SAPIENZA
UNIVERSITÀ DI ROMA

Software Engineering

AY 2023/2024

Madia Nicolò <madia.1960585@studenti.uniroma1.it>
Fabio Figara <figara.1953242@studenti.uniroma1.it>
Aliotta Daniele <aliotta.1957734@studenti.uniroma1.it>

Computer Science Department, Sapienza university of Rome

Progetto:

Controllo formazione Droni

1. Descrizione generale

Il progetto si propone di sviluppare un sistema avanzato di controllo e sorveglianza per monitorare attentamente un'ampia area di dimensioni 6x6 chilometri. Questo sistema si basa sull'utilizzo di droni, dotati di specifiche caratteristiche progettate per garantire una sorveglianza efficace e continua dell'area designata.

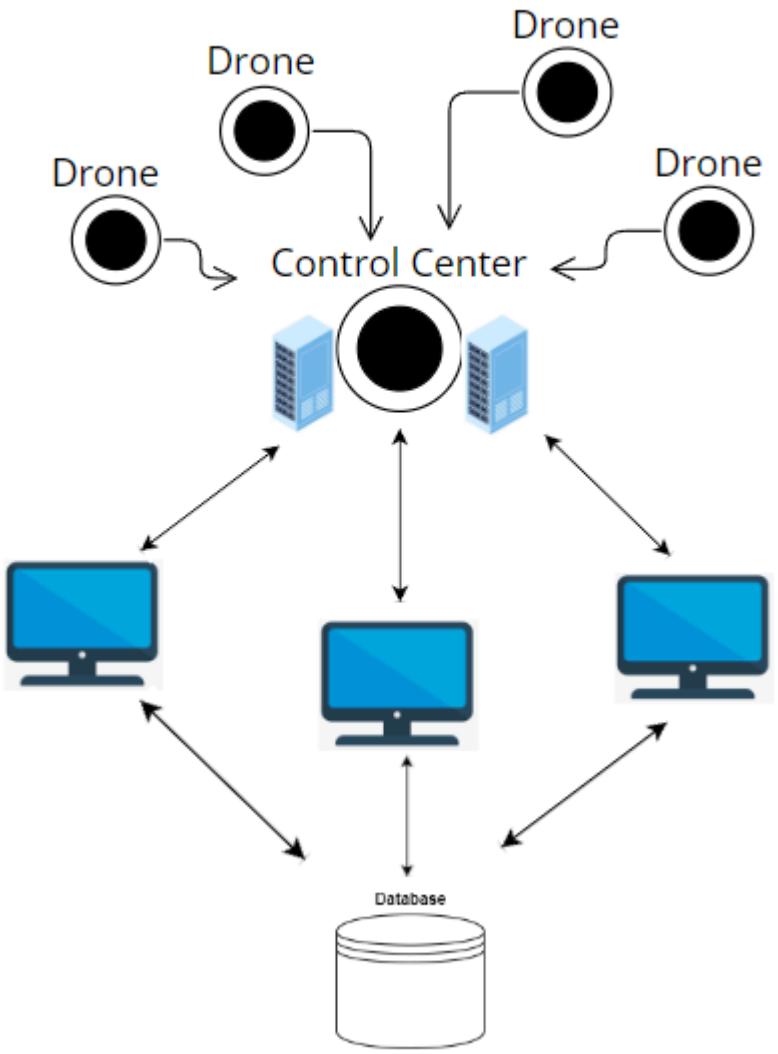
Ogni drone è equipaggiato con una batteria che fornisce un'autonomia di funzionamento di 30 minuti. Tuttavia, per garantire un funzionamento continuo del sistema, è previsto un tempo di ricarica casuale, il quale varia tra le 2 e le 3 ore. Questa variazione nel tempo di ricarica contribuisce a distribuire in modo equo il carico di lavoro tra i droni e a garantire che il sistema rimanga operativo anche durante i periodi di ricarica.

Inoltre, ciascun drone è dotato di un range di visibilità pari a 10 metri e si muove a una velocità costante di 30 chilometri all'ora. Queste specifiche tecniche sono state scelte per garantire una copertura efficace e dettagliata dell'area monitorata, consentendo ai droni di sorvegliare ogni sezione dell'area con precisione e tempestività.

Al centro dell'area è situato un centro di controllo dedicato, il cui ruolo fondamentale è quello di coordinare le attività dei droni lungo le rotte predefinite. Questo centro di controllo agisce come il cervello del sistema, scambiando continuamente informazioni con ciascun drone ad ogni istante di tempo t. Il suo obiettivo principale è garantire che nessuna sezione dell'area rimanga non sorvegliata per più di 5 minuti, mantenendo così un controllo costante e completo sull'intera area di interesse.

