ALMA MATER STUDIORUM · UNIVERSITY OF BOLOGNA

Degree Programme in Engineering and Computer Science
A.A. 2025/26

Smart Drone Hangar

Grazia Bochdanovits de Kavna
matr. 0001117082

Alessandro Rebosio
matr. 0001130557

Contents

Chapter 1

Analysis

1.1 Description and Requirements

Initially, the hangar door (HD) is closed and the system is in the `DRONE INSIDE` state. LED L1 is on, LEDs L2 and L3 are off, and the LCD displays "`DRONE INSIDE`".

Take-off Sequence. When the drone requests take-off through the Drone Remote Unit (DRU), the hangar door opens and the LCD displays "`TAKE OFF`". The system then waits for the drone to exit the hangar. Exit is detected by the distance sensor (DDD): when the measured distance exceeds D_1 for more than T_1, the drone is considered outside. At that moment, the door closes automatically and the LCD displays "`DRONE OUT`".

Landing Sequence. When the drone requests landing via the DRU, the presence sensor (DPD) detects its approach. The hangar door opens and the LCD displays "`LANDING`". The landing is confirmed when the DDD measures a distance below D_2 for longer than T_2. Once landed, the door closes and the LCD returns to "`DRONE INSIDE`".

Indicators and Lighting. During take-off and landing operations, LED L2 blinks with a 0.5 s period to indicate activity. In all other states it remains off.

Temperature Monitoring and Safety Conditions. Temperature control is active whenever the drone is inside the hangar, regardless of the current operation. If the internal temperature remains above `Temp1` for more than `T3`, the system enters a **pre-alarm** condition: new take-offs or landings are suspended, while any operation already in progress may complete. If the temperature rises above `Temp2` (with `Temp2 > Temp1`) for longer than `T4`, the system switches to **alarm** mode. In this state, the hangar door is closed (if open), LED L3 turns on, and the LCD shows "`ALARM`". If the drone is outside during an alarm, an "`ALARM`" message is sent via the DRU. All operations remain suspended until the **RESET** button is pressed, which restores the system to normal operation.

Configurable Parameters and GUI Requirements. The parameters D_1, D_2, T_1, T_2, T_3, T_4, `Temp1`, and `Temp2` are configurable for testing and calibration.

The DRU graphical interface must provide the following functions:

- Sending take-off and landing commands to simulate drone actions;

- Displaying the drone state (*rest*, *taking off*, *operating*, *landing*);
- Displaying the hangar state (*normal* or *alarm*);
- Showing the current distance from the ground during landing.

1.2 Wiring

Below is a concise list of the essential hardware components required to build and test the Smart Drone Hangar. Quantities, components and brief notes on their function are provided to assist with procurement and wiring.

Qty	Component	Notes
1	Microcontroller	Handles sensors, actuators, and serial communication
1	Servo motor	Used as the hangar door actuator
1	Distance sensor (DDD)	Measures the drone's distance inside the hangar
1	Presence sensor (DPD)	Detects the drone approaching the hangar
1	Temperature sensor	Monitors internal hangar temperature
3	LEDs (L1, L2, L3)	System status indicators (normal, activity, alarm)
3	Resistors 220 Ω	Current-limiting resistors for LEDs L1, L2, L3
1	LCD display	Displays system messages
1	RESET button	Used to clear alarms and restore normal operation
1	Resistor 10 kΩ	Pull-up / pull-down resistor for the RESET button

Table 1.1: Essential hardware list

Figure 1.1 shows the wiring diagram of the hangar control unit, highlighting the main connections between the microcontroller, sensors, actuators, and indicators.

