SAPIENZA
UNIVERSITÀ DI ROMA

Ingegneria del Software

Professore: Enrico Tronci

Relazione sul Progetto
Drone8

Autori	*Matricola*
Patryk Jan Mulica	1986671
Gianluca Viviano	1988233
Felix Rehrl	1985504

Sommario

1 Descrizione generale

Il progetto prevede la realizzazione di un **sistema di sorveglianza** basato su una formazione di **droni**, incaricati di **monitorare un'area** predefinita. Ogni **drone** dispone di un'autonomia di volo di 30 minuti e richiede un tempo di ricarica variabile tra 2 e 3 ore, scelto casualmente ad ogni ricarica.

Le specifiche prevedono che ogni **drone** si sposti alla velocità di 30 km/h e che l'area da monitorare sia di dimensioni 6×6 km, con il centro di controllo e ricarica posizionato al centro dell'area stessa.

Il compito del centro di controllo è inviare istruzioni ai **droni** in modo che ogni punto dell'area sorvegliata sia controllato almeno ogni 5 minuti. Un punto è considerato verificato al tempo t se, in quel momento, almeno un **drone** si trova a una distanza inferiore a 10 metri da quel punto.

Per rendere il sistema più modulare e consentire la gestione di più centri di controllo, i dati relativi alla posizione dell'area, alla velocità dei **droni** e ai tempi di ricarica sono parametrizzati e personalizzabili per ogni istanza del progetto. Questo approccio modulare permette una maggiore flessibilità nell'implementazione e nell'adattamento del sistema alle specifiche esigenze di sorveglianza di diverse aree e contesti operativi.

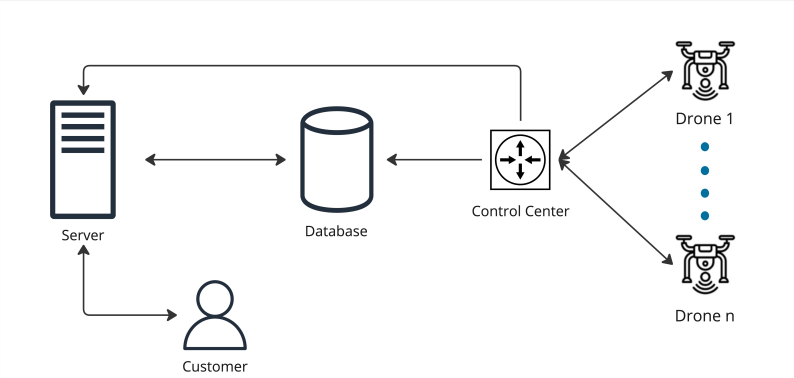

Figure 1: Ambiente operativo

2 User Requirements

Requisiti funzionali:

1. Il centro di controllo deve mandare istruzioni ai droni
2. La scansione di un punto deve scadere dopo t minuti (e.g. $t = 5\ minuti$)
3. I droni devono ricaricarsi in un tempo di ricarica uniformemente casuale nell'intervallo $[2h, 3h]$
4. I droni devono poter scansionare con un raggio di n metri (e.g. $n = 10m$)
5. L'utente deve poter decidere quando incominciare e finire la scansione
6. L'utente deve poter vedere l'attuale copertura dell'area
7. L'utente deve poter configurare il sistema (e.g. raggio di scansione dei droni, grandezza dell'area...)

Requisiti non funzionali:

1. La mappa una volta coperta al 100% deve mantenere una copertura del 100% finché il sistema è in funzione
2. I droni devono sempre tornare al Centro di Controllo
3. Il centro di controllo deve essere posizionato al centro dell'area.
4. La base di ricarica deve essere posizionato al centro dell'area.