In sintesi, il sistema proposto si basa sull'utilizzo innovativo di droni dotati di specifiche caratteristiche e su un centro di controllo centralizzato per garantire una sorveglianza efficace, continua e dettagliata di un'area di dimensioni significative. Grazie a questa combinazione di tecnologie avanzate e strategie di coordinamento sofisticate, il sistema è in grado di rispondere in modo tempestivo a eventuali minacce o situazioni di emergenza, fornendo una sicurezza ottimale per l'area monitorata.

Architettura del Sistema



1.2 Struttura del progetto

Il progetto è organizzato in diverse cartelle, ciascuna svolge un ruolo specifico nell'implementazione e nel funzionamento complessivo del sistema. Di seguito, una descrizione dettagliata di ciascuna cartella:

1. Cartella Control Center:

- Questa cartella contiene il codice sorgente e gli header file relativi al Control Center del sistema. Il Control Center è il cuore del sistema di gestione e sorveglianza, responsabile della pianificazione delle rotte dei droni, del coordinamento delle attività di sorveglianza e della comunicazione con i droni attivi nell'area di interesse.
- All'interno di questa cartella, è presente anche il file makefile che definisce le regole di compilazione specifiche per il Control Center e facilita il processo di compilazione e di generazione dell'eseguibile.

2. Cartella Drone:

- La cartella Drone contiene il codice sorgente e gli header file relativi all'implementazione e al funzionamento dei droni nel sistema. Ogni drone è rappresentato da un'istanza di classe con funzionalità specifiche, incluse le capacità di movimento, comunicazione con il Control Center e gestione dell'autonomia e della ricarica.
- In questa cartella, è incluso anche il file makefile dedicato per la compilazione del codice sorgente dei droni, garantendo un processo di compilazione separato e autonomo.

3. Cartella con2redis:

- La cartella con2redis ospita il codice necessario per l'integrazione del sistema con Redis, un sistema di gestione di database chiave-valore in memoria. Questa integrazione è finalizzata alla sincronizzazione del tempo, alla gestione dei messaggi tra il Control Center e i droni, e alla registrazione dei log di sistema.
- La connessione e la gestione della comunicazione con Redis sono implementate all'interno della classe "swarm", che si occupa di mantenere attiva e gestire la connessione al database Redis durante il funzionamento del sistema.

2. User requirements

i requisiti utente sono i seguenti:

1) i droni devono poter sorvegliare l'area in modo uniforme:
    - ognuno seguirà la tratta di sorveglianza assegnata dal Control Center, garantendo la copertura totale dell'Area di interesse

2) i droni devono essere in grado di comunicare con il Control Center:
    - ad ogni istante di tempo t, devono poter ricevere messaggi dal Centro di Controllo che ne coordina gli spostamenti

3) i droni devono essere in grado di gestire la loro ricarica in Autonomia:
    - ad ogni passo, se la loro carica consente ancora n spostamenti e dal punto in cui si trovano sono necessari k passi per tornare alla stazione di ricarica se $n = k + \varepsilon$ *(margine)* all'istante di tempo t +1 si dirigeranno verso il Control Center che invierà un Drone sostituto, altrimenti se $n > k + \varepsilon$ continueranno a proseguire sulla rotta di sorveglianza assegnata

3. System requirements

il principale obiettivo del sistema è consentire, mediante il Control Center, il monitoraggio di un’area 6x6km grazie alla frammentazione logica di quest’ultima in sottoaree controllabili tramite la gestione e il coordinamento di Droni.
I ***requisiti funzionali*** di Sistema sono i seguenti:

1) Sorveglianza Uniforme dell’area da monitorare:
    - I droni devono coprire l'intera area di sorveglianza in modo uniforme, assicurando che ogni punto della rotta assegnatagli da Control Center venga verificato almeno ogni 5 minuti.
    - L’area deve essere interamente verificata in un tempo $\Delta t <= 15$ minuti.

2) Coordinamento dei Droni:
    - Il Sistema deve tenere traccia della posizione dei droni e dirigere i loro movimenti in modo da ottimizzare la copertura totale dell’area da monitorare.
    - Il Sistema deve gestire efficientemente le richieste di ricarica, garantendo che il livello della batteria dei Droni non deve scenda mai sotto il margine 1.0.
    - Il Sistema deve evitare colli di bottiglia e garantire quindi la copertura relativa al punto 1 e la modularità del sistema stesso.

3) Comunicazione Affidabile:
    - i droni devono trasmettere regolarmente informazioni sul loro stato e sulla loro posizione.
    - il Control Center deve ricevere e processare i dati in tempo reale, inviando istruzioni di risposta in base ai messaggi ricevuti dai dispositivi attivi

I ***requisiti non funzionali*** di Sistema sono i seguenti:

1) Efficienza energetica:
    - Il sistema deve essere in grado di gestire un numero variabile di droni, quelli necessari a garantire i requisiti di sistema.
    - La ricarica completa di un singolo Drone deve avvenire in un tempo variabile tra 120 e 180 minuti.

