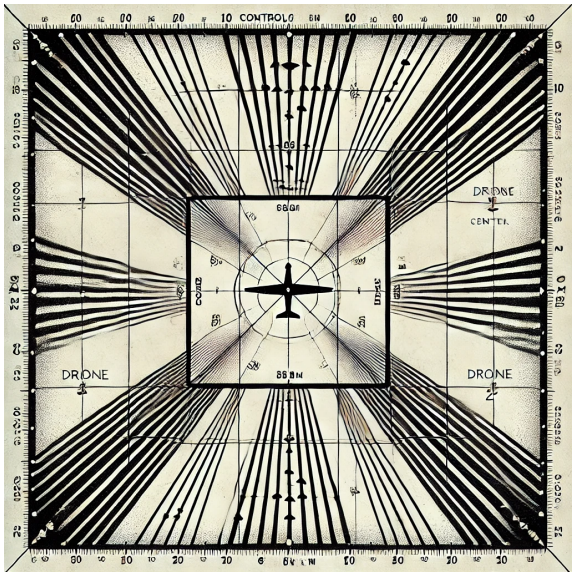

SAPIENZA
UNIVERSITÀ DI ROMA

# Ingegneria del Software

## Progetto Sky Watch




Autori
*Davide Pietragalla 1965270*
*Damiano Pezzillo Iacono 1892597*

# Indice

# 1 Introduzione

Il progetto prevede l'implementazione di un sistema di sorveglianza per un'area di $6km^2$, utilizzando droni gestiti da un **centro di controllo** situato al centro dell'area stessa.

Le specifiche fornite stabiliscono che ogni **drone** abbia un'autonomia di volo di 30 minuti, un tempo di ricarica compreso tra le 2 e le 3 ore (scelto randomicamente), e che si muova alla velocità costante di 30Km/h.

L'obiettivo è quello di garantire che ogni punto dell'area venga verificato almeno ogni 5 minuti.
Un punto è verificato quando è presente un drone entro i 10 metri di distanza.

Infine è stato richiesto un modello per i droni, un modello per il centro di controllo, una base di dati, 3 monitor di proprietà funzionali, 2 monitor di proprietà non funzionali e un'adeguata comunicazione tra i componenti.

# 2 Requisiti dell'Utente

Basandoci sulle specifiche del progetto siamo riusciti ad individuare i seguenti user requirements, e li abbiamo suddivisi in **requisiti funzionali**, e **requisiti non funzionali**.

## 2.1 Requisiti Funzionali

Questa prima categoria descrive le funzionalità specifiche del sistema:

I. Ogni punto dell'area deve essere verificato entro 5 minuti

II. I droni devono avere un raggio di verifica di 10 metri

III. Il centro di controllo deve gestire i droni

IV. Ogni ricarica deve avere una durata presa casualmente tra le 2 e le 3 ore

V. Il sistema deve registrare in un database tutte le informazioni rilevanti

VI. L'utente deve poter vedere la percentuale di coverage attuale

VII. L'utente deve poter avviare e terminare il sistema

## 2.2 Requisiti Non Funzionali

In secondo luogo abbiamo i requisiti che riguardano le prestazioni, l'affidabilità e altri aspetti qualitativi:

I. Il sistema deve essere in grado di mantenere un certo grado di affidabilità ed economicità

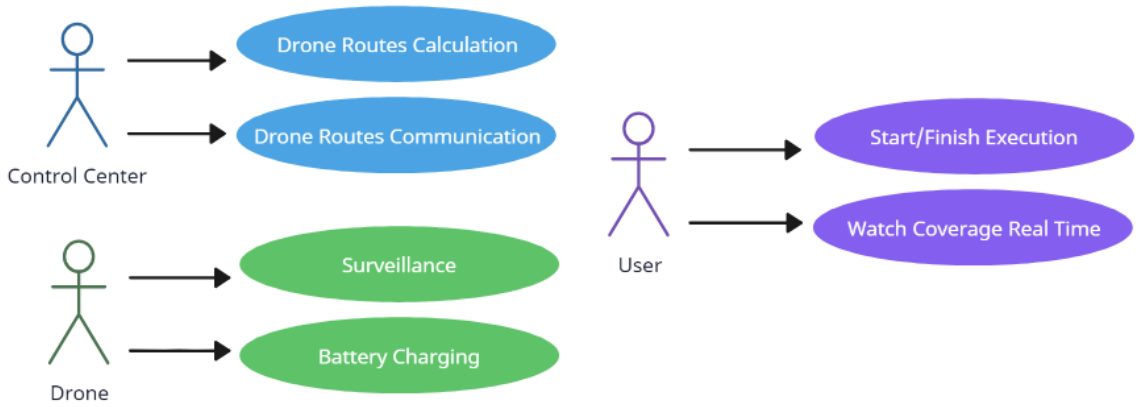
II. Una volta raggiunto il 100% del coverage, il sistema deve essere in grado di mantenerlo fino alla terminazione

III. Ogni drone deve tornare alla base di ricarica prima di esaurire la propria autonomia

IV. Il centro di controllo e la base di ricarica devono trovarsi al centro dell'area

## 2.3 UML degli Use Case

Segue la rappresentazione in UML (Unified Modeling Language) degli Use Case, diagrammi che descrivono le interazioni tra un sistema e i suoi utenti:



# 3 Requisiti del Sistema

In modo analogo al capitolo precedente, abbiamo individuato alcuni requisiti del sistema.

## 3.1 Requisiti Funzionali

Le funzionalità specifiche che il sistema deve offrire per soddisfare validità e integrità:

I. Il centro di controllo deve calcolare le rotte dei droni

II. Il centro di controllo deve inoltrare le rotte ai droni

III. Il centro di controllo deve ricevere le informazioni necessarie dai droni

IV. Il centro di controllo deve stabilire se un punto non è stato verificato entro 5 minuti

V. Il drone deve ricevere la rotta dal centro di controllo

VI. Il drone deve essere inserito tra i droni disponibili una volta terminata la ricarica

VII. Il centro di controllo deve registrare nel database i dati di una sessione

## 3.2 Requisiti Non Funzionali

Le qualità e le prestazioni che il sistema deve mantenere, come usabilità, affidabilità e scalabilità:

I. Il sistema deve essere in grado di ottenere un coverage del 100% con un massimo di 9000 droni

II. La comunicazione tra i vari componenti avviene tramite stream Redis

III. Le connessioni redis non devono superare le 10.000 unità

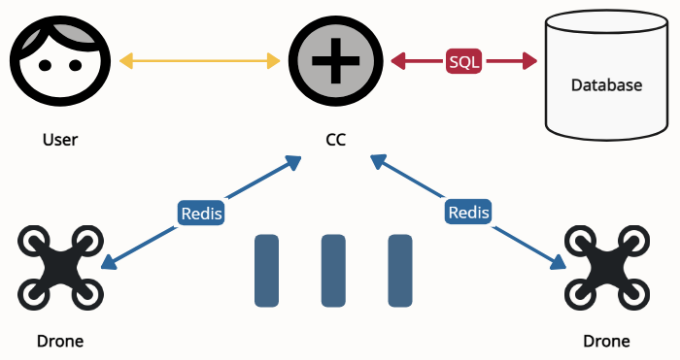
IV. La lunghezza degli stream non deve superare la soglia di 4000 messaggi