2.1 UML degli use cases

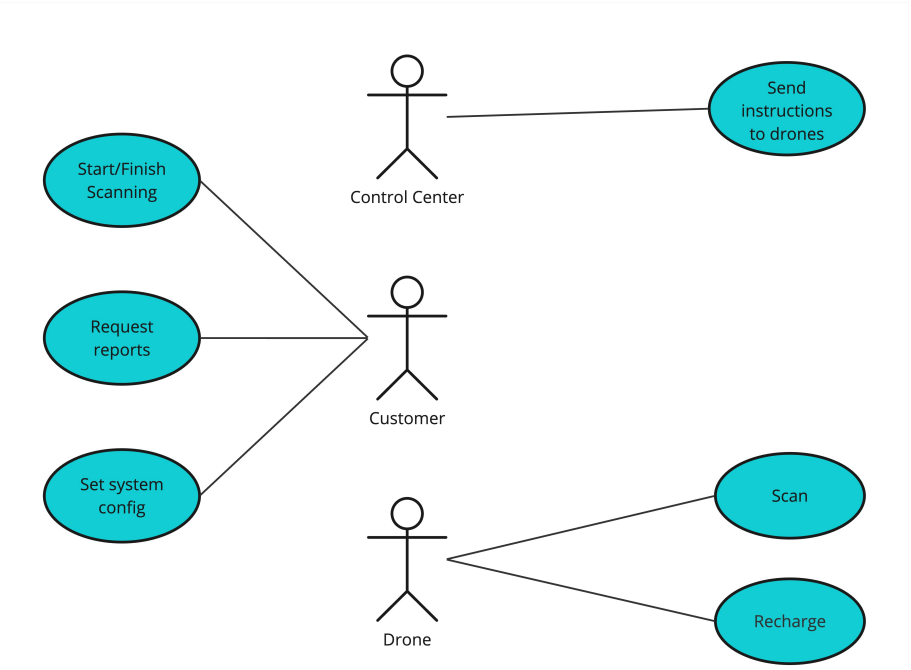

Figure 2: Diagramma UML degli use cases

3 System Requirements

Requisiti funzionali:

1. Il Centro di Controllo deve mandare i tragitti da far seguire ai droni
2. Il Centro di Controllo deve essere in ascolto per i dati dei droni
3. Il Centro di Controllo deve aggiornare un punto dell'area quando viene notificato dal drone che ci é passato
4. Il Centro di Controllo deve ricavare i tragitti da far eseguire ai droni
5. Il drone deve essere in ascolto per il Centro di Controllo
6. Il server deve poter mandare l'ultimo report ricevuto dal Centro di Controllo

Requisiti non funzionali:

1. Il drone e il Centro di Controllo comunicano tramite stream *Redis*
2. Non devono essere superate le 10.000 connessioni *Redis*
3. Il Centro di Controllo deve processare tutti i messaggi che gli vengono inviati entro 500ms
4. Le informazioni vengono salvate su un database PostgresSQL
5. Il server deve poter gestire le richieste del Centro di Controllo e dei client
6. La generazione del report deve essere fatta in meno di un secondo

3.1 Architettura del Sistema

Questo diagramma di architettura del sistema rappresenta i principali componenti del sistema, le loro interazioni e il flusso di dati tra di essi.

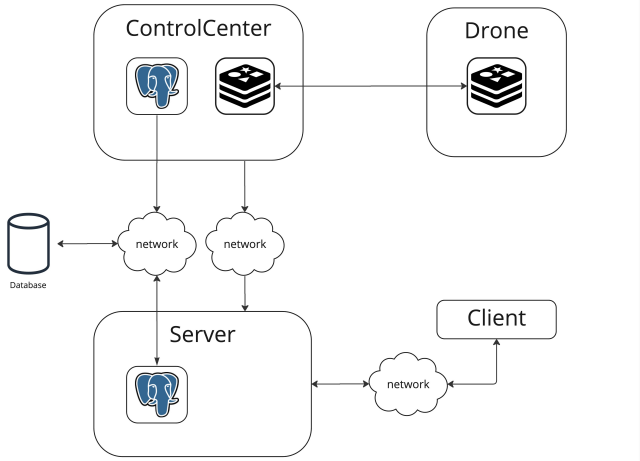

Figure 3: Diagramma dell'Architettura di Sistema

3.2 Activity Diagram

Questo activity diagram fornisce una rappresentazione dettagliata del flusso di lavoro dei droni durante la sorveglianza dell'area, dalla preparazione del volo alla scansione dell'area, fino al ritorno al centro di controllo e alla chiusura della missione.