2) Scalabilità del sistema:
    - L'architettura del sistema deve essere modulare e flessibile per consentire, qualora necessaria. l'aggiunta di nuovi droni o l'estensione dell'area di copertura senza influire sulla logica di funzionamento.

3) Vincoli di Sistema:
    - il Sistema deve consentire la circolazione di un numero N di Droni pari o inferiore a 10’000

Monitors

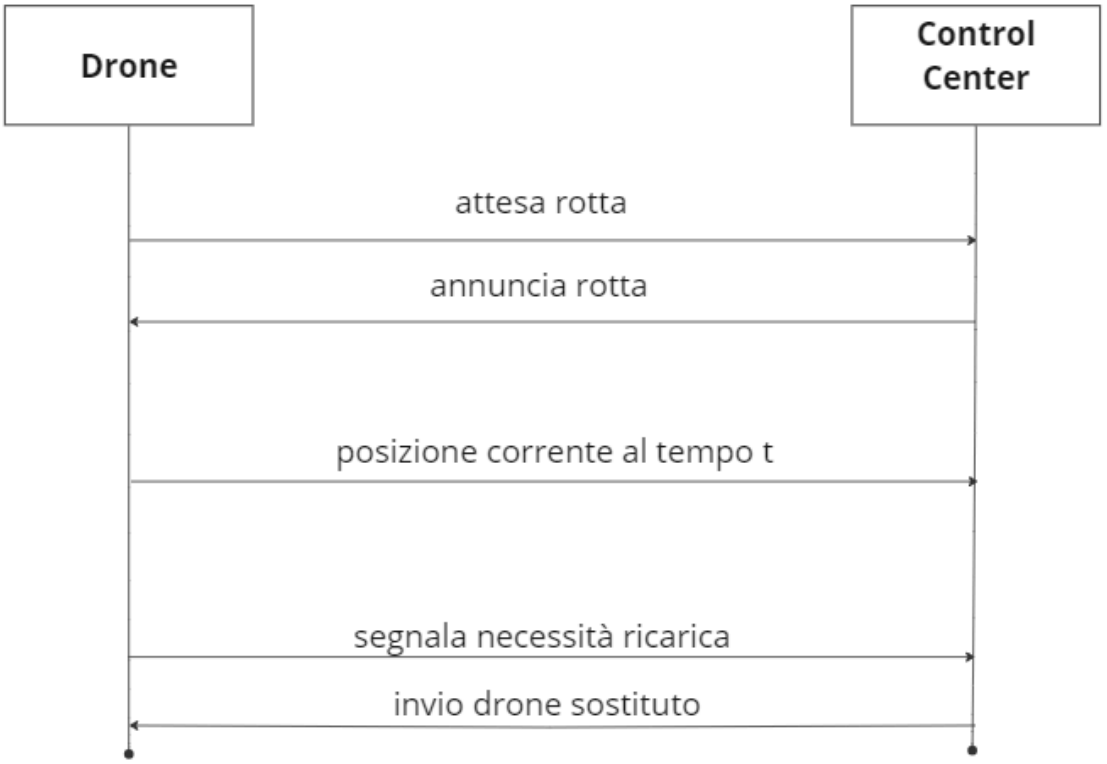
I monitors sono utilizzati per verificare il rispetto dei suddetti requisiti Utente e di Sistema soddisfacendo i seguenti requisiti:
:

- si attiva ad ogni istante di tempo t e monitora che ogni punto dell'area sia verificato ogni 5 minuti.

