Relazione di Progetto: Smart Drone Hangar

Assignment 2 - IoT Systems

Karim El Berni Luca Fabbri Luca Dellasantina

11 febbraio 2026

Indice

1 Introduzione

Il progetto **Smart Drone Hangar** consiste nella realizzazione di un sistema IoT per la gestione automatizzata di un hangar per droni. Il sistema è composto da due sottosistemi principali che comunicano tramite protocollo seriale:

- **Drone Hangar (Arduino):** L'unità fisica che gestisce sensori (temperatura, distanza, presenza) e attuatori (motori, LED, display LCD) per controllare l'accesso e lo stato dell'hangar.
- **Drone Remote Unit (Java/PC):** Un'interfaccia grafica (GUI) che funge da dashboard di controllo remoto, permettendo all'operatore di inviare comandi e monitorare la telemetria in tempo reale.

2 Architettura del Sistema

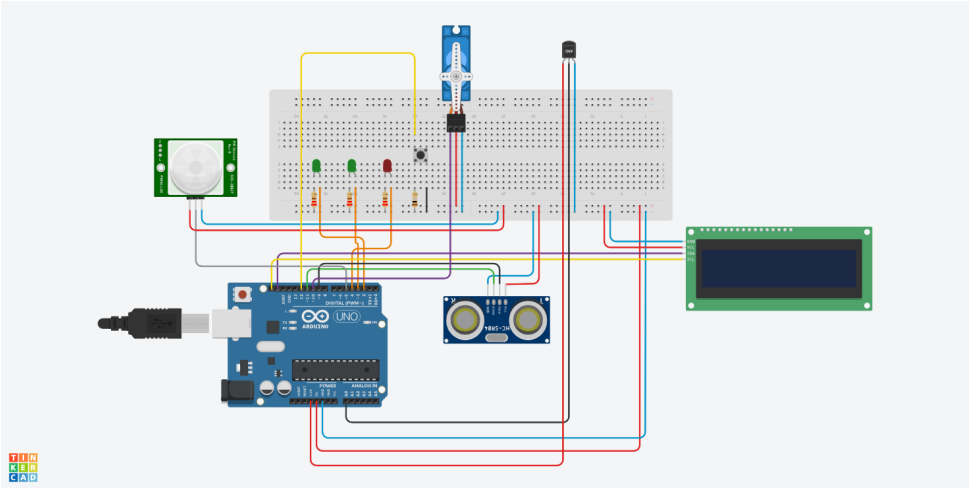

Figura 1: Schema del circuito su Breadboard

2.1 Hardware - Lato Arduino

Il sottosistema embedded è basato su Arduino Uno e utilizza i seguenti componenti collegati ai pin specificati:

2.2 Software - Lato PC

Il lato PC è un'applicazione Java basata su **Swing** per l'interfaccia grafica. La comunicazione seriale è gestita tramite la libreria `jSerialComm`, che permette l'invio asincrono di comandi e la ricezione di eventi.

Componente	Pin Arduino	Funzione
Led L1 (Verde)	D2	Stato "Power On" / "Rest"
Led L2 (Verde)	D3	Indicatore di movimento (Blinking)
Led L3 (Rosso)	D4	Indicatore di Allarme
Servo Motore	D5	Apertura/Chiusura portellone
Sonar (HC-SR04)	D9 (Trig), D10 (Echo)	Misurazione distanza drone
Sensore PIR	D11	Rilevamento presenza drone esterno
Pulsante	D12	Reset manuale allarme
Sensore Temp (TMP36)	A0	Monitoraggio surriscaldamento
Display LCD (I2C)	SDA/SCL	Feedback visuale all'utente

Tabella 1: Mappa dei collegamenti hardware

3 Implementazione Firmware (Arduino)

3.1 Architettura a Task

Il firmware è stato progettato seguendo un pattern basato su uno **Scheduler Cooperativo**. Invece di utilizzare un unico ciclo `loop()` bloccante o complesso, il sistema è diviso in task indipendenti:

- **Scheduler:** Gestisce l'esecuzione periodica dei task.
- **SmartHangarTask:** Implementa la logica principale e la macchina a stati finiti (FSM).
- **BlinkTask:** Gestisce il lampeggio dei LED in modo non bloccante.
- **TempMonitorTask:** Monitora la temperatura e innesca gli allarmi.