V. I dati rilevanti verranno registrati in un database PostgreSQL

VI. Generazione e invio delle query inerenti la reportiva deve concludersi nel giro di un secondo

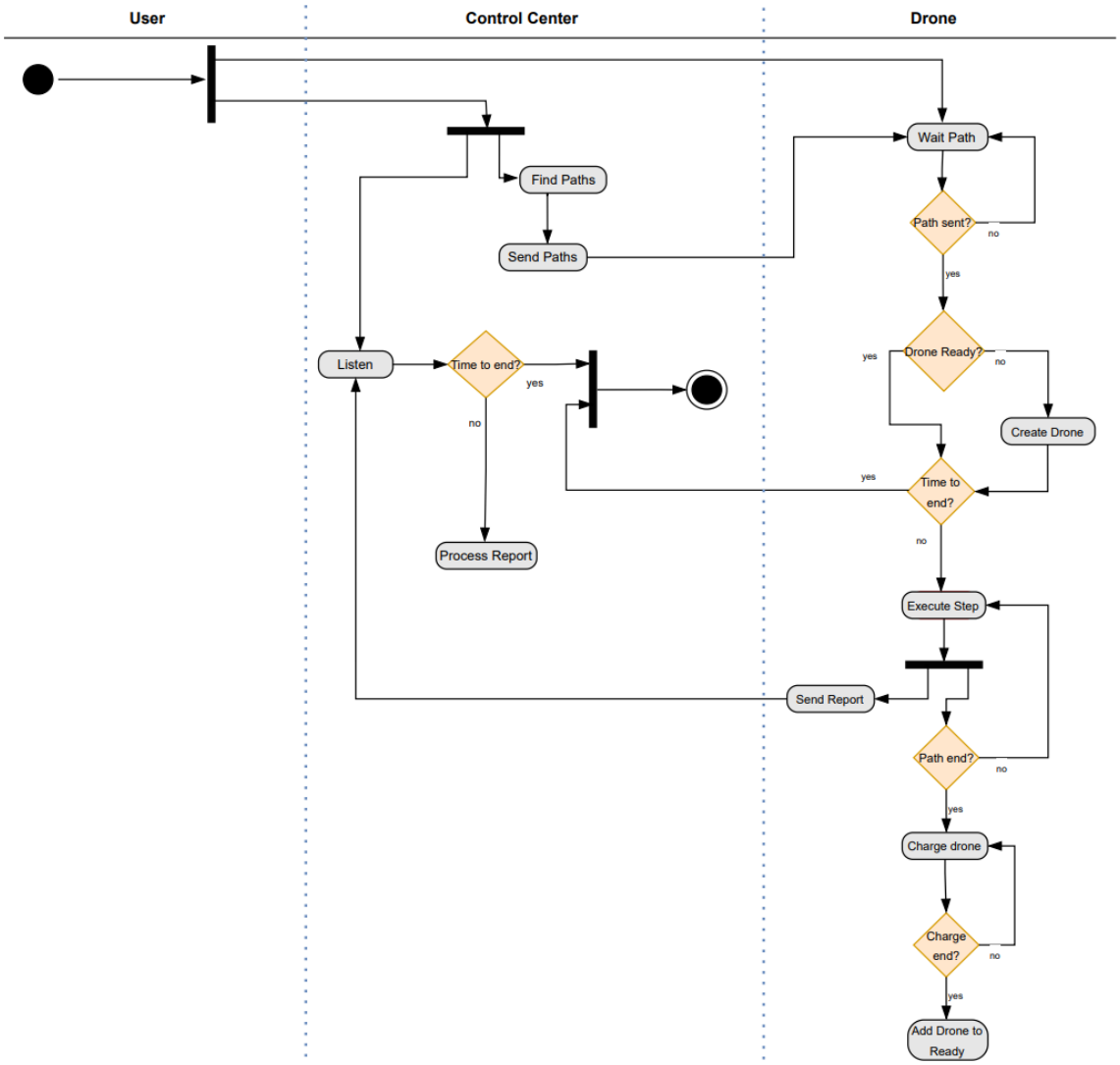
## 3.3 Architettura del Sistema

Lo schema illustra i componenti principali del sistema e le interazioni fra di essi:



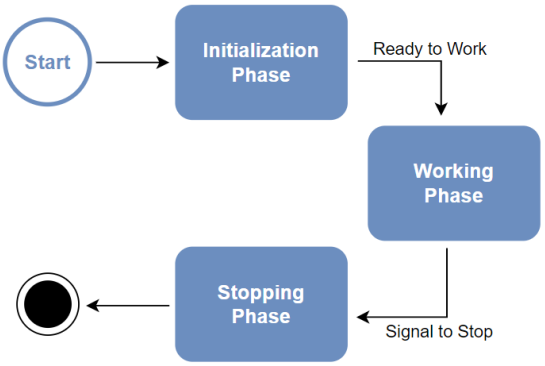
## 3.4 Activity Diagram

Il seguente diagramma fornisce una rappresentazione del flusso di esecuzione del centro di controllo e dei droni dall'inizio alla fine dell'applicazione:

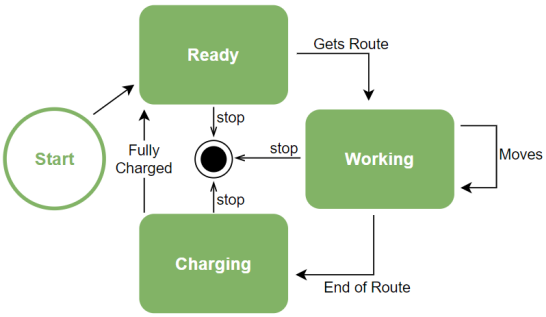


## 3.5 State Diagrams

Una rappresentazione degli stati del centro di controllo e delle transizioni tra di essi in base ai possibili eventi:

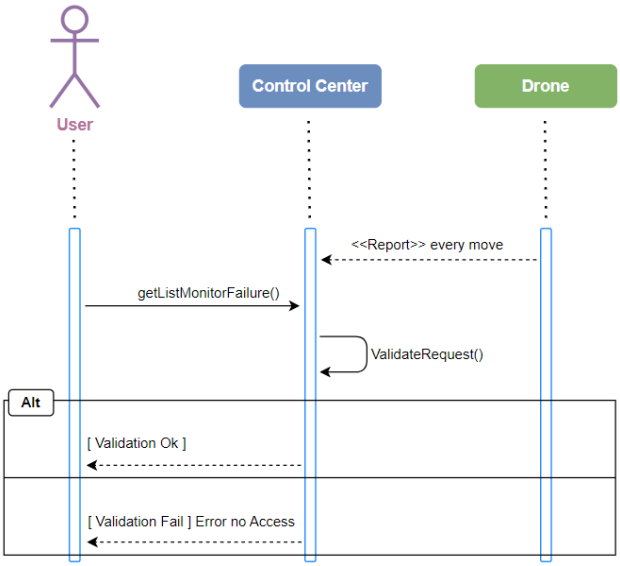



Una rappresentazione degli stati dei droni e delle transizioni tra di essi in risposta ai vari eventi e condizioni possibili:




## 3.6 Message Sequence Chart

Questo grafico raffigura lo scambio di messaggi che avviene per l'ottenimento di una lista di monitor_failure:



# 4 Implementazione

## 4.1 Strategia Sfruttata

La strategia che il centro di controllo utilizza per calcolare il percorso dei droni e garantire una copertura ottimale, consiste nell'indicare a ciascun drone un punto strategico dove iniziare la sua ronda.

Il drone proseguirà verso est quando la coordinata Y è pari, e verso ovest quando è dispari, spostandosi verso il basso una volta raggiunto il limite dell'area; inoltre se il drone arriva ad avere l'autonomia sufficiente per tornare al centro di controllo, procede a ritirarsi.

I punti strategici partono dalle coordinate (0,0), e vengono aggiunti quei punti a cui la ronda precedente non può arrivare per via della batteria.

Ogni drone poco prima che passino 5 minuti dalla partenza, invia un segnale di richiesta di copertura, e un altro drone parte sfruttando lo stesso percorso del padre. Questo garantisce la verifica dell'area entro i 5 minuti poiché i droni hanno tutti una velocità costante di 30Km/h.