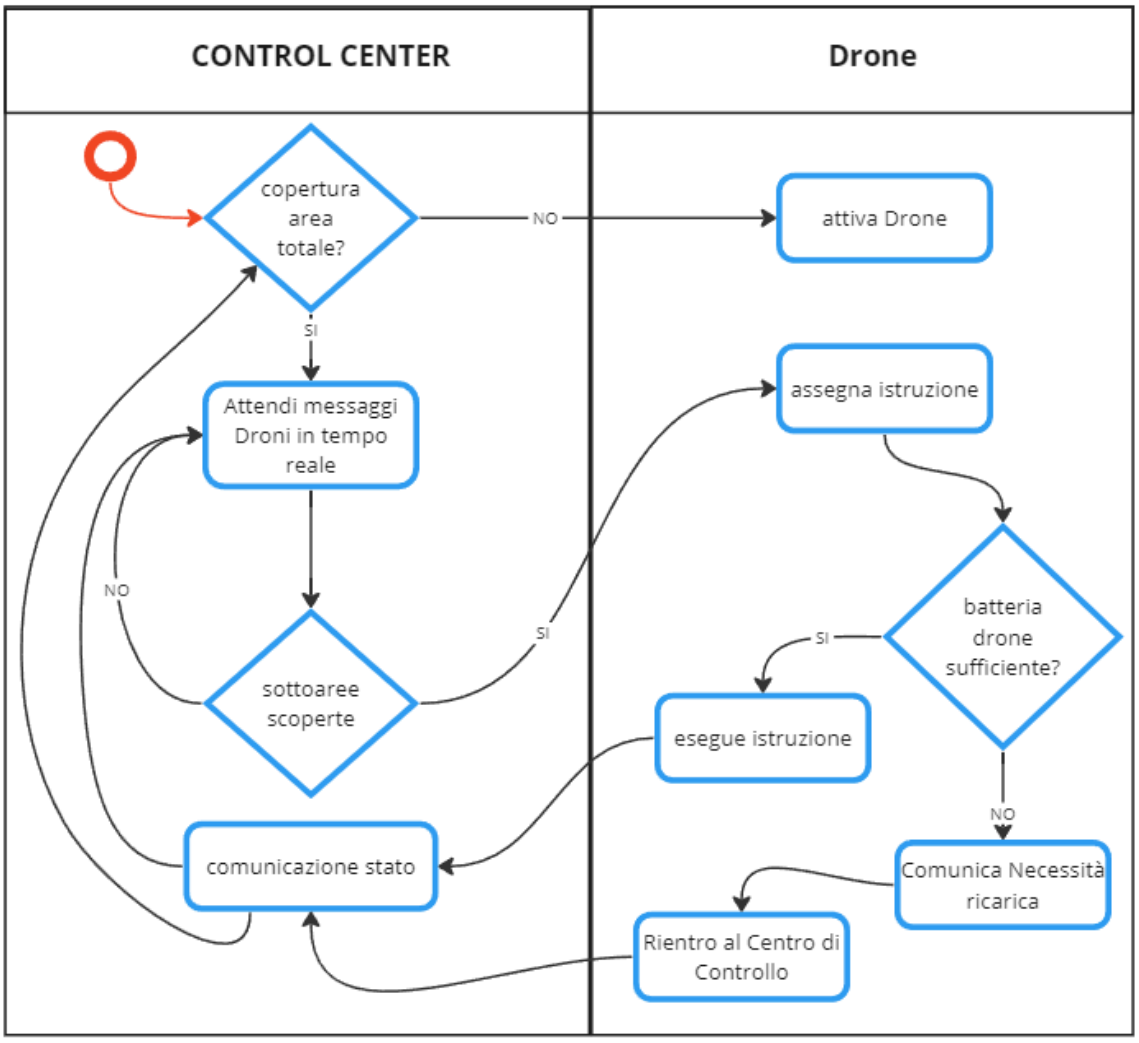
- Verifica che, dall' inizio della simulazione, la sorveglianza completa dell'area si ottenga in un tempo inferiore o uguale a 15 minuti.

- si attiva ad ogni istante di tempo t e verifica che la batteria dei droni non scenda mai al di sotto del limite 1.0.