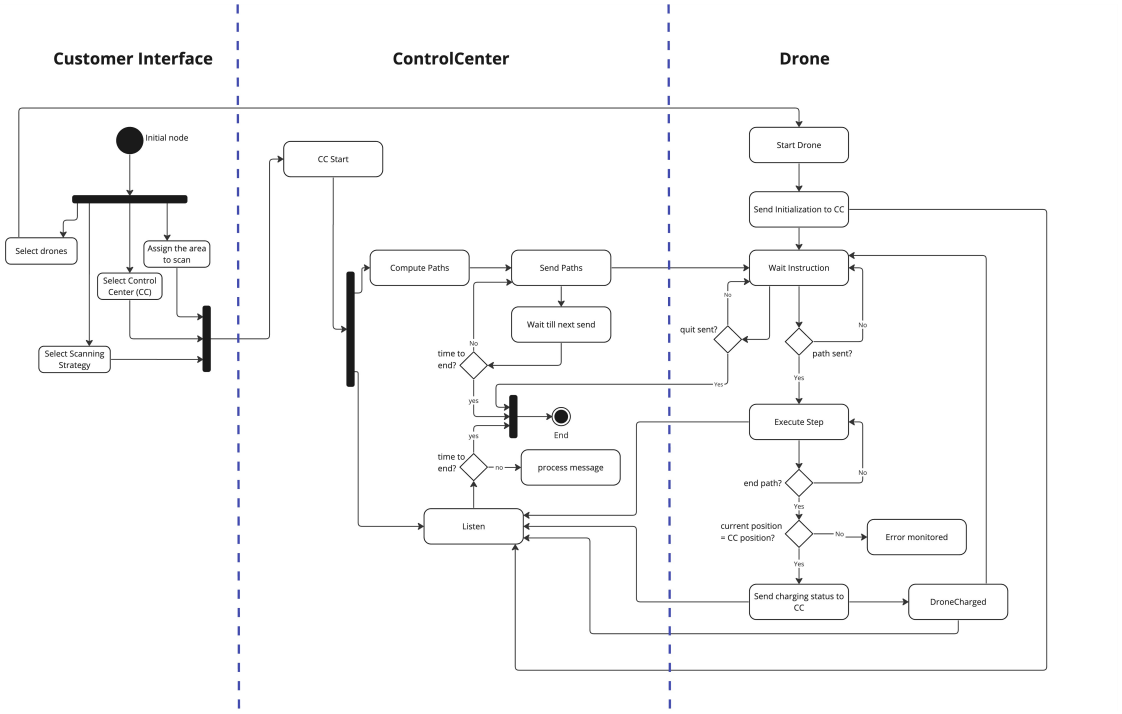

Figure 4: Activity Diagram of the Drones Scanning the Area

3.3 State Diagrams

3.3.1 Drone State Diagram

Questo diagramma rappresenta tutti i possibili stati operativi del drone, illustrando come il drone transiti tra i vari stati in risposta a diverse condizioni e comandi.