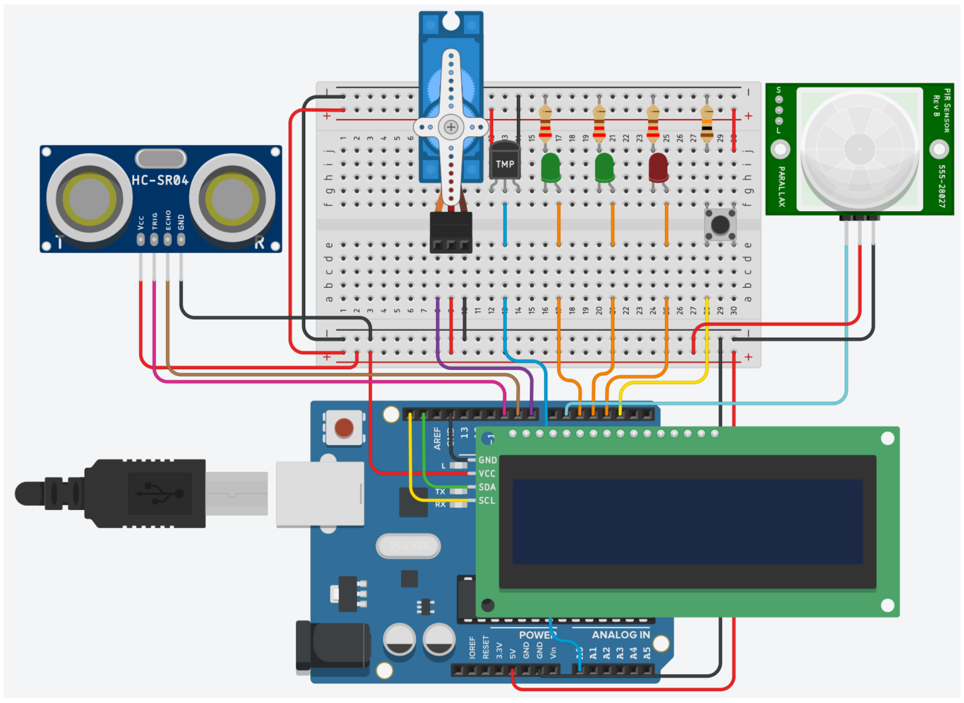


Figure 1.1: Wiring diagram of the hangar control unit.

Chapter 2

Architecture

The system architecture is organized around a set of concurrent tasks, each responsible for a specific function of the Smart Drone Hangar. A lightweight cooperative scheduler periodically activates these tasks in a cyclic order. Although executed sequentially, their high activation rate allows the system to behave as if all processes were running concurrently.

All tasks interact through a shared data structure called the *Context*, which maintains the global state of the hangar — including drone position, door status, temperature conditions, and system flags. The *Context* serves as a shared memory that enables coordination among components via Boolean variables and message codes, ensuring modularity and preventing direct coupling between tasks.

The main functional components are:

- **System Task** – manages the overall operating mode (`NORMAL`, `PREALARM`, `ALARM`);
- **Flight Task** – coordinates flight operations and activates the other control tasks;
- **Takeoff Task** and **Landing Task** – specialized observer tasks managing their respective flight phases;
- **Gate Task** – controls the hangar door actuator;
- **Blink Task** – manages LED blinking during active operations;
- **Observer Task** – monitors sensors and serial messages, updating the *Context*.

2.1 System Task

The **SystemTask** manages the global operating mode of the hangar, switching between three states: `NORMAL`, `PREALARM`, and `ALARM`.

In `NORMAL`, all flight operations are allowed. If the temperature exceeds `TEMP1` for longer than `T1`, the system enters `PREALARM`, suspending new take-offs and landings but allowing ongoing ones to complete. If the temperature remains above `TEMP2` for more than `T2`, the system transitions to `ALARM`: the hangar door is closed, the alarm LED (L3) is activated, and all operations are halted. The system remains in this state until the **RESET** button is pressed, which returns it to `NORMAL`. If the drone is outside during an alarm, an alert message is sent via the DRU.