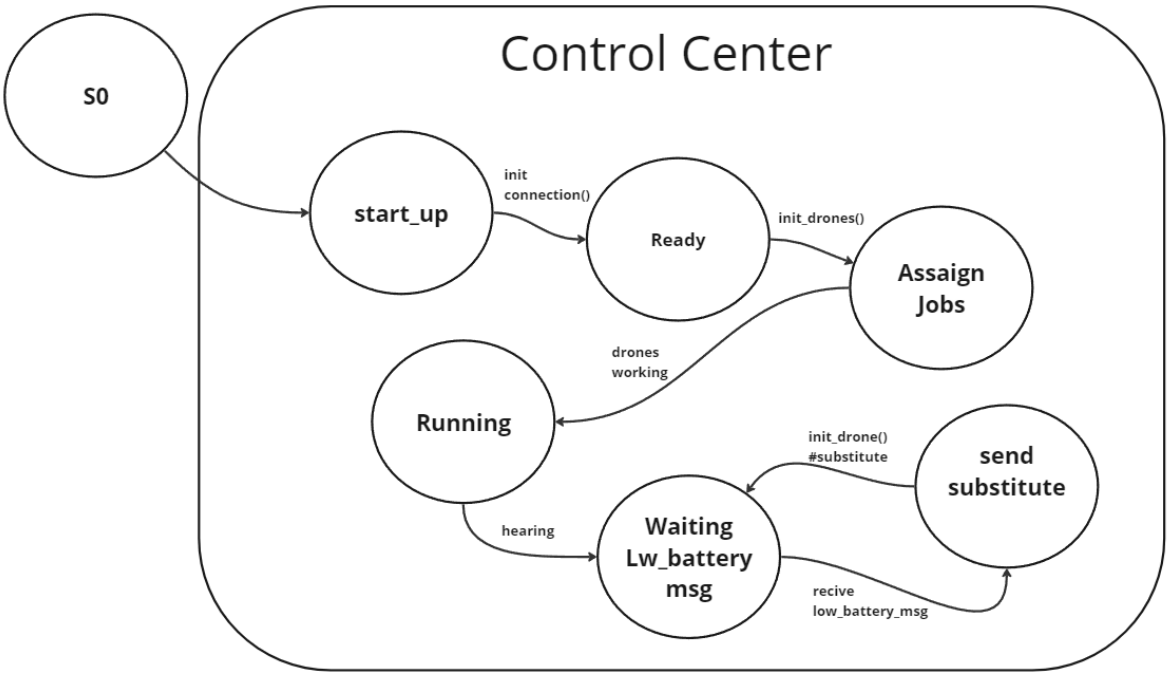
Message sequence Chart Diagram

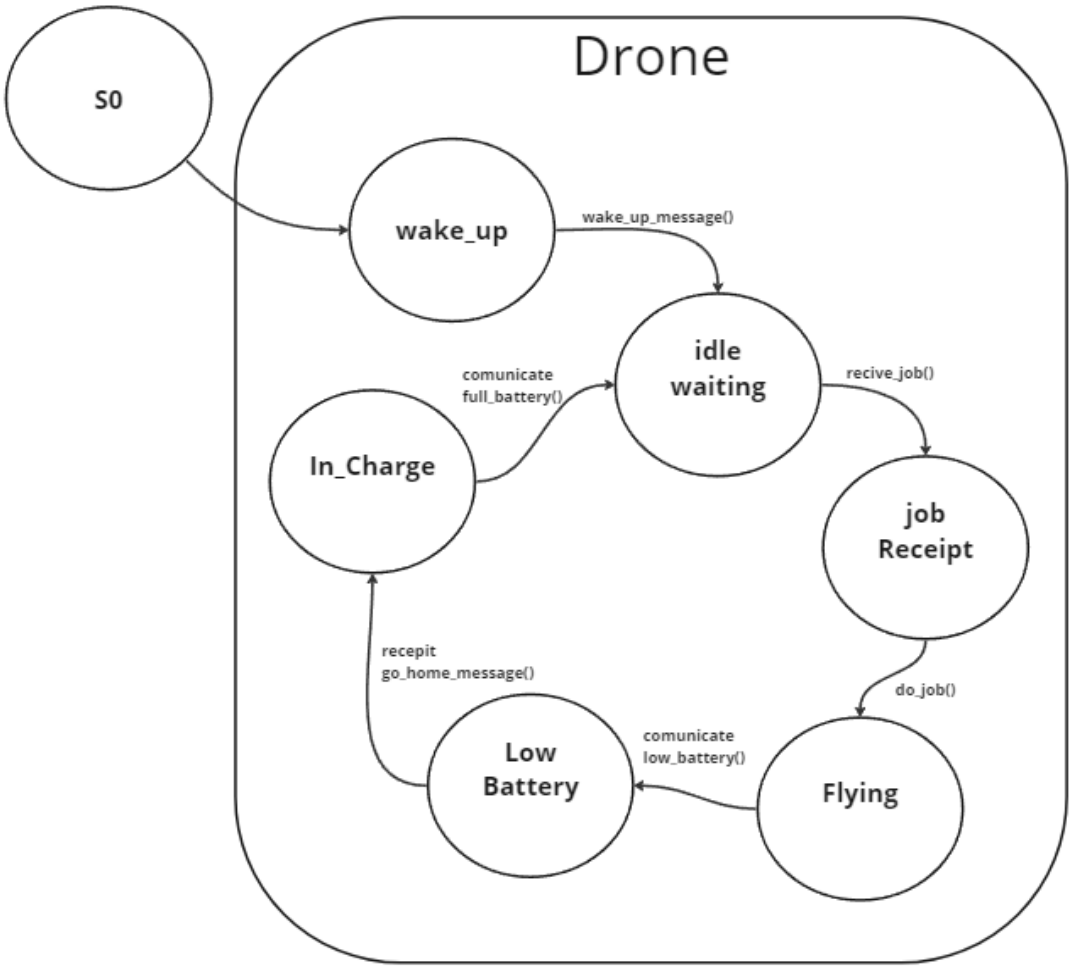


Activity Diagram



State Diagram





4. Implementation

Il progetto di controllo formazione droni implementa le seguenti proprietà:

- Rispetto dei requisiti: Il progetto implementa tutti i requisiti utenti e di sistema.
- Efficienza: Il progetto è efficiente in termini di tempo e consumo di energia.