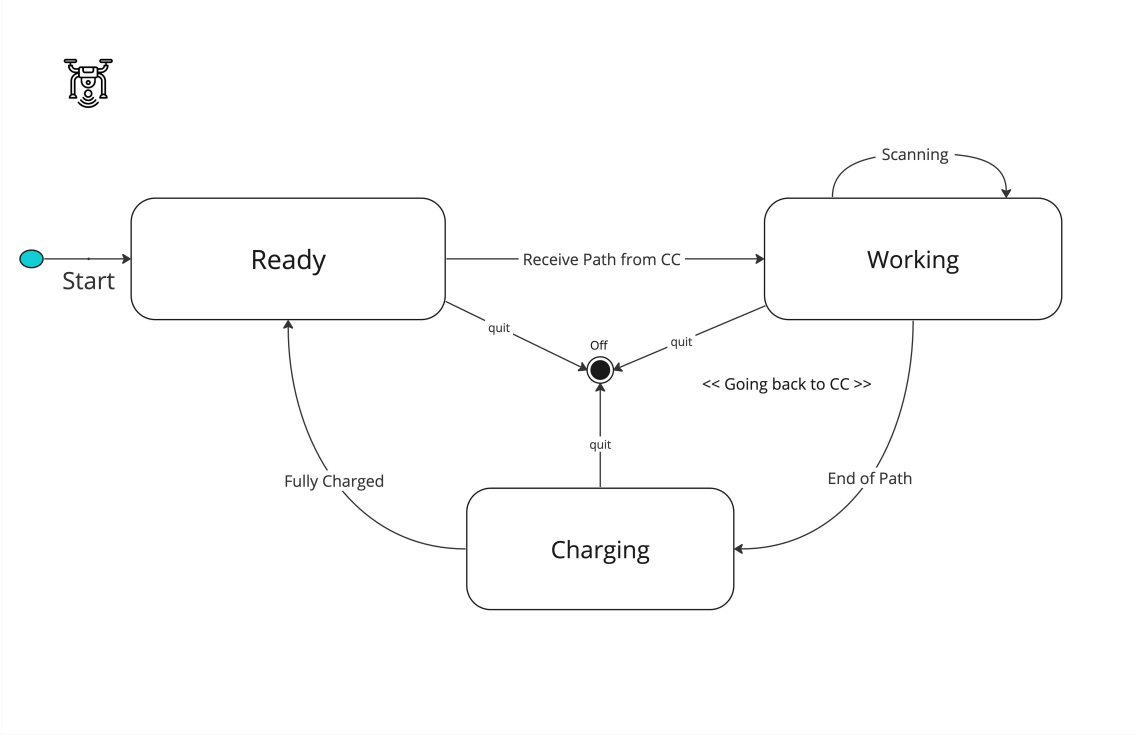

Figure 5: Drone State Diagram

3.3.2 ControlCenter State Diagram

Questo diagramma rappresenta tutti i possibili stati operativi del centro di controllo, illustrando come il centro di controllo transiti tra i vari stati in risposta a diverse condizioni e comandi.

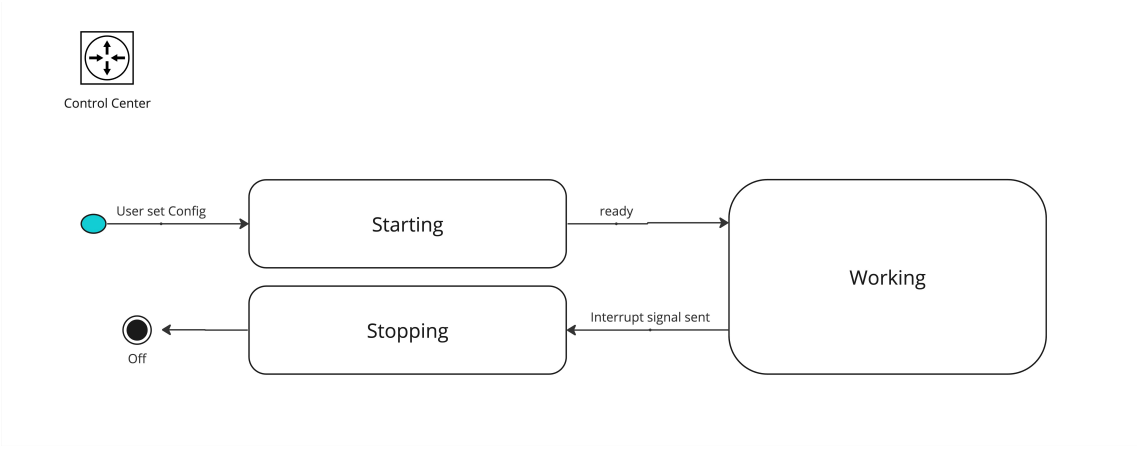

Figure 6: Control Center State Diagram

3.4 Message Sequence Chart

Questo diagramma rappresenta lo scambio di messaggi quando un cliente vuole ricevere un report generato automaticamente dal centro di controllo.

