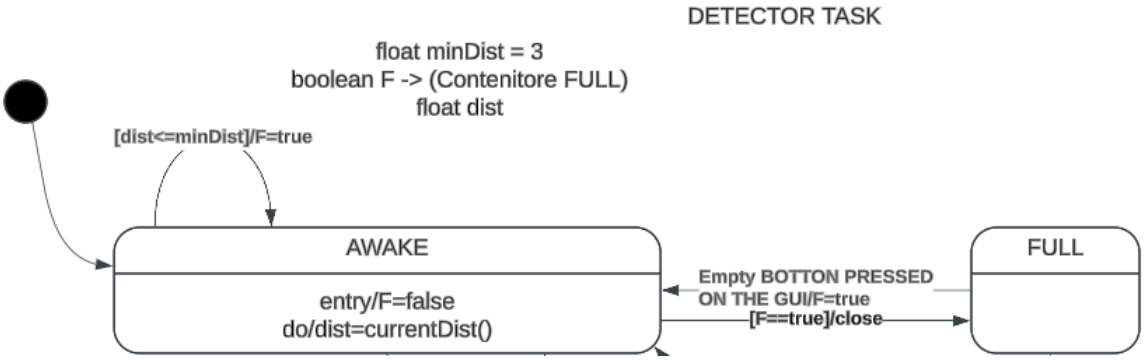
Tasks: Dettaglio

ServoTask



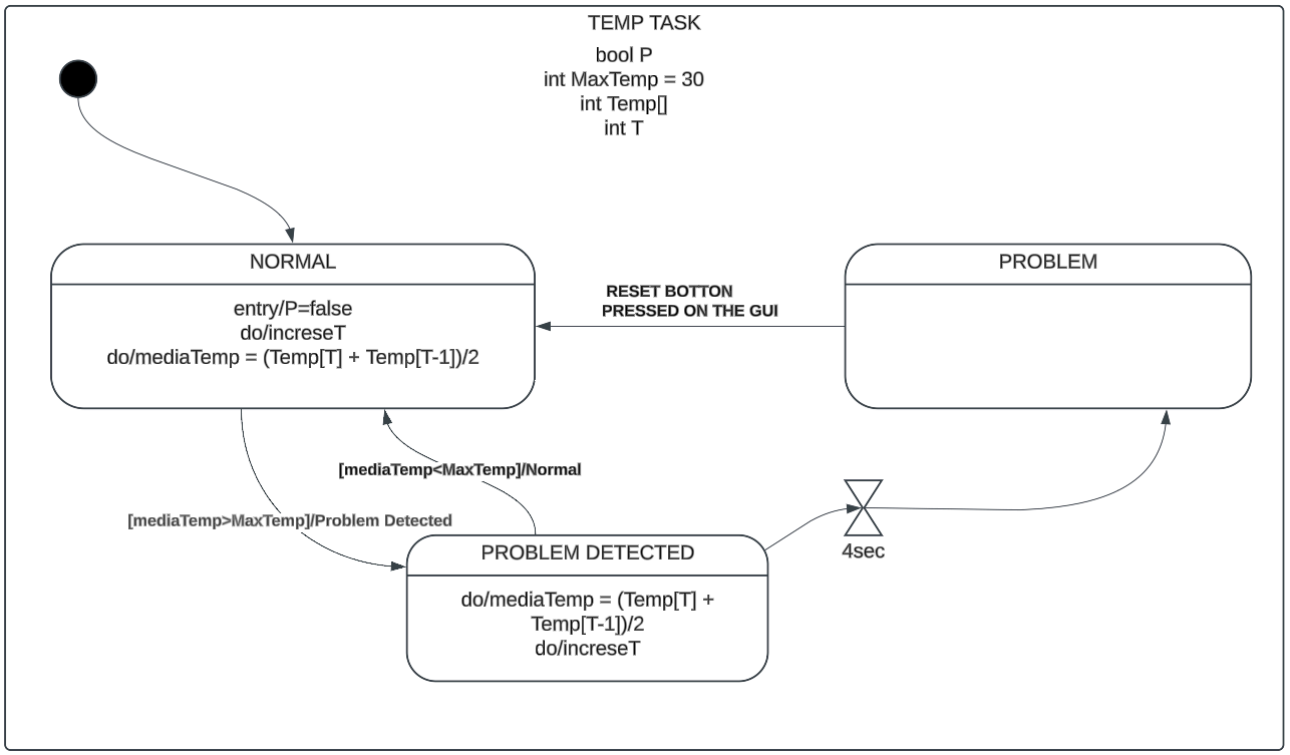
Utilizza i *device* **servoMotor** e **button**. Gestisce l'apertura e la chiusura della porta del contenitore in condizioni di normalità. In caso di problemi (**FULL** o **PROBLEM**) o di *sleep*, i pulsanti smettono di funzionare fino al ritorno allo stato **AWAKE**.