4.1 Pseudocodice

4.1.2 ControlCenter.cpp

Funzione tick()

```
Se lo stato è STARTUP allora
    Inizializzazione del sistema

Se lo stato è READY allora
    Assegnazione dei compiti ai droni

Se lo stato è RUNNING allora
    Gestione dei messaggi dai droni
```

Funzione handle_msg(tipo_messaggio, messaggio)

```
Switch su tipo_messaggio
    Caso "low_battery":
        Creazione di un nuovo compito per un drone
    Caso "charging":
        Aggiornamento dello stato di un drone a CHARGING_D
    Caso "recharged":
        Aggiornamento dello stato di un drone a IDLE_D
```

Funzione check_area(tempo_corrente)

- Controllo dell'area per verifica completa
- Ritorno del risultato della verifica

Funzione log(tempo)

- Registrazione dei dati e monitoraggio
- Controllo dell'area e del tempo
- Aggiornamento della griglia e dei dati dei droni

Funzione shutdown()

- Pulizia delle risorse
- Chiusura della connessione a Redis

Drone.cpp

Funzione int_to_string(x: intero):
    Converti l'intero x in una stringa
    Ritorna la stringa risultante

Funzione read_stream(c: puntatore a redisContext, stream_name: puntatore a char):
    Leggi uno stream di dati da Redis a partire dall'inizio
    Ritorna la risposta ottenuta dalla lettura

Costruttore Drone(did: intero):
    Inizializza un nuovo drone con l'ID specificato
    Imposta i valori iniziali della batteria, posizione, stato, velocità, ecc.

Metodo addDrone(drone: Drone):
    Aggiunge un drone alla lista dei droni gestiti dalla Swarm

Metodo init_drones(n: intero):
    Inizializza un numero specificato di droni e li aggiunge alla Swarm

Metodo await_sync():
    Attende messaggi di sincronizzazione dalla rete
    Invia un messaggio di sincronizzazione alla rete

Metodo init():
    Inizializza la connessione Redis e verifica la connessione

Funzione get_job_from_reply(reply: puntatore a redisReply):
    Ottiene un lavoro da una risposta Redis e lo restituisce

Funzione get_random_charge_time():
    Genera un tempo casuale di ricarica compreso tra un minimo e un massimo
    Ritorna il tempo di ricarica casuale generato

Metodo get_distance_control_center():
    Calcola la distanza del drone dal centro di controllo
    Ritorna la distanza calcolata

Metodo is_charged():
Verifica se il drone è completamente carico
Ritorna true se il drone è completamente carico, altrimenti false

Metodo charge():
Simula il processo di ricarica del drone

Metodo is_home():
Verifica se il drone è nella posizione di casa (centro di controllo)
Ritorna true se il drone è nella posizione di casa, altrimenti false

Metodo is_done():
Verifica se il compito del drone è completato
Ritorna true se il compito del drone è completato, altrimenti false

Metodo get_autonomy():
Calcola l'autonomia residua del drone
Ritorna l'autonomia residua calcolata

Metodo is_low_battery():
Verifica se il livello della batteria del drone è basso
Ritorna true se il livello della batteria è basso, altrimenti false

Metodo set_last_dist():
Imposta la distanza dall'ultimo punto di riferimento del drone

Metodo send_replace_drone_msg(c: puntatore a redisContext):
Invia un messaggio di sostituzione del drone alla rete

Metodo calc_velocity(destx: doppio, desty: doppio):
Calcola la velocità del drone per raggiungere la destinazione specificata

Funzione reset_job(job: puntatore a Job):
Reimposta il lavoro del drone per un nuovo compito

Metodo move(t: intero):
Simula il movimento del drone in base al tempo trascorso

Metodo tick(c: puntatore a redisContext, t: intero):
Esegue il tick del drone in base al tempo trascorso e alla connessione Redis

readreply.cpp

Funzione ReadNumStreams(r: puntatore a redisReply):
Ritorna il numero di stream presenti nella risposta r

Funzione ReadStreamName(r: puntatore a redisReply, streamname: puntatore a char, k: intero non segnato):
Legge il nome del k-esimo stream dalla risposta r e lo memorizza in streamname

Funzione ReadStreamNumMsg(r: puntatore a redisReply, streamnum: intero non segnato):
Ritorna il numero di messaggi presenti nel k-esimo stream dalla risposta r

Funzione ReadStreamNumMsgID(r: puntatore a redisReply, streamnum: intero non segnato, msgnum: intero, msgid: puntatore a char):
Legge l'ID del messaggio msgnum-esimo nel k-esimo stream dalla risposta r e lo memorizza in msgid