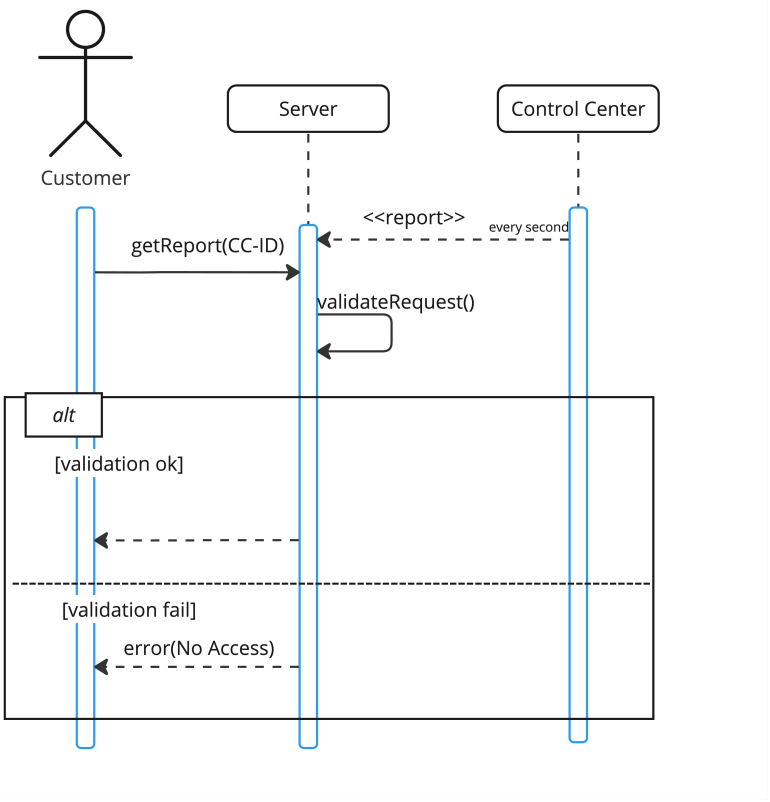

Figure 7: Message Sequence Chart for report request

4 Implementation

4.1 Descrizione generale

Il **Control Center** si occupa di orchestrare l'intera operatività dei **droni**. Quando riceve un messaggio attraverso lo **stream Redis**, innesca immediatamente l'inizializzazione dei **droni**, impartendo loro istruzioni su percorsi predefiniti in base a una strategia di scansione stabilita. Costantemente vigila sullo stato dei **droni**, recependo i dati da essi e adattando lo stato dell'**area** di esplorazione in base alle informazioni ricevute. Inoltre, si interfaccia con un server esterno per inviare **report** sull'andamento della missione.

Per agevolare la comunicazione e il coordinamento con i **droni**, il **Control Center** adotta **Redis** come canale di comunicazione bidirezionale. Inoltre, per trasmettere i dati al server esterno, fa affidamento sulla libreria *cURL* per effettuare richieste **HTTP**.

L'intero sistema è supportato da un database **PostgreSQL**, utilizzato per immagazzinare dati vitali sui **droni**, lo stato dell'**area** di esplorazione e i **log** delle attività. Questa architettura consente al **Control Center** di gestire in modo efficiente e coordinato le operazioni di scansione dell'**area** e di fornire aggiornamenti tempestivi sul progresso della missione.

Il **Drone** si connette a un **server Redis** per ricevere direttive dal **Control Center**, seguire rotte predeterminate, monitorare il proprio stato e trasmettere aggiornamenti al **Control Center**.

I suoi metodi consentono di avviare il **drone**, ricaricare la sua autonomia e ascoltare i comandi provenienti dal **Control Center**. Inoltre, è in grado di seguire percorsi specifici, inviare aggiornamenti sul proprio stato e gestire la carica della batteria.

4.2 ScanningStrategy

Per garantire flessibilità nella determinazione dei percorsi di scansione per i droni nell'area, abbiamo adottato uno **strategy pattern**. Questo approccio ci consente di gestire diverse strategie di scansione in modo modulare e intercambiabile. Abbiamo definito un'interfaccia denominata "**ScanningStrategy**", la quale rappresenta il contratto che ogni strategia di scansione deve seguire.

Durante l'inizializzazione del **ControlCenter**, viene selezionata e impostata la strategia di scansione desiderata. Utilizzando l'interfaccia **ScanningStrategy** e la funzione **createSchedules**, il centro di controllo calcola i percorsi di scansione, che vengono quindi memorizzati nel **database**. Durante l'esecuzione, il **ControlCenter** invia i percorsi ai droni, avviando la scansione dell'area. La funzione **createSchedules** restituisce anche un timer per ogni percorso, che indica semplicemente quando un particolare percorso deve essere mandato a un drone per garantire una copertura continua e efficace dell'area.

Un'implementazione specifica della strategia è la **BasicStrategy**. Questa strategia crea percorsi semplici che coprono l'area con un movimento a zig-zag, dall'alto verso il basso della mappa.

4.3 Pseudo-codice

4.3.1 ControlCenter

```
class ControlCenter:
    function ControlCenter(id, num_drones):
        costruttore Centro di Controllo
        connetti al server redis
        connetti al database postgres
        configura stream e gruppi redis

    function start():
        inizializza droni
        avvia thread per:
            - ascoltare i droni
            - inviare percorsi ai droni
            - inviare stato dell'area al server

    function stop():
        invia comando di stop a tutti i droni
        invia comando di quit al centro di controllo

    function initDrones():
        aspetta che i droni inviino le loro posizioni e stati
        invia posizione del centro di controllo ai droni
        aggiorna database con informazioni sui droni

    function sendPaths():
        ottieni percorsi in base alla scanning strategy
        crea thread per inviare percorsi ai droni
        aspetta che tutti i thread finiscano

    function handleSchedule(schedule, redis_context):
        invia percorso al drone
        aggiorna database con log del drone
        aspetta intervallo successivo

    function listenDrones():
        ascolta continuamente messaggi dai droni
        elabora ogni messaggio
        aggiorna database e stato dei droni

    function processMessage(message):
        aggiorna area con informazioni del drone
        aggiorna stato del drone nel database e nelle liste
```

```
    function sendAreaToServer():
        invia continuamente stato dell'area al server
        stampa percentuale di copertura dell'area

    function executeQuery(query):
        blocca mutex della query
        esegui comando SQL

    function addDroneToWorking(drone):
        blocca mutex delle liste
        aggiungi drone alla lista di lavoro

    function addDroneToCharging(drone):
        blocca mutex delle liste
        aggiungi drone alla lista di ricarica

    function addDroneToReady(drone):
        blocca mutex delle liste
        aggiungi drone alla lista dei pronti

    function removeDroneFromWorking(drone):
        blocca mutex delle liste
        rimuovi drone dalla lista di lavoro

    function removeDroneFromCharging(drone):
        blocca mutex delle liste
        rimuovi drone dalla lista di ricarica

    function removeDroneFromReady():
        blocca mutex delle liste
        rimuovi e restituisci drone dalla lista dei pronti
```