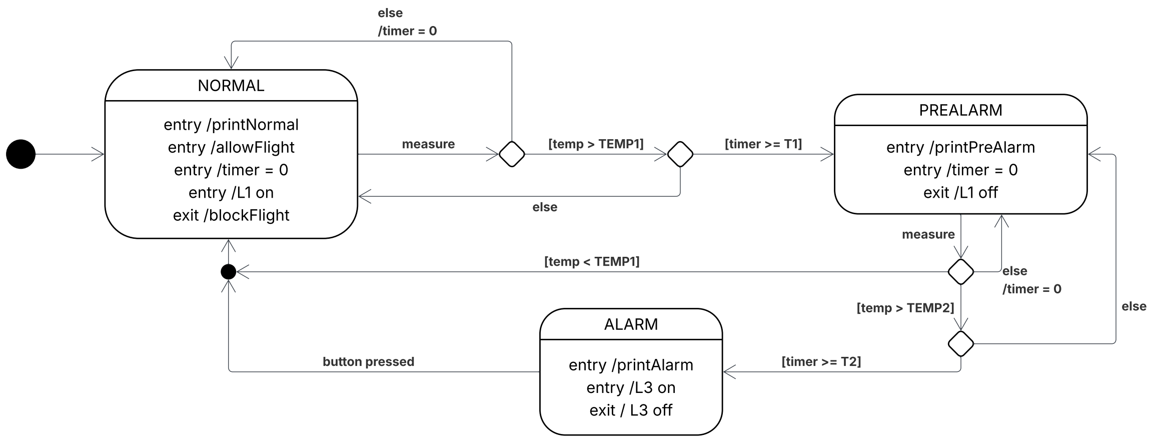


Figure 2.1: System Task state diagram.

When the entry action `/AllowFlight` is executed, a Boolean flag is set to `true`; conversely, the exit action `/BlockFlight` sets it to `false`. This flag informs other tasks whether flight operations are currently permitted.

2.2 Flight Task

The **FlightTask** coordinates the take-off and landing sequences, acting as the intermediate layer between the high-level system logic and the low-level actuator control. It operates through three main states: `IDLE`, `WAITING`, and `OPERATING`.

At system startup, the FlightTask determines its initial state based on Context flags: if all operations are complete, it starts in `IDLE`; otherwise, it initializes directly in `OPERATING`. This mechanism allows the hangar to resume ongoing operations after a power loss or reset, ensuring consistency between software logic and physical state.

During normal operation, when a take-off or landing command is received, the task activates the corresponding sequence by updating the Context. The hangar door opens and LED L2 begins blinking. Once sensors confirm completion — e.g., the drone has exited or landed — the door closes, blinking stops, and the system returns to `IDLE`.

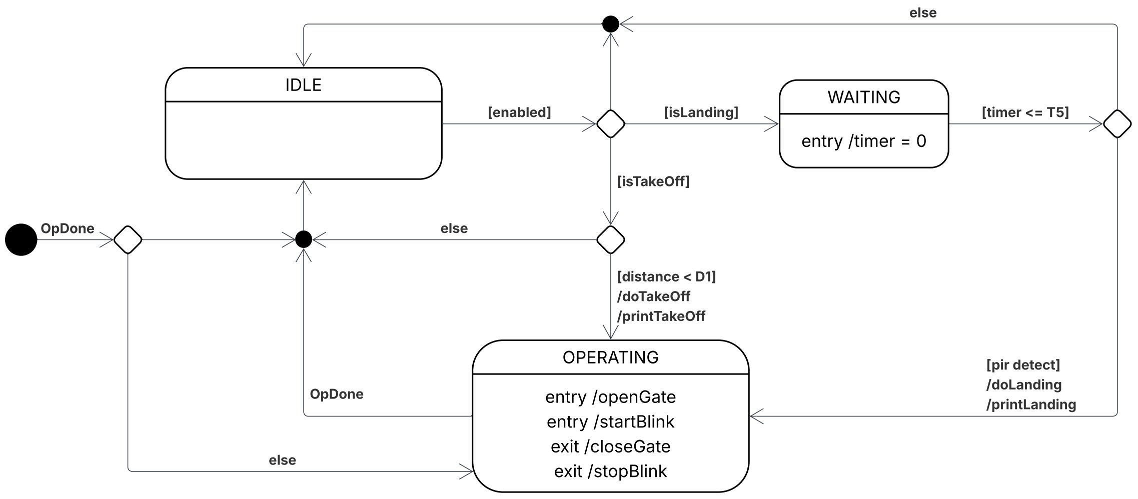


Figure 2.2: Flight Task state diagram.