Funzione ReadStreamMsgNumVal(r: puntatore a redisReply, streamnum: intero non segnato, msgnum: intero):
Ritorna il numero di valori nel messaggio msgnum-esimo nel k-esimo stream dalla risposta r

Funzione ReadStreamMsgVal(r: puntatore a redisReply, streamnum: intero non segnato, msgnum: intero, entry: intero, value: puntatore a char):
Legge il valore dell'entry-esimo campo nel messaggio msgnum-esimo nel k-esimo stream dalla risposta r e lo memorizza in value

redisfun.cpp

Funzione print_reply_types():
    Stampa i tipi di risposta Redis disponibili

Funzione assertReplyType(c: puntatore a redisContext, r: puntatore a redisReply, type: intero):
    Se r è NULL, interrompi con un errore specificando il messaggio di errore nel contesto c
    Se il tipo di r è diverso da type:
        Stampa i tipi di risposta Redis disponibili
        Interrompi con un errore specificando il tipo di risposta atteso e il tipo di risposta ricevuto

Funzione assertReply(c: puntatore a redisContext, r: puntatore a redisReply):
    Se r è NULL, interrompi con un errore specificando il messaggio di errore nel contesto c

Funzione dumpReply(r: puntatore a redisReply, indent: intero):
    buffer = stringa vuota
    Switch sul tipo di risposta r:
        Caso REDIS_REPLY_STRING:
            Aggiungi "(string) " seguito dal valore di r->str a buffer
        Caso REDIS_REPLY_STATUS:
            Aggiungi "(status) " seguito dal valore di r->str a buffer
        Caso REDIS_REPLY_INTEGER:
            Aggiungi "(integer) " seguito dal valore di r->integer a buffer
        Caso REDIS_REPLY_NIL:
            Aggiungi "(nill)" a buffer
        Caso REDIS_REPLY_ERROR:
            Aggiungi "(error) " seguito dal valore di r->str a buffer
        Caso REDIS_REPLY_ARRAY:
            Per ogni elemento i in r->elements:
                Richiama ricorsivamente dumpReply(r->element[i], indent + 2)
    Se la lunghezza di buffer è maggiore di 0:
        Per ogni i da 0 a indent:
            Stampa uno spazio
        Stampa buffer
    Libera la memoria allocata per buffer

Funzione read_1msg(c: puntatore a redisContext, group: puntatore a char, consumer: puntatore a char, stream_name: puntatore a char):
    Invia un comando Redis per leggere un messaggio dallo stream specificato
    Interrompi con un errore se la risposta è nulla
    Ritorna la risposta

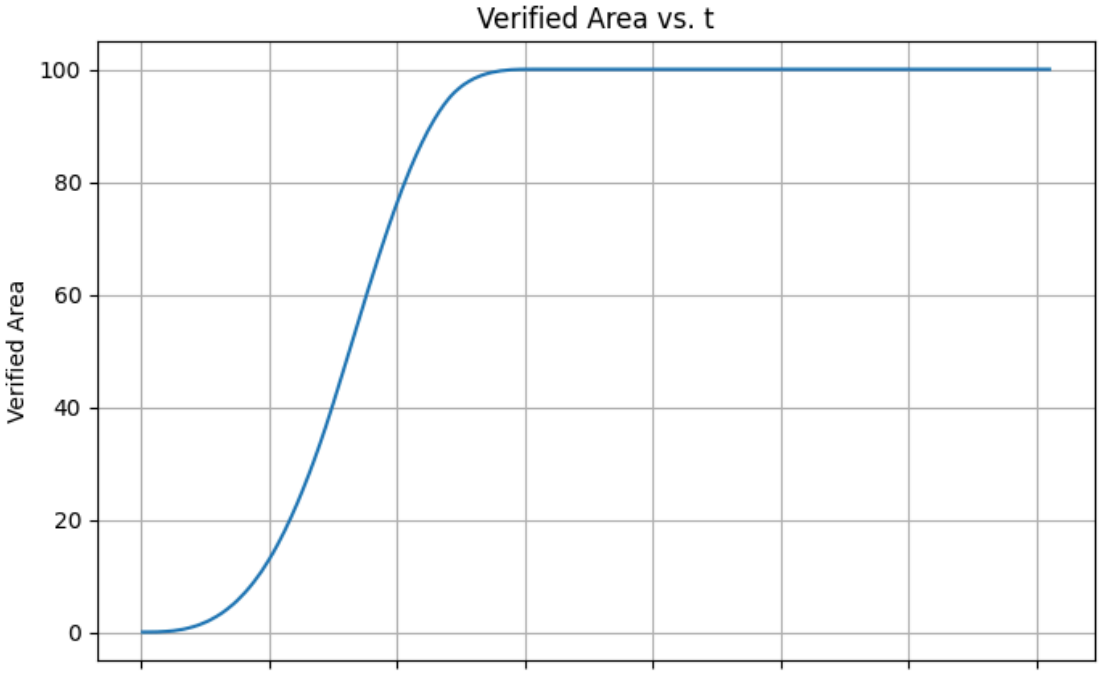
Funzione read_1msg_blocking(c: puntatore a redisContext, group: puntatore a char, consumer: puntatore a char, block: intero, stream_name: puntatore a char):
    Invia un comando Redis per leggere un messaggio dallo stream specificato con blocco
    Interrompi con un errore se la risposta è nulla
    Ritorna la risposta