4.3.2 Drone

```
class Drone:
    function Drone(id):
        costruttore drone
        stabilisci connessione a Redis
        configura stream e gruppi Redis

    function Drone(id, cc_id):
        chiama il construttore
        imposta id del centro di controllo

    function destructor():
        pulisci stream e gruppi Redis
        chiudi connessione a Redis

    function start():
        invia dati attuali al centro di controllo:
            - coordinate
            - batteria
            - stato
        inizia ad ascoltare il centro di controllo in un thread separato

    function chargeDrone():
        simula processo di ricarica
        aggiorna stato della batteria
        notifica al centro di controllo quando la ricarica e' completata
```

```
    function listenCC():
        ascolta continuamente messaggi dal centro di controllo
        elabora comandi ricevuti

    function followPath(path):
        se non pronto:
            registra stato non pronto
            ritorna
        imposta stato a lavoro
        segui il percorso dato passo dopo passo
        aggiorna posizione e stato della batteria
        notifica il centro di controllo dei progressi
        se batteria scarica o comando di stop ricevuto, esci
        quando il percorso e' completato, inizia processo di ricarica

    function sendDataToCC(changedState):
        prepara e invia dati attuali al centro di controllo
```

4.3.3 Server

```
class Server:
    function Server(host, port, password):
        costruttore server
        imposta host, port, password

    function start():
        configura contesto di rete
        risolvi indirizzo host
        crea endpoint di rete
        ascolta per connessioni in arrivo
        gestisci eventi di rete

class http_session:
    function initialize(socket):
        configura socket di rete
        stabilisci connessione al database

    function run():
        leggi richiesta in arrivo
        elabora richiesta in base al tipo

    function process_request():
       routing del server

    function handle_post_report():
        analizza dati della richiesta
        memorizza csv_url e copertura nel DB
        crea file CSV

    function handle_get_report(cc_id):
        recupera dati del report dal database
        invia file CSV in risposta
```