When entering the `OPERATING` state, the FlightTask triggers the `/startBlink` and `/openGate` actions, enabling the corresponding tasks. Upon completion of the operation, it executes `/stopBlink` and `/closeGate`, restoring the system to its resting configuration.

2.3 Takeoff Task

The **TakeoffTask** supervises the drone's departure from the hangar. It begins its operation when enabled by the `FlightTask` and enters the `MEASURING` state, where it continuously monitors the distance measured by the internal sensor. When the detected distance remains greater than the threshold `D1` for longer than the configured time `T3`, the drone is assumed to have left the hangar completely. At this point, the task returns to the `IDLE` state, updates the `Context` by printing the `DRONE OUTSIDE` message on the LCD, and signals the completion of the take-off sequence. If the distance drops below the threshold before the timer expires, the counter is reset, ensuring that only stable readings are considered valid. This simple timing-based mechanism filters out sensor noise and guarantees reliable detection of the drone's exit.

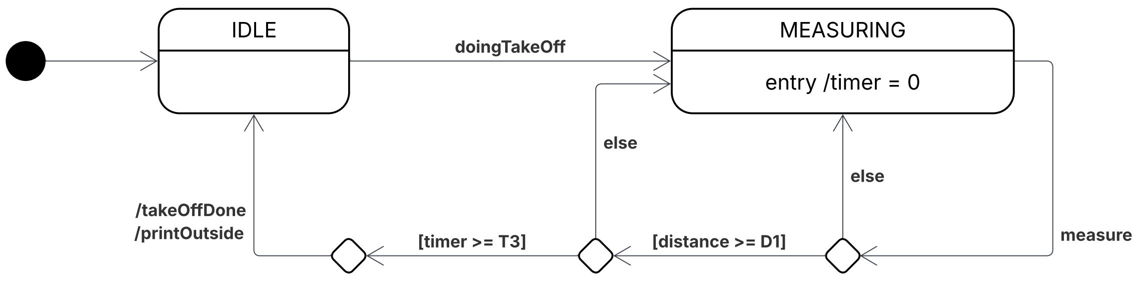


Figure 2.3: Takeoff Task state diagram.

2.4 Landing Task

The **LandingTask** operates similarly but monitors the approach of the drone during landing. Once activated, it enters the `MEASURING` state and evaluates the distance between the drone and the ground using the same ultrasonic sensor. When this distance remains below the threshold `D2` for a duration longer than `T4`, the system concludes that the drone has safely landed inside the hangar. The task then switches back to `IDLE`, updates the `Context` by printing `DRONE INSIDE`, and notifies the completion of the landing sequence. As in the take-off process, transient or unstable readings reset the internal timer, ensuring that the transition to the landed state occurs only when the condition is consistently verified. This approach provides a robust and noise-tolerant mechanism for landing detection.

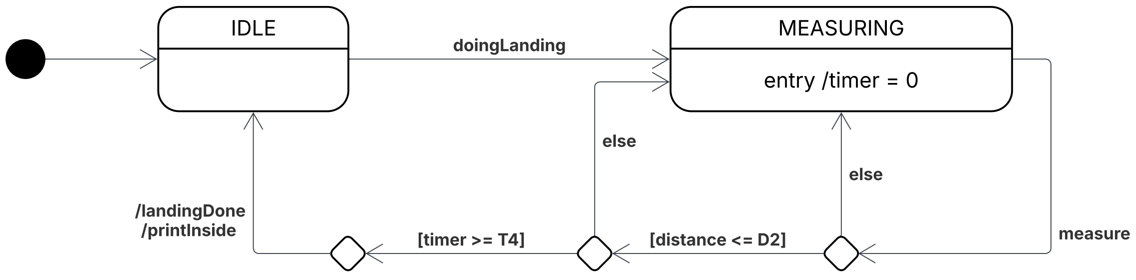


Figure 2.4: Landing Task state diagram.