DetectorTask



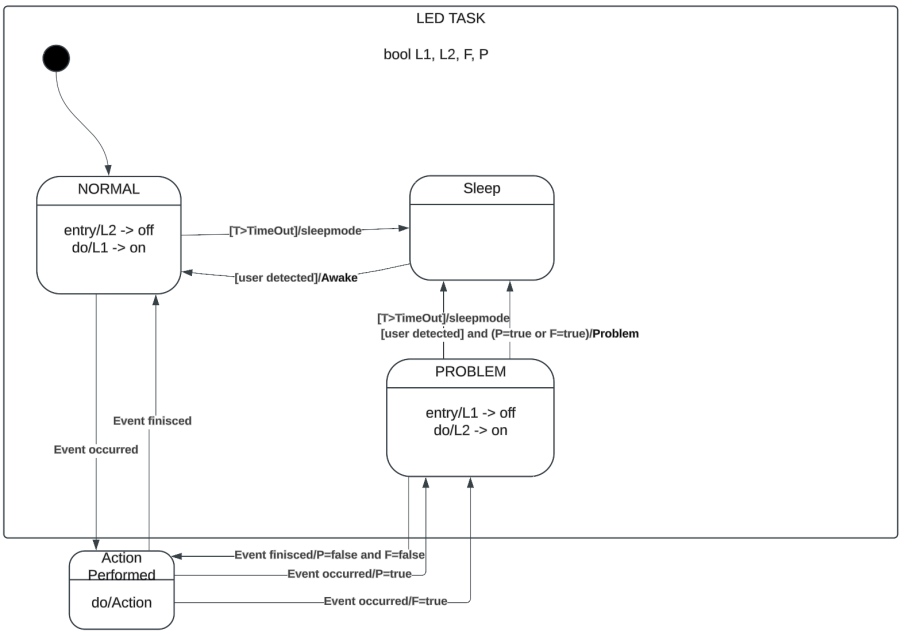
Usa il *device* **sonar** per controllare costantemente il livello di riempimento (escluso lo stato *sleep*). Raggiunto un certo livello, passa allo stato **FULL** e chiude il contenitore se aperto. Riceve il messaggio *EMPTY* dall'app Java per tornare allo stato normale.

TempTask



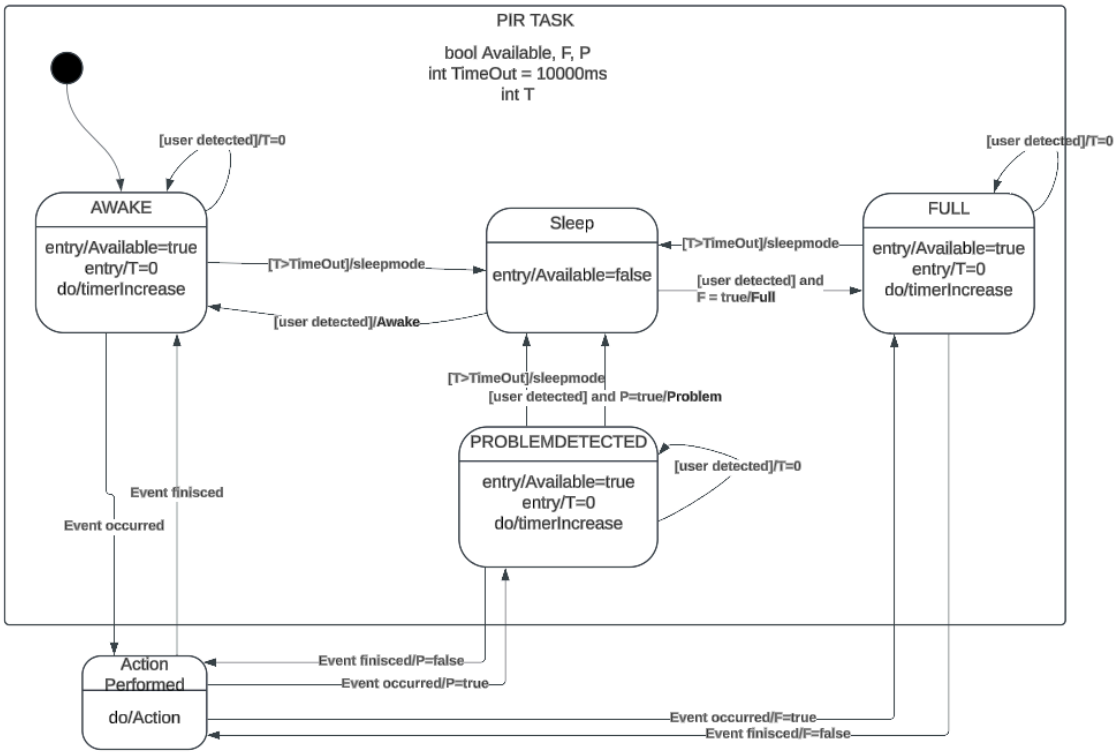
Simile al *DetectorTask*, utilizza il sensore di temperatura. Se la media delle temperature rilevate supera una soglia per più di 4 secondi, entra in **PROBLEM DETECTED**. Ritorna allo stato normale ricevendo il messaggio *RESTORE* dall'app Java.