## 4.2 Algoritmi Utilizzati

```
Class Control_Center{
    public:

    Control_Center():
        determino la dimensione del CC
        stabilisco una connessione con il DB postgres
        creo una nuova sessione nel DB
        inizializzo la griglia che rappresenta l'area

    updateGrid(x, y):
        aggiorno l'ultima visita del punto (x, y) al tempo corrente

    getTimeFromGrid(x, y):
        restituisco il momento dell'ultima visita del punto (x, y)

    executeQuery(query):
        stabilisco un lock sul mutex delle query
        eseguo la query di creazione o modifica
        sblocco il mutex

    getValueQuery(query, row, column):
        prendo il risultato di una query tramite getTuples(query)
        restituisco il valore alla riga row e colonna column
```

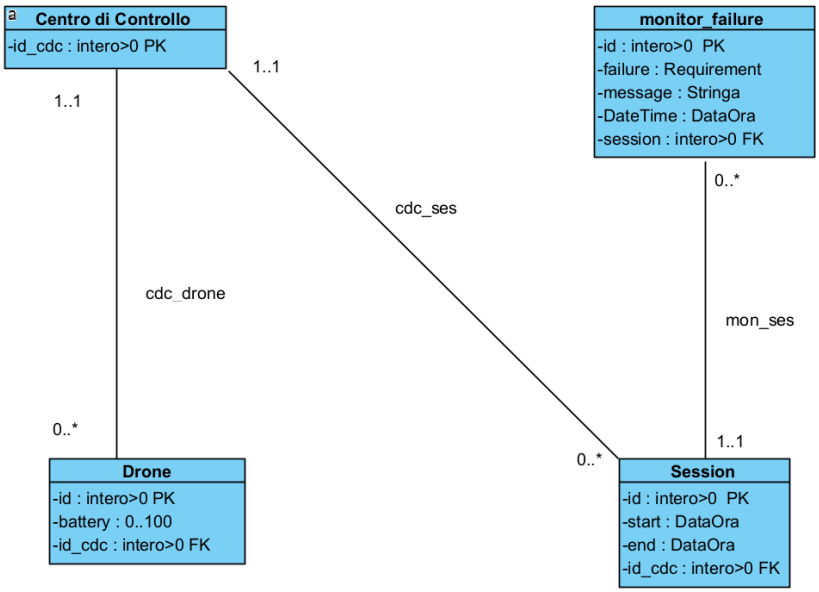
```
    getTuples(query):
        stabilisco un lock sul mutex delle query
        eseguo la query di lettura
        sblocco il mutex
        restituisco il risultato

    getSessionId():
        restituisce l'id della sessione

    getCCId():
        restituisce l'id del CC
}

    goTo(x, y, nx, ny):
        crea una stringa vuota
        aggiunge l se da (x, y) a (nx, ny) serve andare a sinistra
        aggiunge r se da (x, y) a (nx, ny) serve andare a destra
        aggiunge u se da (x, y) a (nx, ny) serve andare su
        aggiunge d se da (x, y) a (nx, ny) serve andare giu
        aggiorna (x, y)
        ripeti finche (x, y) = (nx, ny)
        restituisco la stringa ottenuta

    goToGridX(x, y, step):
        creo una stringa vuota
        procedo andando a destra se y è pari e aggiungo r
        procedo andando a sinistra se y è dispari e aggiungo l
        se mi trovo al limite della riga vado giù e aggiungo d
        decremento di uno step
        ripeto finché step = mosse necessarie per tornare a (150, 150)
        utilizzo goTo() per tornare al punto (150, 150)
        concateno le stringhe in una
        restituisco la stringa

    findPaths(paths, cc):
        inizializzo x, y a 0
        inizializzo un vettore di stringhe
        utilizzo goTo() per posizionare il drone
        utilizzo goToGrid() per calcolare il restante percorso
        aggiungo la stringa ottenuta al vettore
        ripeto fino alla completa visita della griglia
        restituisco il vettore

    doQuery(cc, query):
        esegue cc.executequery(query)
```

```
    listenDronesX(cc):
        ascolta i report dei droni
        preso un report esegue il check di:
        CC_OVERLOAD
        MISSING_REPORT
        AREA_COVERAGE
        ad ogni report aggiorna le informazioni di cc
        termina quando stopflag diventa true

    timer():
        rende stopflag true allo scadere di un tot di secondi

    periodicReport():
        stampa un report di cc periodicamente

Class Drone{
    public

    Drone():
        costruttore drone

    ExecuteMove(char move):
        chiama una delle funzioni relative al movimento

    moveLeft():
        aggiorna il campo x del drone

    moveUp():
        aggiorna il campo y del drone

    moveRight():
        aggiorna il campo x del drone

    moveDown():
        aggiorna il campo y del drone

    fillMovesLeft():
        inizializza il campo moves_left ad una costante

    getRecharginTime():
        restituisce il valore del campo recharging_time

    getMovesLeft():
        restituisce il valore del campo moves_left

    getX():
        restituisce il valore del campo x
```

```
    getY():
        restituisce il valore del campo y

    getId():
        restituisce il valore del campo id
}

    add_to_ready(Drone drone):
        inserisce il drone dato in argomento nei ready

    chargeDrone(Drone):
        simula il processo di ricarica
        chiama fillMovesLeft() sul drone in argomento
        chiama add_to_ready() sul drone in argomento

    getDrone():
        restituisce un drone dal vettore dei ready se ce ne sono
        altrimenti ne restituisce uno nuovo

    initDroneX():
        fa eseguire al drone le mosse descritte dal path ricevuto dal CC
        chiama i droni di copertura ogni 124 mosse
        quando il drone conclude il suo path esegue chargeDrone()
```