2.5 Gate Task

The **GateTask** controls the hangar door using four states: `CLOSE`, `OPENING`, `OPEN`, and `CLOSING`. At startup, the gate is in `CLOSE`. When enabled, it transitions to `OPENING` until the door is fully open, then to `OPEN`. If the enable signal becomes false, it moves to `CLOSING` until the door is fully closed. If a new command arrives while the door is moving, the task immediately reverses its direction.

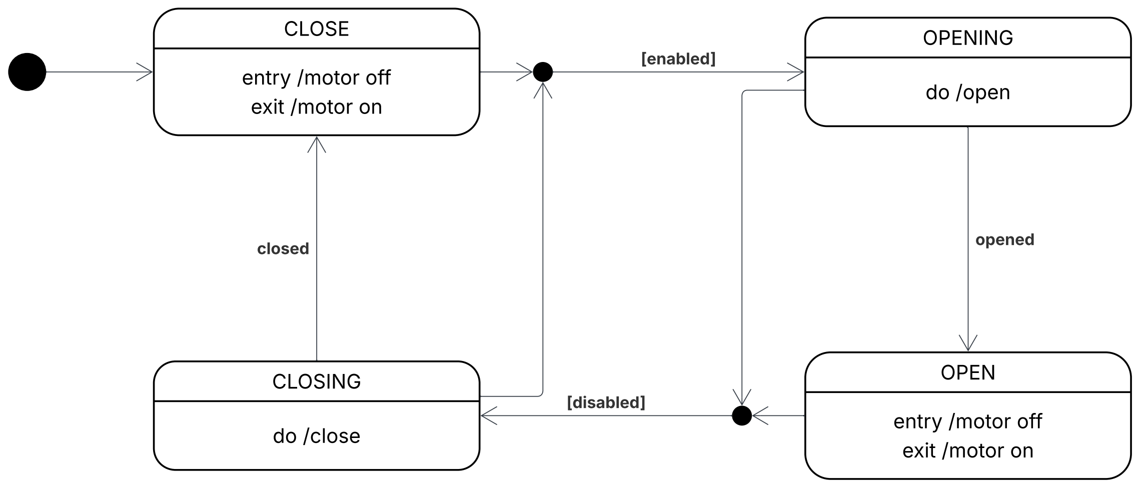


Figure 2.5: Gate Task state diagram.

2.6 Blink Task

The **BlinkTask** handles LED L2 blinking, providing visual feedback during active flight phases. It operates in two alternating states, `ON` and `OFF`. At each activation, the task toggles the LED state; however, a guard condition prevents the transition from `OFF` to `ON` when blinking is disabled, keeping the LED off during idle periods.

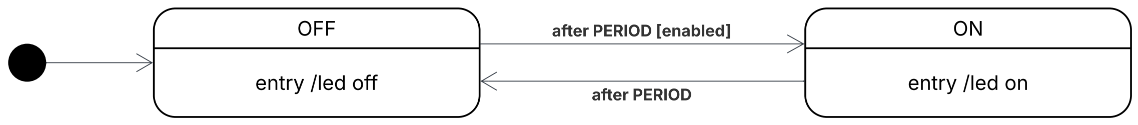


Figure 2.6: Blink Task state diagram.

2.7 Observer Task

The **ObserverTask** periodically monitors sensors and communication channels by evaluating a predicate function at each activation. When the predicate evaluates to true, the associated callback is executed, allowing the system to react immediately to specific events such as sensor updates or incoming commands. This provides a lightweight mechanism for asynchronous observation without interfering with the main control logic.

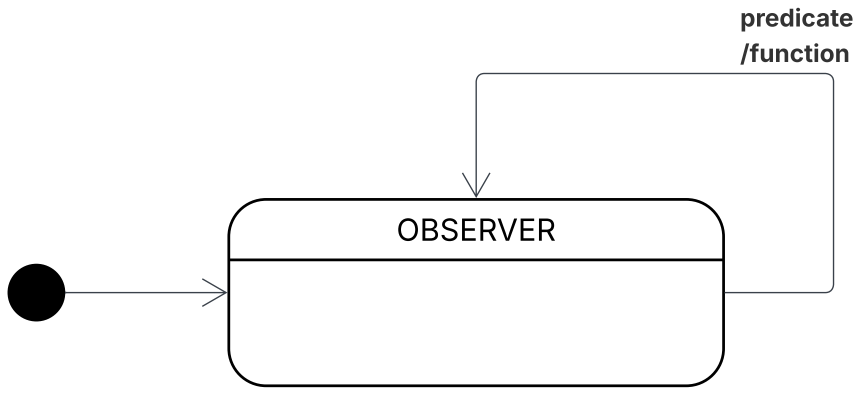


Figure 2.7: Observer Task state diagram.