4.4 DB Schemas

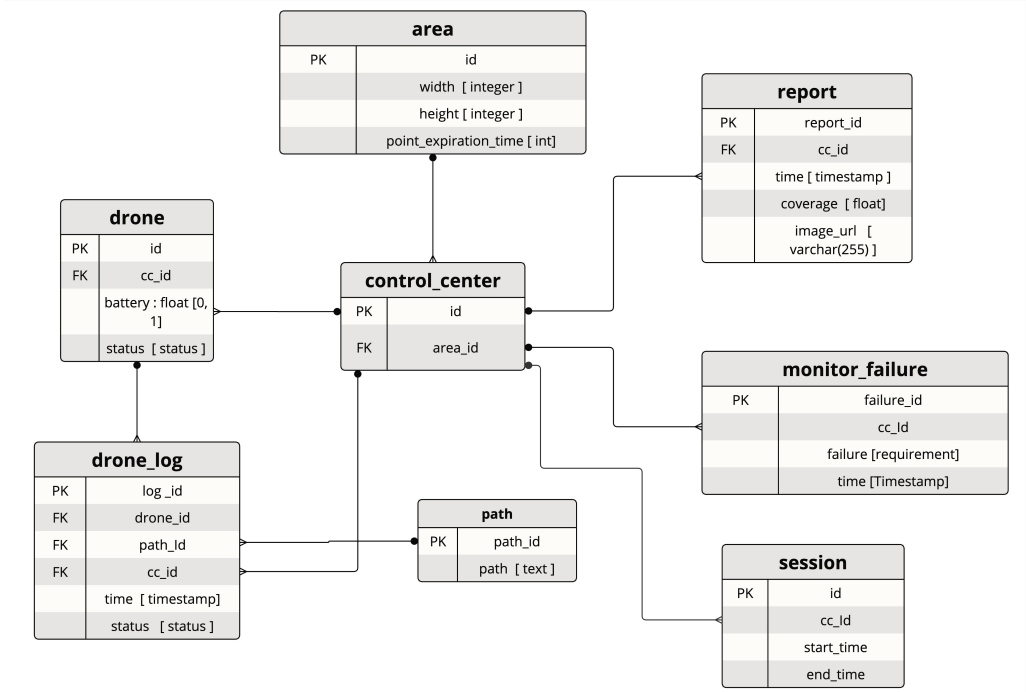

Figure 8: Schema del database

4.5 Connessioni Redis

Abbiamo utilizzato Redis per la comunicazione tra il `ControlCenter` e i droni. Per questo scopo, abbiamo creato i seguenti stream:

- Stream comune per tutti i droni: `cc_<cc-id>`
  - Tutti i droni scrivono le proprie informazioni su questo stream.
  - Il `ControlCenter` legge queste informazioni e le processa.
  - In questo stream i droni scrivono soltanto, mentre il ControlCenter legge soltanto.
- Stream individuale per ogni drone: `stream_drone_<drone-id>`
  - Ogni drone ha uno stream dedicato con il proprio ID.
  - Il `ControlCenter` scrive il percorso che il drone deve seguire su questo stream.
  - Il drone con l'ID corrispondente ascolta questo stream quando si trova nello stato `READY`.
  - In questo stream il drone associato scrive soltanto mentre il ControlCenter scrive soltanto.

5 Risultati Sperimentali

5.1 Test in condizioni ottimali

Abbiamo testato il sistema con 9000 droni e le condizioni della traccia proposta per una durata di 4 ore.

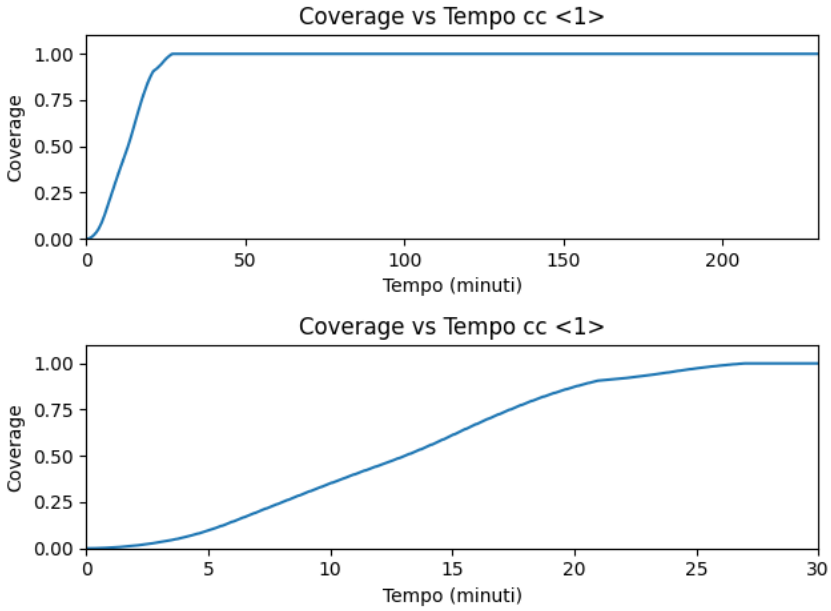

Figure 9: Test con condizioni ottimali

Dalla figura 9, si può osservare che inizialmente l'area viene coperta al 100% in circa mezz'ora. Una volta raggiunta la copertura totale, il sistema continua a mantenere questa condizione fino al termine del tempo specificato, senza mai diminuire la copertura.

5.2 Test con droni insufficienti

Per questo test abbiamo utilizzato 5000 droni. Appena avviato il programma, è stato immediatamente registrato nel database un monitor che segnalava la presenza di pochi droni, con un numero minimo di 8136.