## 4.3 Database

Segue una rappresentazione del database implementato in **UML**:



## 4.4 Connessioni Redis

Abbiamo usato gli **stream** di **Redi**s per far comunicare centro di controllo e droni:
- Avviene una prima comunicazione affinché il CC fornisca i percorsi ai droni.
- I droni ad ogni mossa che eseguono, inviano un messaggio contenente la loro nuova posizione.
- Al termine dell'esecuzione il main dei droni fornisce il numero esatto di droni utilizzato al CC.

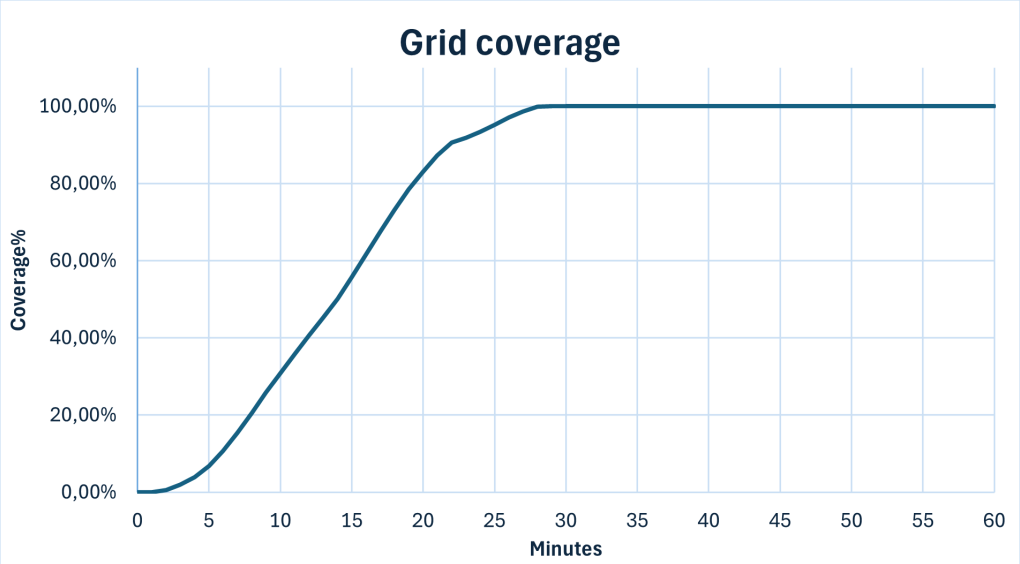
## 4.5 Monitor

Abbiamo implementato 3 monitor di requisiti funzionali e 2 di requisiti non funzionali, i quali eseguono un insert nel database al verificarsi del problema:
- `PATH_CALCULATION`: riferito al requisito di sistema funzionale I.
- `MISSING_REPORT`: riferito al requisito di sistema funzionale III.
- `AREA_COVERAGE`: riferito al requisito di sistema funzionale IV.
- `NUM_DRONES`: riferito al requisito di sistema non funzionale I.
- `CC_OVERLOAD`: riferito al requisito di sistema non funzionale IV.

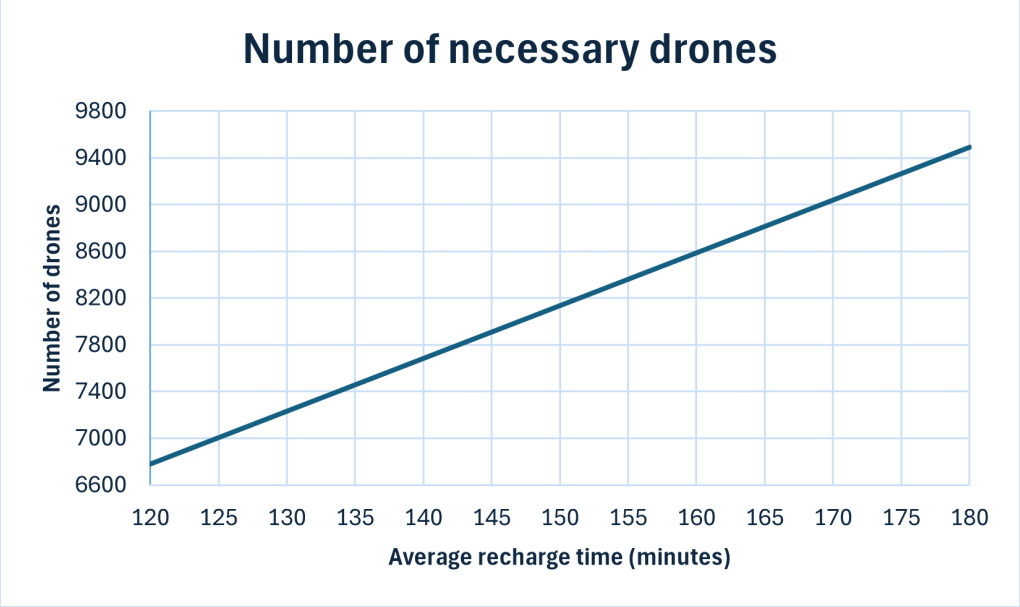
# 5 Test e Conclusioni

## 5.1 Test

Al termine dell'implementazione è stato fatto un test volto a determinare il tempo che impiegano i droni per fornire una **copertura massimale** dell'area, e per determinare se tale risultato viene effettivamente mantenuto nel tempo.
Il test (durato 12 ore) ha prodotto i seguenti risultati:




Un ulteriore test svolto ha consentito di individuare i **droni necessari** per il raggiungimento di uno stato di **riciclo perpetuo**, in base al tempo medio di ricarica:



## 5.2 Conclusioni

L'implementazione del progetto **Sky Watch** ha permesso di raggiungere gli obiettivi preposti.

I test effettuati hanno confermato che il sistema è in grado di raggiungere una copertura massimale dell'area e mantanerla per tutta la durata della sessione.
I test dalle 12 alle 24 ore hanno inoltre anche dimostrato la tenuta del sistema nel lungo termine.
Infine il calcolo a priori dei percorsi minimizza il numero di droni necessari senza compromettere la continuità operativa.

I requisiti sia funzionali che non funzionali sono stati soddisfatti, evidenziando l'efficacia della strategia di implementazione adottata.