The Observer acts as a central listener: it periodically reads inputs and messages (e.g., from the DRU via serial communication) and dispatches relevant information to the appropriate tasks — System, Flight, Gate, etc. This approach keeps each functional task lightweight and focused on its own logic. Moreover, the architecture supports multiple SystemTask instances, each with its own Context, enabling a single microcontroller to manage multiple hangars in parallel and simplifying message routing and scalability.

Chapter 3

Arduino Workflow

3.1 Overview

This chapter describes the dynamic behaviour of the Smart Drone Hangar once deployed on the Arduino platform. Each system function is implemented as an independent task, periodically activated by a cooperative scheduler. Although tasks are executed sequentially, their rapid activation makes the system react as if all processes were concurrent, continuously responding to inputs and updating outputs in real time.

All components share a common data structure, the **Context**, which represents the global state of the hangar. Through this structure, tasks communicate indirectly by setting and reading Boolean flags that describe the hangar's condition, the drone's position, and the current operating mode.

3.2 System Dynamics

At startup the system initializes all hardware components, ensuring a safe condition: the hangar door is closed, the drone is detected inside, and flights are temporarily disabled until normal parameters are restored. The scheduler then maintains continuous control by cyclically activating all tasks to supervise sensors, actuators, and communication interfaces.

The **SystemTask** governs the overall behaviour of the hangar. It monitors temperature values and selects the appropriate operating mode. Under normal conditions flights are permitted. If the temperature exceeds the first threshold (`Temp1`) for longer than `T1`, the system enters a pre-alarm state, suspending new take-offs or landings while allowing those in progress to finish. If it surpasses the higher threshold (`Temp2`) for more than `T2`, the system switches to alarm mode: the door closes, the alarm indicator turns on, and all operations remain halted until the **RESET** button is pressed. This mechanism ensures that flight activity occurs only under safe environmental conditions.

3.3 Coordination of Operations

When the system starts, the **FlightTask** determines its initial state from the Context. If all operations are complete, it begins in `IDLE`; otherwise, it starts in `OPERATING`. This allows automatic recovery after a power loss or reset, resuming any sequence that was already in progress.

During operation, the FlightTask monitors Context flags to follow the progress of take-off or landing. It stays in `OPERATING` while subordinate actions—such as gate movement or landing detection—are active, and returns to `IDLE` only when all are completed. This guarantees synchronization between software logic and the physical hangar.

3.4 Supporting Tasks and Context Interaction

Several supporting tasks ensure system responsiveness:

- The **GateTask** drives the servo motor to open or close the hangar door.
- The **BlinkTask** controls LEDs, providing visual feedback during flight operations.
- The **TakeoffTask** and **LandingTask** monitor distance sensors to detect when the drone has left or entered the hangar.
- The **ObserverTask** reads sensors and serial messages, updating the Context with temperature, distance, and command data.

All tasks interact exclusively through the Context, which acts as shared memory and coordination channel. For example, the SystemTask may block new flights by setting a flag interpreted by the FlightTask, while the ObserverTask records a command that will be processed in the next cycle.

3.5 Global Workflow

The overall operation arises from the cooperation of all tasks under the scheduler. The ObserverTask updates the Context with real-time data, the SystemTask enforces safety conditions, and the FlightTask coordinates flight sequences by activating the Gate and Blink tasks and delegating monitoring to Takeoff or Landing. When the sequence concludes, the system automatically returns to standby, ready for a new command. Through this cooperative design, the Smart Drone Hangar achieves safe, modular and deterministic behaviour, maintaining continuous operation even after temporary interruptions or environmental changes.

Appendix A

User Interface

The user interface of the application is organized into three main panels, each serving a distinct purpose within the system.

A.1 Connection Panel

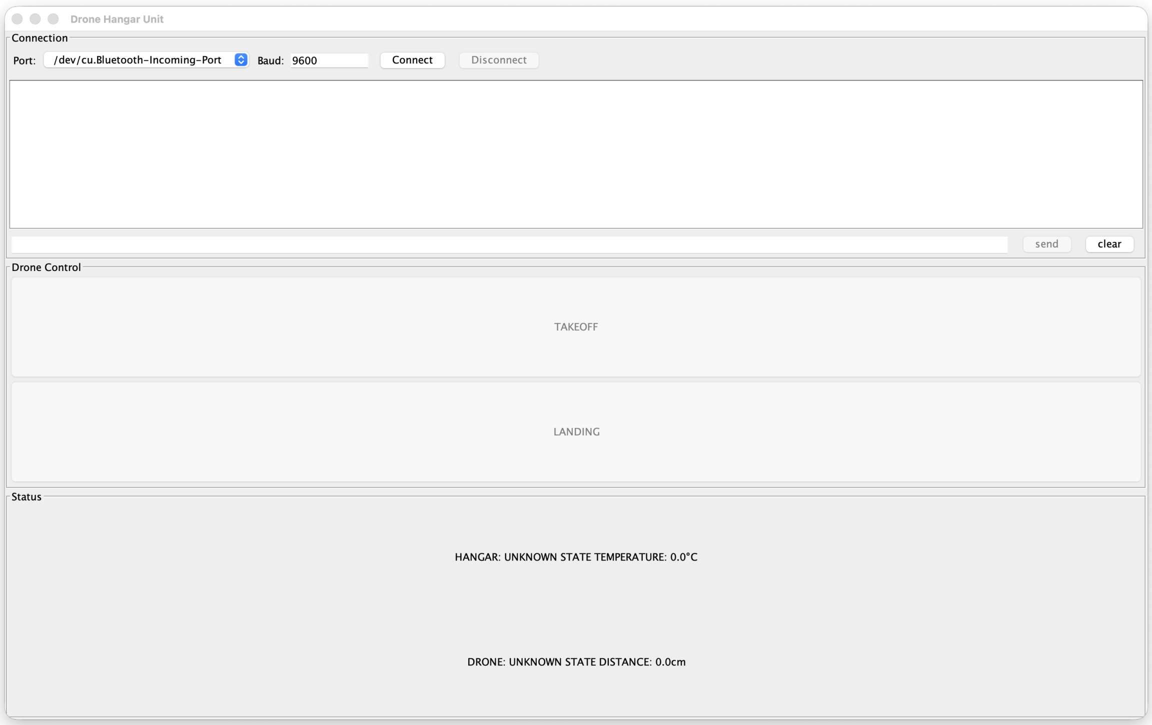
The upper part of the view is dedicated to managing the connection with the serial port. The user can select the desired port, configure the communication parameters, and establish or terminate the connection with the device.

A.2 Control Panel

The central part of the interface provides simple controls for operating the drone. It includes two buttons — *Takeoff* and *Landing* — which send the corresponding commands to the controller. These buttons are automatically enabled or disabled based on the current connection status and system state.

A.3 Status Panel

The lower part of the view displays real-time information about the *Hangar* and *Drone* states, allowing continuous monitoring of the overall system status.



HANGAR: UNKNOWN STATE TEMPERATURE: 0.0°C

DRONE: UNKNOWN STATE DISTANCE: 0.0cm

Figure A.1: App view

Appendix B

User Guide

B.1 Cloning the repository

Clone the project from GitHub and change to the project directory:

```
> git clone https://github.com/alessandrorebosio/ESIOT25.git
> cd ESIOT25/assignment-02
```

B.2 Connecting and starting the application

Connect the Arduino to your computer, identify the serial port, then start the application with:

```
> java -jar drone-hangar-unit-all.jar
```

Alternatively you can start the project with:

```
> ./gradlew run
```