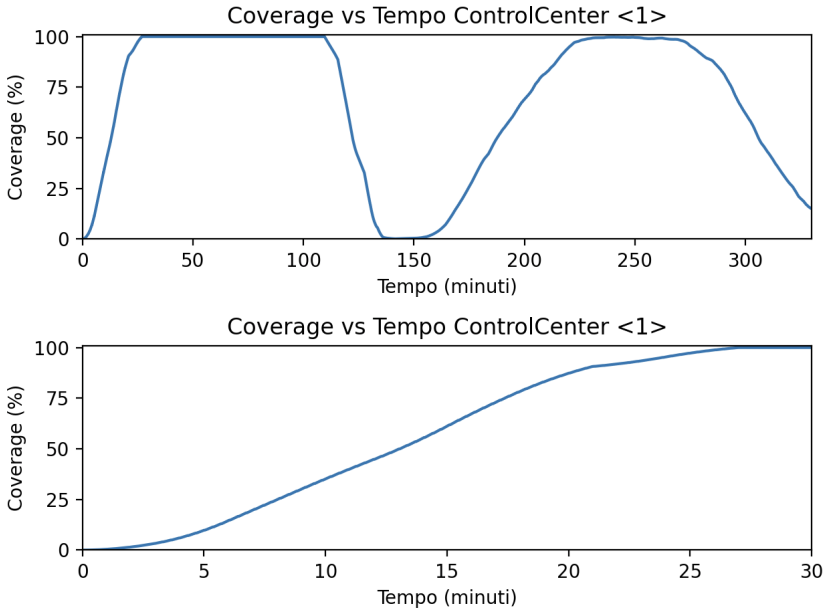

Figure 10: Test con 5000 droni

Come si può vedere nella Figura 10, all'inizio il sistema di copertura è paragonabile alla situazione ottimale. Tuttavia, in seguito, la mancanza di droni disponibili ha comportato una diminuzione della copertura. Quando tutti i droni sono andati in carica, ogni punto della mappa è diventato non coperto fino a quando i droni non sono tornati operativi.

6 Note

6.1 Monitor implementati

Abbiamo implementato diversi monitor che registrano nel database quando si verifica un problema. I tipi di monitor sono:

- `NUM_DRONES`: Notifica quando il numero di droni utilizzati è insufficiente.
- `CC_OVERLOAD`: Notifica quando lo stream Redis del centro di controllo (CC) ha troppe entries, impedendo di tenere il passo con la scansione dei droni.
- `DRONE_AUTONOMY`: Notifica quando l'autonomia del drone è troppo bassa per coprire l'intera area.
- `DRONE_OUT_OF_BATTERY`: Notifica quando la batteria di un drone è scarica e il drone non sta all'interno del centro di controllo.
- `AREA_COVERAGE`: Verifica se una volta raggiuto il 100% della copertura dell'area, essa rimane sempre del tutto coperta.

6.2 Considerazioni sull'area

Poiché il drone effettua la scansione in modo quadrato, è possibile ridimensionare l'area in modo che ogni quadrato visualizzato dal drone corrisponda a una porzione quadrata dell'area totale.

Pertanto, la larghezza e l'altezza dell'area viene ricalcolate utilizzando le seguenti formula:

$$transformed\_width = \frac{original\_width}{scan\_range \times 2} \tag{1}$$

$$transformed\_height = \frac{original\_height}{scan\_range \times 2} \tag{2}$$

6.3 Considerazioni su TIME_ACCELERATION

Il sistema può essere sottoposto a test regolando il fattore di accelerazione temporale nel file `config.h`. Questo parametro può essere modificato, (e.g. `TIME_ACCELERATION = 0.5`, che dimezza il tempo). Tuttavia, è essenziale considerare che impostare il valore di questo fattore prossimo allo 0 potrebbe compromettere l'affidabilità dei risultati ottenuti.

Tale inconveniente deriva dal sovraccarico dello stream Redis del `ControlCenter`, noto come `"cc_<id>"`, dovuto al grande volume di richieste, generando così un **bottleneck**.

Si evidenzia il fatto che i test precedenti sono stati eseguiti in tempo reale, corrispondente a `TIME_ACCELERATION = 1`.

7 Conclusioni

Grazie a questo progetto, abbiamo avuto l'opportunità di approfondire e applicare le nostre conoscenze di C++ e Redis in un contesto pratico. Questo ci ha permesso di non solo migliorare le nostre abilità tecniche, ma anche di comprendere le migliori pratiche per lo sviluppo di software complessi. Abbiamo esplorato diversi paradigmi di programmazione, focalizzandoci su tecniche che promuovono la modularità e la leggibilità del codice. Questa esperienza ci ha insegnato l'importanza di una progettazione accurata e di un'implementazione strutturata.