LedTask



Usa due *device* LED. Il LED verde resta acceso in tutti gli stati eccetto **FULL** e **PROBLEM**, dove si accende il LED rosso. Questo task resta attivo anche in *sleep*.

PIR Task



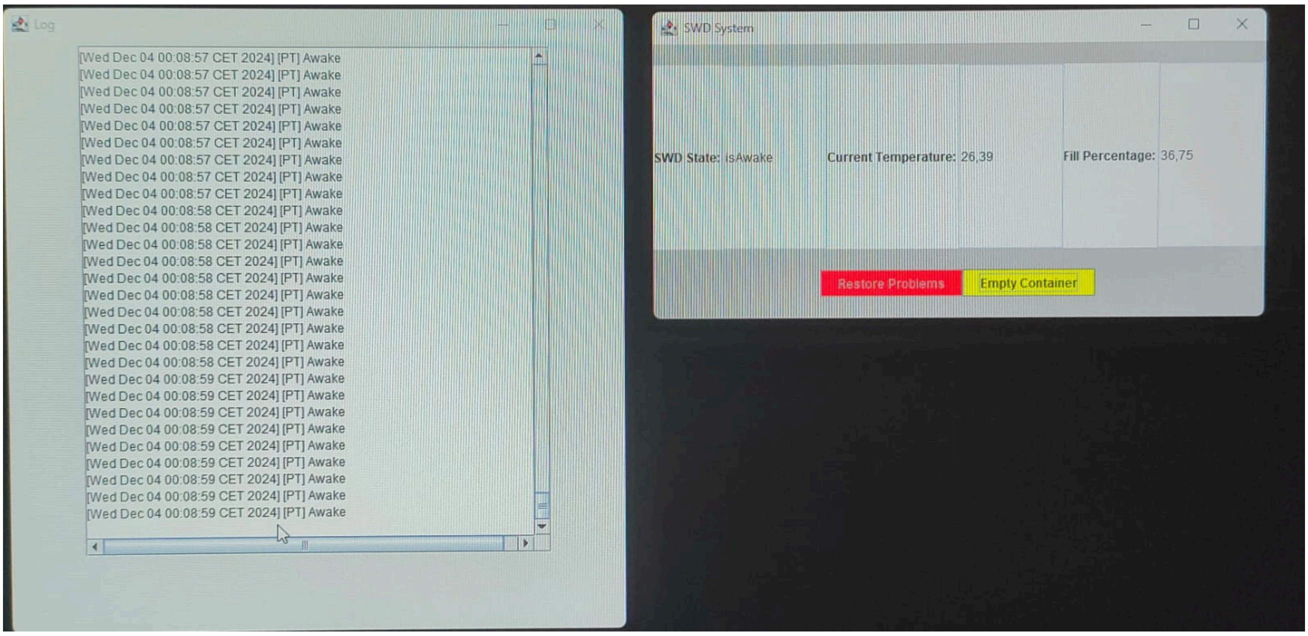
Utilizza il *device* **PIR** per mandare in *sleep* il sistema dopo 10 secondi di inattività negli stati **AWAKE**, **FULL** e **PROBLEM DETECTED**. All'uscita dallo *sleep*, riporta il sistema allo stato precedente.

Communication Task

Implementa la classe **MSGService** per inviare messaggi in formato stringa all'app Java. Ha due stati:

1. Attesa di 0,5 secondi.
2. Invio di informazioni sulla macchina, temperatura e livello di riempimento nel formato:
   `cw:st:"statusMessage":"wasteLevel":"temperature"`

Applicazione Java



Lanciando prima il programma Arduino, si avvia poi l'applicazione Java che apre due finestre:

- **Finestra Log**: mostra i messaggi dei task quando cambiano stato o resettano i timer (es. *PIR Task*).
- **Finestra Sistema**: mostra lo stato del sistema, la temperatura attuale e la percentuale di riempimento.

Gli stati visualizzabili sono:

- "Awake"
- "Full"
- "Problem"
- "InSleep"
- "Open"
- "Close"
- "Emptying"
- "Restore"

I pulsanti saranno attivi solo negli stati **FULL** e **PROBLEM**.

Link al repository GitHub: [Assignment-02 Repository]