3.2 Macchina a Stati Finiti (FSM)

Il cuore del sistema è la classe `SmartHangarTask`, che evolve attraverso i seguenti stati:

1. **REST:** Il sistema è a riposo, il drone è dentro. Il servo è chiuso (0°). Attende il comando seriale "TO".
2. **TAKE_OFF:** Il servo si apre (90°). Il sistema attende che il Sonar rilevi che il drone è uscito ($distanza > D1$).
3. **TO_OUT:** Il drone è fuori e il servo si richiude. Attende il comando "LD" E il rilevamento del PIR.
4. **LANDING:** Il servo si riapre. Attende che il Sonar rilevi il rientro del drone ($distanza < D2$).
5. **ALARM:** Stato di emergenza attivato da temperatura elevata. Blocca le operazioni finché non viene premuto il pulsante di Reset.

Di seguito un esempio del codice della FSM nel metodo `tick()`:

```
void SmartHangarTask::tick() {
  if (state == REST) {
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      if (cmd == "TO" && !preAlarm) {
        state = TAKE_OFF;
        servo->setPosition(90);
        // ...
      }
    }
  } else if (state == TAKE_OFF) {
    if (sonar->getDistance() > D1) {
        // Logica transizione a TO_OUT
    }
  }
  // Altri stati...
}
```

3.3 Monitoraggio Temperatura

Il task `TempMonitorTask` legge periodicamente il sensore TMP36. Gestisce due soglie:

- **Temp1:** Attiva un flag di pre-allarme.
- **Temp2:** Se superata per un tempo $T4$, porta il sistema nello stato globale di **ALARM**, accendendo il Led L3 e bloccando le operazioni.

4 Implementazione Software (Java)

La **Drone Remote Unit** fornisce una GUI semplice con:

- Due pulsanti per i comandi: *Take Off* e *Land*.
- Label per visualizzare lo stato corrente, la distanza rilevata e la temperatura.

La classe `SerialComm` gestisce la comunicazione su un thread separato per non bloccare l'interfaccia utente (EDT).

```
// Esempio di gestione eventi seriali in Java
serialPort.addDataListener(new SerialPortDataListener() {
   @Override
   public void serialEvent(SerialPortEvent event) {
       // Lettura dati e aggiornamento GUI tramite SwingUtilities.
   invokeLater
   }
});
```

5 Protocollo di Comunicazione

Arduino e PC comunicano tramite messaggi ASCII terminati da `\n` a 9600 baud.

Direzione	Messaggio	Significato
PC → Arduino	`TO`	Richiesta di decollo (Take Off)
PC → Arduino	`LD`	Richiesta di atterraggio (Land)
Arduino → PC	`TAKE OFF`	Conferma inizio decollo
Arduino → PC	`DRONE OUT`	Drone uscito dall'hangar
Arduino → PC	`LANDING`	Inizio procedura atterraggio
Arduino → PC	`DRONE INSIDE`	Drone rientrato (Stato REST)
Arduino → PC	`ALARM`	Allarme temperatura critica
Arduino → PC	`TEMP:xx.x`	Telemetria Temperatura

Tabella 2: Protocollo Seriale

6 Conclusioni

Il progetto dimostra l'integrazione efficace tra un sistema embedded real-time (Arduino con scheduler cooperativo) e un sistema di controllo desktop (Java). L'uso di una macchina a stati finiti garantisce robustezza nella gestione logica dell'hangar, prevenendo stati inconsistenti, mentre il monitoraggio della temperatura aggiunge un livello di sicurezza fondamentale per un sistema IoT.