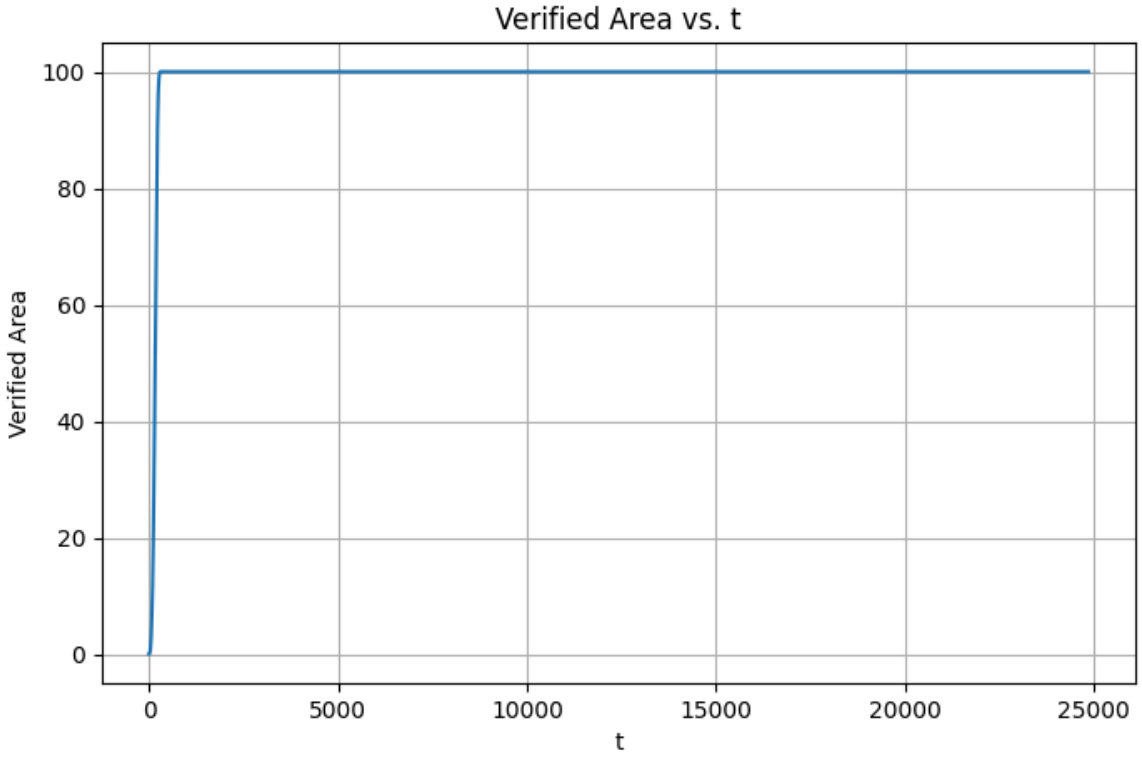
Funzione send_redis_msg(c: puntatore a redisContext, stream_name: puntatore a char, message: puntatore a char):
    Invia un messaggio Redis allo stream specificato
    Interrompi con un errore se la risposta è nulla
    Ritorna la risposta

Funzione initStreams(c: puntatore a redisContext, stream: puntatore a char):
    Invia un comando Redis per inizializzare gli stream specificati
    Interrompi con un errore se la risposta è nulla
    Libera la memoria allocata per la risposta

5. Risultati sperimentali

Attraverso l'analisi dei dati, intendiamo individuare tendenze, identificare picchi di carico e valutare l'impatto delle configurazioni del sistema sulle prestazioni complessive.





Sono state eseguite due simulazioni, rispettivamente una da 720 s e una da 25000 s, per poter ottenere i campioni utili al fine della valutazione.
Dai test eseguiti si evincono le seguenti conclusioni:

- Per garantire la copertura totale dell'area rispettando il vincolo che un qualunque punto della mappa deve essere scansionato almeno ogni 5 minuti sono necessarie circa 10.000 unità di Droni.

- dall'avvio della simulazione sono necessari al più 15 minuti per garantire che l'intera Area sia sorvegliata.

Qualora i droni non siano sufficienti a mantenere l'area verificata la simulazione avrà l'esito riportato nella seguente immagine: