Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Sviluppo di Sistemi di Controllo di uno Swarm di Droni Utilizzando il Protocollo MAVLink

Relatore:
Prof.
Angelo Trotta

Presentata da:
Robert Vicentiu Burduja

Correlatore:
Prof.
Federico Montori

Sessione II
Anno Accademico 2023/2024

Alla mia famiglia e alla mia compagna:
grazie per il supporto
e l'amore

Abstract

Questa tesi propone un progetto che permette la generalizzazione dell'utilizzo degli Unmanned Aerial Vechicles (UAVs). L'attuale stato del mercato dei droni è composto da numerose aziende che mettono a disposizione software proprietari e non. Questi software permettono sì l'uso programmatico di UAV, tuttavia, molto spesso sono closed source [3] oppure compatibili solo con i droni prodotti dall'azienda stessa [12].

L'obiettivo di questo progetto di tesi è la costruzione di un engine per semplificare e rendere accessibile a tutti il controllo di uno swarm di droni. In particolare, la programmazione del movimento e la gestione dallo swarm dovrebbe essere comparabile a quella dello sviluppo di una scena in un videogame engine. Questa filosofia di design rende l'utilizzo di swarm di droni accessibile a più persone, permettendo loro di innovare il settore ed inventare nuovi impieghi per questa tecnologia.

Il caso d'uso alla base dello sviluppo del progetto riguarda l'utilizzo di tale engine per costruire una demo od un proof of concept, oppure per testare un algoritmo per un qualsiasi utilizzo di un gruppo di droni; tutto questo in modo semplice, veloce, rapido da correggere e ripetibile.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Gli UAV considerati da questa tesi sono quei droni caratterizzati dalla presenza di rotori per il movimento. A differenza dei droni più simili ad aeroplani dai quali nascono, questi si comportano invece come elicotteri con quattro o più motori elettrici che li rendono in grado di librarsi nell'aria e rimanere sospesi in volo sul posto. Come per molte altre tecnologie i droni hanno origine in campo militare e si sono successivamente diffusi anche in campo industriale fino a diventare economicamente accessibili ai consumatori.

Secondo un report del 2022[6] il mercato degli UAV ha raggiunto il valore di 22.4 miliardi di dollari. Ormai i droni hanno numerosi utilizzi possibili e correnti che vanno dalla fotogrammetria[18][23] al monitoraggio di eventi naturali[21][19] e punti di interesse[20], dalla fotografia e ripresa video professionale al semplice uso per fare dei selfie[5].

Questo progetto è stato pensato per diffondere e semplificare l'utilizzo dei droni, aumentandone ulteriormente l'accessibilità attraverso la riduzione della barriera tecnica. L'obiettivo è proprio di rendere la gestione di uno swarm molto più simile a quella di oggetti in un video game engine: il movimento e la posizione devono essere basati su un sistema semplice, come gli assi x, y e z di un piano tridimensionale.

Per costruire l'engine, denominato Heli, sono state prese in considerazione

varie tecnologie esistenti, come l'open Software Development Kit (oSDK) dell'azienda francese Parrot ed il protocollo di comunicazione MAVLink; queste sono le possibili basi che decidono come e con quali droni potrà comunicare l'engine.

Anche la scelta del linguaggio è vincolante; tra i candidati abbiamo Python e C++. Il primo è diffuso anche tra programmatori meno esperti ed è semplice da utilizzare, il secondo invece è conosciuto per essere un linguaggio di basso livello molto performante.

Tra le altre tecnologie utilizzate in questa tesi è presente l'Android SDK. Dato che il caso d'uso principale dell'engine prevede la creazione di moduli per la gestione di uno swarm, è necessaria un'app per controllarne l'esecuzione.

Nel capitolo successivo verranno esplorati più a fondo gli strumenti utilizzati, e come questi hanno influenzato lo sviluppo di Heli dal punto di vista tecnico. L'architettura generale dell'engine, tutte le sue parti ed infine il modo in cui comunicano verranno esposte nel capitolo 3. Alcuni dei dettagli implementativi più interessanti e le loro motivazioni saranno spiegati nel capitolo 4. Nel capitolo 5 verrà discusso un test di validazione programmato tramite l'engine stesso sotto forma di modulo. Infine verranno tratte le conclusioni e discussi futuri realistici miglioramenti per l'engine nel capitolo 6.

Capitolo 2

Related works

Il progetto realizzato per soddisfare le necessità di questa tesi deve permettere il controllo di uno swarm di droni, che è il passo successivo e più naturale all'utilizzo di un solo drone. Di seguito vedremo come questa scelta rende necessario l'utilizzo di un autopilota e l'integrazione di una GCS. I droni saranno gestibili tramite applicativi chiamati moduli, per cui le loro azioni devono essere programmabili; inoltre devono comunicare con la GCS, e per farlo è necessario un protocollo abbastanza diffuso ed utilizzato in prodotti moderni. Per quanto riguarda l'effettiva esecuzione di Heli abbiamo optato per l'utilizzo di droni virtuali tramite simulazione del firmware e di una componente che si occupa di fisica e grafica.

2.1 Swarm di droni

I droni vengono usualmente utilizzati come un processore single-core senza supporto per più thread, per eseguire un unico lavoro alla volta. Esistono tuttavia compiti che si possono svolgere in parallelo e, dato che un rapido cambio di thread non è adatto ad un drone, è più semplice dividere il compito con più UAV come un processore multi-core.

The Design Challenges of Drone Swarm Control [24] è uno studio che discute come uno swarm possa essere controllato da un unico utente, dato

che i droni da controllare sono molteplici. In particolare, vengono proposti due metodi di input: diretto ed indiretto. Il primo si riferisce ad un controllo diretto di uno o più droni da parte dell'utente in tempo reale, con o senza altri droni che seguono un leader od una formazione per eseguire un compito con un livello di autonomia definito a priori. Il secondo metodo è quello indiretto, in cui i droni eseguono autonomamente un compito predeterminato. Lo studio conclude che entrambi i metodi sono validi per casistiche di utilizzo differenti e finiscono per essere complementari. Il metodo diretto è ottimo per eseguire compiti a breve termine con diversi imprevisti ed input necessari, mentre il metodo indiretto è più adatto per compiti a lungo termine, come per esempio l'esecuzione della fotogrammetria di un'area particolarmente estesa, in cui l'input richiesto all'utente è minimo.

Per quanto riguarda l'engine sviluppato per questo progetto è stato scelto di non escludere un metodo piuttosto che un altro. È quindi parte degli obbiettivi rendere possibile la costruzione di moduli per l'esecuzione di lavori autonomi e moduli che danno il controllo all'utente per input continui. Così facendo si potrebbero anche creare moduli ibridi, in cui l'esecuzione può essere inizialmente diretta, ed indiretta in un secondo momento, o viceversa. Il test di validazione descritto nel capitolo 5 è di tipo ibrido.

2.2 Autopiloti e Ardupilot

Che si scelga di controllare i droni tramite un metodo diretto o indiretto, i droni devono avere un certo grado di autonomia e fare scelte in base alle richieste dell'utente, ed è proprio questo il ruolo ricoperto dagli autopiloti. Un autopilota è un software che segue determinati protocolli di comunicazione come MAVLink e garantisce funzionalità di autonomia la cui assenza renderebbe l'utilizzo, anche di un singolo drone, particolarmente complesso. Autopiloti come Ardupilot [1] e PX4 [14] (entrambi open source) infatti si occupano di:

- Inizializzare i sistemi dei droni sui quali sono installati.

- Integrare tutti i sensori e e salvare ed inviare i dati rilevati.
- Controllare la navigazione del drone.
- Decidere interamente in autonomia la spinta da dare ai rotori per aumentare e diminuire l'altitudine del drone in sicurezza, nel caso di comandi come atterraggio o decollo.
- Mantenere la posizione del veicolo sul posto in attesa di nuovi comandi, anche in condizioni di instabilità e forte vento.
- Riportare i droni al sito del decollo (cosiddetta Home Position) nel caso in cui la batteria sia scarica oppure in caso di uscita dalla zona d'azione.

Per questo progetto di tesi i droni simulati sono controllati ciascuno da un'istanza di Ardupilot Copter. In futuro è prevista l'aggiunta di compatibilità con PX4, avvantaggiata dal fatto che utilizzano lo stesso protocollo di comunicazione GCS-drone.

2.3 Ground Control Station e le loro funzionalità

Le GCS sono software che si occupano di pianificare missioni e controllare i droni dall'esterno. I dati rilevati dagli autopiloti vengono pubblicati sulla rete, e sono le GCS a rappresentare il lato ricevente delle comunicazioni. Allo stesso modo, i movimenti di uno o più droni possono essere cambiati, facendo richiesta attraverso la GCS.

Il vantaggio che si ha nell'utilizzare una Ground Control Station per il movimento dei droni piuttosto che far comunicare i droni tra loro viene discusso in Ground Control System Based Routing for Reliable and Efficient Multi-Drone Control System[22]. In particolare, il lavoro citato conclude che la gestione dei droni basata su una GCS permette la costruzione di network affidabili ed efficienti per sistemi di controllo multi drone; perciò sono state

considerate alcune GCS da integrare all'interno di Heli; tra le più conosciute abbiamo:

- DJI Ground Station Pro; permette l'utilizzo gratuito di alcune funzionalità e a pagamernto di altre[4].
- QGroundControl è una GCS open source compatibile con MAVLink [15].
- MAVProxy Developer GCS è un'altra GCS che presenta un'interfaccia a riga di comando utilizzata insieme al simulatore SITL [9].

Delle GCS elencate, nessuna permette la programmazione in codice di missioni, nonché cambiamenti drastici degli obbiettivi di uno e soprattutto di più droni in contemporanea (più droni con compiti diversi). Il nostro progetto mira a dare più libertà di sviluppo e creatività possibile all'utente. È quindi decisa l'integrazione di una GCS all'interno di Heli; tuttavia questa sarà implementata da zero. Anche se la GCS integrata disporrà di capacità minori rispetto a quelle elencate per alcuni aspetti, permetterà il rapido cambio di compiti dello swarm tipico del controllo diretto, in modo da rispettare i requisiti posti in precedenza.

2.4 Programmabilità e Parrot oSDK

Il paragone fatto per gli swarm ed i processori multi-core non è del tutto corretto. I compiti non si possono cambiare rapidamente, a meno che non sia un comportamento previsto a priori e programmato come reazione in base ad un determinato input od evento. Con una struttura di questo genere gli swarm di droni possono assumere comportamenti tipici del controllo indiretto, ricevere un qualche input per andare in controllo diretto (ottenendo così un controllo ibrido), ed infine tornare all'esecuzione dello stesso controllo indiretto, come succede per un cambio di thread di esecuzione.

Per permettere ciò, Heli prevede l'utilizzo di una companion Android App, dalla quale può ricevere vari input, e sulla quale è possibile controllare l'esecuzione dei moduli; questo sistema funziona solo se le azioni da far compiere ai droni sono previste al momento della programmazione del modulo nell'engine. L'open Software Development Kit dell'azienda francese Parrot offre già alcune di queste funzionalità. In particolare, Parrot mette a disposizione le seguenti opzioni di nostro interesse:

- Ground SDK[10], una GCS direttamente per cellulare programmabile in Java.

- Olympe SDK[11], che permette di programmare in Python missioni per i droni ed è potente a sufficienza da essere utilizzato come base per la costruzione di una GCS.

Il motivo principale per cui il progetto proposto non si basa su Olympe SDK è il susseguente obbligo di utilizzare solo i droni Parrot una volta completato l'engine. Il Ground SDK invece rappresenterebbe una forte limitazione di espressività: la produzione di un'app che permette l'esecuzione di un solo compito per uno swarm è paragonabile alla programmazione di un solo modulo nel progetto proposto in questa tesi, che invece vuole promuovere la creazione di moduli multipli da eseguire in qualsiasi momento. Inoltre, con Ground SDK, per modificare il compito nell'app od aggiungerne uno nuovo bisognerebbe programmarlo in Java ed eseguire un aggiornamento; in Heli invece la proposta è quella di lavorare solamente nell'engine e fare un sync veloce con il cellulare. Questa differenza diventa importante se si pensa ad un eventuale utilizzo di droni veri e si considera la rapidità di correzione dei bug del codice di esecuzione.

2.5 Micro Air Vehicle Communication Protocol

MAVLink[7] è un protocollo di comunicazione open source in utilizzo su molti velivoli e integrato in molti SDK come il precedentemente discusso Parrot Olympe.

Mavlink rappresenta una serie di messaggi strutturati di invio e conferma di ricezione inter droni e GCS-drone. Il protocollo espande la comunicazione anche su velivoli ad ala fissa, veicoli da terra, ed è persino utilizzabile per guidare droni subacquei; nel nostro caso l'utilizzo sarà quello dei velivoli con rotori definiti come Copter. I messaggi sono definiti per un ampio numero di linguaggi: Python, C++, C, C# sono solo alcuni di questi. Esistono vari studi sulla reale efficienza di MAVlink che ne analizzano rapidità e rateo di perdita di pacchetti, come in Data analysis of the MAVLink communication protocol[17]. Secondo i test eseguiti, la perdita di pacchetti si aggira attorno al 0.63% per una distanza GCS-Drone di 20 metri; per la stessa distanza la latenza di invio e ricezione misurata è di 0.42 secondi.

Del protocollo MAVLink esistono due versioni numerate, 1.0 e 2.0. La versione 2.0 è retro compatibile e permette nuove funzionalità come signing dei pacchetti, estensione di alcuni messaggi 1.0 e message ID per l'enumerazione dei pacchetti. Entrambe le versioni prevedono i seguenti metodi di comunicazione:

- Topic Mode, di categoria publish-subscribe, viene utilizzata per pubblicare dati pertinenti all'atteggiamento e posizione del drone.
- Point to Point Mode invece richiede l'uso di un target ID per rappresentare un drone, e di un campo target component per controllarne un preciso sistema, come la telecamera od il comportamento dei rotori.

Per il caso d'uso di questa tesi la versione 1.0 è sufficiente. MAVLink verrà quindi parzialmente implementato per la comunicazione tra lo swarm e la GCS integrata.

2.6 Simulatori

Quando si parla di simulatori bisogna tenere in considerazione le scelte fatte in precedenza, come quelle descritte nelle sezioni precedenti per ragioni di compatibilità.

La strada inizialmente intrapresa per questo progetto di tesi ha puntato all'uso dei software ed api Parrot. Oltre a vari sdk infatti la Parrot mette anche a disposizione degli acquirenti dei propri droni un sistema di simulazione chiamato Sphinx [13]. Sphinx si occupa di rappresentare il normale comportamento dell'autopilota presente sui droni dell'azienda; per la grafica, Parrot Sphinx permette l'utilizzo di uno di due software compatibili:

- Unreal Engine 4.26, famoso programma per lo sviluppo di videogiochi, CGI, e simulazioni di qualsiasi genere. Al momento esiste una nuova versione 5.5 [16] non supportata da Sphinx.

- Gazebo 11, simulatore open source altrettanto capace ma più indirizzato verso il testing e lo sviluppo di nuove tecnologie.

Come già discusso, la scelta non è ricaduta sull'utilizzo di Sphinx nonostante sia molto semplice da utilizzare in congiunzione con gli altri sdk offerti dalla Parrot, data la scelta precedente di non utilizzare tali software. Questa piccola deviazione non è tuttavia stata inutile, dato che Gazebo è diventato il principale candidato per la simulazione dei droni. Gazebo infatti è in grado di espandere la propria compatibilità con l'installazione di vari plugin; tra questi ne esiste uno per l'utilizzo di Ardupilot SITL [2]. I due software funzionano insieme: Gazebo si occupa della grafica con le librerie OpenGL e della fisica con il motore ODE, mentre Ardupilot SITL simula l'autopilota installato su un drone. Con una breve ricerca è inoltre facile scoprire come simulare molteplici droni per lavorare con uno swarm.

Tutti i software e tecnologie moderne descritti in questo capitolo sono necessari per questa tesi. Alcuni, come la GCS, sono da costruire da zero

per essere integrati, altri invece sono esterni di supporto, come Gazebo e Ardupilot SITL. Nel prossimo capitolo vedremo l'architettura del progetto sviluppato e la divisione dei ruoli dei vari software qui descritti.

Capitolo 3

Architettura del framework

Teniamo a mente che il progetto deve essere sviluppato con l'obbiettivo di mettere a disposizione dell'utente un modo semplice e veloce di creare degli applicativi per l'utilizzo di più droni. Da qui in poi gli applicativi prenderanno il nome di moduli.

Di seguito viene esposta l'architettura generale del progetto ed il ruolo che hanno le sue varie componenti.

3.1 Vista generale

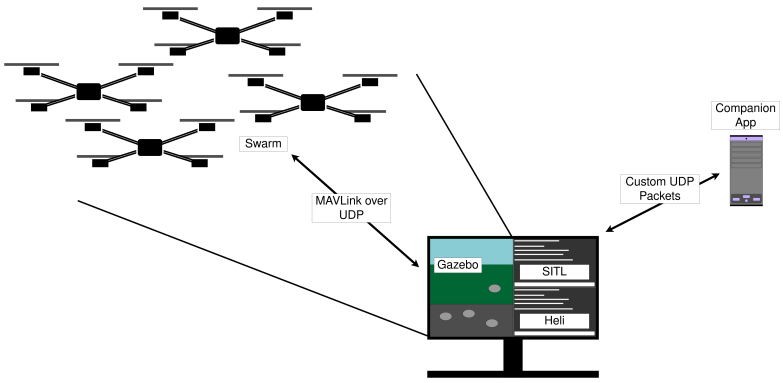

Figura 3.1: Architettura generale del progetto

In figura 3.1 è mostrato l'ambiente di lavoro, diviso nelle seguenti parti:

- Heli, l'engine che permette la creazione dei moduli da far eseguire ai droni tramite la GCS integrata. Si può dire che l'engine faccia da "colla" per tenere insieme ed in armonia tutti i sistemi: si occupa di comunicazioni interne ed esterne, tiene aggiornati i dati rilevati dai droni e le loro posizioni e li mette a disposizione dei moduli. Infine controlla gli input ricevuti dalla companion app.

- La companion app sviluppata con Android SDK è sempre in comunicazione con Heli tramite pacchetti UDP. L'app mostra all'utente la lista dei moduli implementati all'interno dell'engine e ne rende possibile l'utilizzo in una prima fase di setup, e successivamente di running ed infine pause. Dispone inoltre di supporto per controller fisici, in alternativa al controllo virtuale in-app, utili per cambiare velocemente il funzionamento di un modulo che ne supporta l'input.

- Gazebo e SITL simulano lo swarm di droni. La simulazione è in corso sullo stesso sistema su cui viene utilizzato l'engine: è possibile eseguire Heli su un sistema terzo all'interno della stessa LAN, ma l'avvio di tutte le parti risulta essere un processo più lungo rispetto all'utilizzo totale sulla stessa macchina.

- Swarm simulato in comunicazione con Heli tramite messaggi MAVLink [8] attraverso porte UDP. L'aggiunta di più droni alla simulazione è semplice: è sufficiente duplicare il modello di un drone esistente, assegnargli porte di comunicazione in ingresso ed uscita differenti, ed infine aumentare il valore del parametro *-I*, il quale rappresenta l'id del drone all'interno del SITL.

3.2 I moduli

I moduli sono delle applicazioni molto leggere divise in tre parti, per permettere a chi li programma il massimo dell'espressività.

3.2.1 Back end

Prima ancora di iniziare la simulazione di uno swarm l'utente deve poter già lavorare su di un modulo. Al momento, per creare un modulo è sufficiente un qualsiasi IDE, preferibilmente con supporto per il linguaggio Python. Partendo da una semplice idea, come per esempio far spostare il gruppo di droni di 100 metri in avanti (che corrisponde al Nord), è sufficiente descrivere:

1. Alcuni dati, come titolo e spiegazione di come si usa il modulo.
2. Quali input richiedere all'utente dell'app al momento della simulazione.
3. Cosa fare dei dati ricevuti in risposta.
4. Cosa succede una volta fatto partire il modulo; qui l'utente può utilizzare la funzione di movimento per spostare uno o più droni e determinare come processare l'utilizzo del controller.
5. Cosa fare quando l'engine richiede al modulo di andare in pausa.
6. Come eseguire il reset dei dati del modulo.

3.2.2 Front end

Una volta definito il modulo all'interno dell'engine secondo le modalità appena descritte l'utente può immediatamente testarlo tramite l'Android app. Quando tutti i droni sono in volo ed un rapido sync tra l'engine ed il cellulare viene eseguito, il modulo diventa disponibile ed utilizzabile. Vedremo più in dettaglio il funzionamento dell'app nella sezione 3.4

3.2.3 Esecuzione

Per quanto riguarda le azioni ed il movimento dei droni, questi vengono controllati sempre da Heli. Il modulo front end, quindi dal cellulare, manda una richiesta all'engine, che determina cosa va fatto passando il controllo all'azione corrispondente nel modulo back end. Il modulo decide come agire

sui droni nella propria funzione *run()*. La comunicazione con i droni e la loro organizzazione avviene nella GCS integrata, la quale si si occupa di gestire i droni come richiesto dal modulo; invia quindi un messaggio MAVLink strutturato in direzione della simulazione. Per quanto riguarda il debugging di un determinato modulo, è sufficiente:

1. Fermare l'esecuzione dell'engine, ma non necessariamente l'esecuzione della simulazione o dell'applicazione sul cellulare.
2. Modificare il modulo per correggere gli errori.
3. Avviare nuovamente l'engine ed eseguire il sync (un solo tasto sull'app) con il cellulare.
4. Continuare il lavoro da completare.

3.3 L'engine Heli

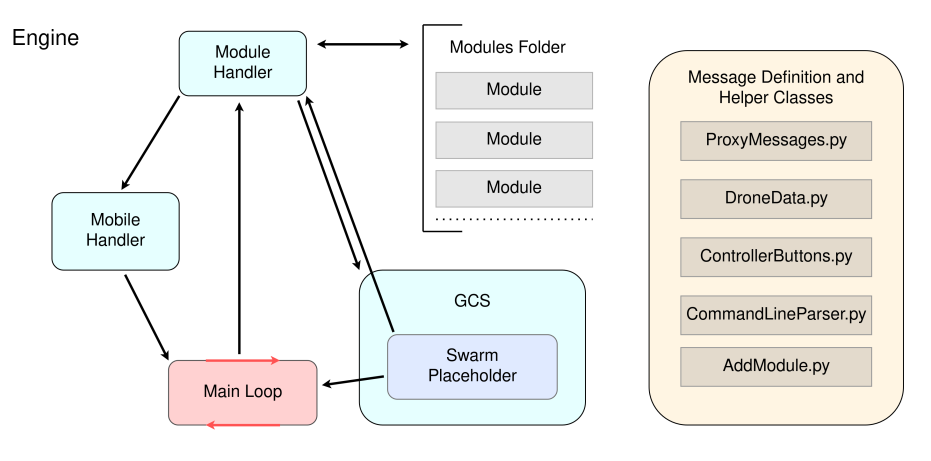

Figura 3.2: Architettura dell'engine

Per ottenere i sistemi di back end ed esecuzione dei moduli appena descritti è stato necessario costruire un engine apposito. L'engine, scritto in

Python, è suddiviso tra vari file, ed i compiti da svolgere sono stati distribuiti tra varie classi.

Nella figura 3.2 vediamo come l'architettura risponda agli input dei droni, del cellulare ed ai parametri scelti al suo avvio a riga di comando. Le frecce indicano chiamate di funzioni, caricamenti in liste e pubblicazione di dati verso gli altri componenti.

3.3.1 Message Definition e Helper Class

Nella sezione destra della figura 3.2 è presente un gruppo di classi e file che, nonostante non sia connesso al resto dell'engine, ha un ruolo centrale ed è di accesso globale. I compiti svolti da questi file sono, in ordine di figura:

1. Definire enumerazioni che descrivono l'attuale comportamento in modo sintetico di un singolo drone o dello swarm per intero, in base ai dati ricevuti dalla GCS; definire inoltre le classi di rappresentazione dei dati di invio e ricezione dei comandi da e verso il cellulare.

2. Definire classi in cui vengono salvati gli effettivi dati di posizione numerici sui quali si basano le enumerazioni del punto 1.

3. Definire l'enumerazione di codifica dei tasti del controller ricevuti dal cellulare e quale valore rappresentano.

4. Controllare l'input a riga di comando per l'avvio dell'engine: i dati richiesti sono il numero di droni e l'indirizzo ipv4 locale del cellulare, senza i quali l'engine non funzionerebbe come previsto.

5. Permettere l'aggiunta del proprio modulo, una volta definito secondo i criteri richiesti dalla classe omonima presente in Module.py della sezione successiva.

3.3.2 Cartella dei moduli

La cartella dei moduli è popolata dall'utente finale. Una volta scaricato l'engine è già presente un modulo non eseguibile al suo interno, che non apparirà nel front end, discusso in 3.2.2. Il modulo in questione è definito come estensione di *ABC*, ed è la classe da estendere a sua volta per la creazione di un nuovo modulo. Contiene infatti tutte le funzioni ed i dettagli necessari per costruirne uno.

3.3.3 GCS e Swarm Placeholder

Swarm Placeholder è rappresentato in figura come facente parte della GCS. Questa scelta strutturale porta due vantaggi:

- Allocazione dei dati. I dati inviati dai droni e ricevuti dalla GCS vanno organizzati per essere resi visibili al modulo attualmente in esecuzione ed al Main Loop. Il metodo scelto per rendere possibile questa meccanica è istanziare nel Swarm Placeholder delle classi Drone, tante per quanti droni vanno utilizzati, ed una classe Swarm per dati che riguardano il gruppo. Queste istanziano a loro volta strutture definite in DroneData.py e ProxyMessages.py.

- Creazione di una separazione tra le azioni che un modulo può e non può eseguire sui droni. In particolare un modulo può fare richieste di spostamento ma non richieste di atterraggio o decollo: queste ultime sono un potere esclusivo della GCS.

La GCS, oltre ad occuparsi dell'atterraggio e decollo, attende su un thread spearato la ricezione dei dati dei droni per distribuirli all'interno dello Swarm Placeholder.

3.3.4 Main Loop e Handlers

Dei due handler, quello del cellulare è l'unico che riporta dati al Main Loop. Il Mobile handler attende input dall'app su un thread differente e li

invia a sua volta al Main Loop. Si occupa inoltre di mantenere la connessione attiva con il telefono su un thread separato; in più, su richiesta del Module handler, invia dati di risposta da mostrare all'utente come Toast nell'app.

Il resto dell'esecuzione ricade sul Main Loop e sul Module Handler:

- Il primo riceve dal Mobile Handler richieste di utilizzo di un modulo, di atterraggio, di decollo, di sync con la GCS e lo stato della connessione. Riceve anche dalla GCS un riassunto dello stato dei droni e la loro connessione. Una volta raccolti tutti i dati li processa e passa un comando al Module Handler.

- Il secondo, ricevuto il comando lo esegue in base alla natura del comando stesso. Alcuni ricadono sotto la categoria *super* e non hanno nulla a che fare con i moduli. Vengono quindi eseguiti direttamente richiedendo al Mobile Handler l'invio di dati al cellulare oppure alla GCS decolli ed atterraggi; l'implementazione dei comandi viene spiegata più a fondo nel capitolo 4. Il resto dei comandi invece dipende dal modulo attualmente in utilizzo. Il Module handler raccoglie i dati riguardanti i droni da Swarm Placeholder e cede a sua volta il controllo dell'esecuzione al modulo per le fasi di setup, run e pause.

3.4 La Companion App

L'applicazione è costruita con l'Android SDK ed è basata su un'architettura a single activity chiamata Main Activity. All'interno di questa Main Activity è presente un Fragment Container per rendere interattiva una di quattro schermate alla volta: Settings, Module List View, Module View e Controller View. Per la condivisione di variabili di interesse globale viene sfruttato uno Shared View Model, sempre implementato tramite l'SDK. Infine, per aggiungere ulteriore funzionalità, è presente un Foreground Service, chiamato Keep Alive Service, il cui compito principale è mantenere autonomamente la connessione con l'engine anche ad applicazione chiusa. Difatti, la

sezione colorata di lilla nella figura 3.3 racchiude le componenti dell'app che non terminano automaticamente l'esecuzione né vengono resettate all'uscita dell'utente dalla UI dell'applicazione.

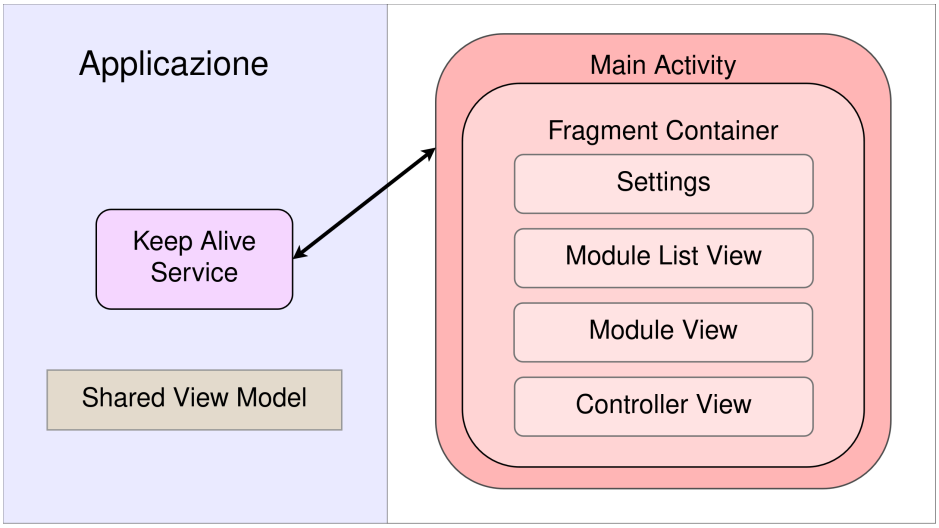

Figura 3.3: Architettura dell'applicazione

3.4.1 Keep Alive Service

L'esecuzione del servizio non è direttamente visibile all'utente. Nonostante ciò, l'utente può vedere come lo stato della connessione mantenuto in background cambi nell'interfaccia utente (UI) della Main Activity. È solo con la Main Activity infatti che il servizio va a comunicare tramite broadcast espliciti.

L'obbiettivo secondario del service, oltre al mantenimento della connessione, è quello di far presente all'engine che l'app è ancora in utilizzo. In tal modo, l'engine determina che un eventuale modulo ancora in esecuzione può continuare a comandare i droni. Una volta chiuso il service dalle notifiche o bloccata l'esecuzione totale dell'app dal task manager di Android, l'engine attende 7 secondi per una possibile riconnessione, dopodiché fa atterrare automaticamente i droni. Questa funzionalità è il motivo per cui Keep Alive

non è un semplice Background Service ma viene invece implementato come Foreground Service.

3.4.2 Main Activity

La Main Activity viene mostrata nella figura 3.3 come contenitore per i vari fragment. Nonostante ciò sia vero dal punto di vista dell'interfaccia utente, nessun processo legato ai quattro fragment può comunicarvi direttamente. Proprio per questo motivo lo Shared View Model è parte dell'architettura dell'app, permettendo la creazione di variabili condivise sia alla Main Activity che ai fragment. Per funzioni che necessitano accesso a sensori, come per la Controller View, è stato invece scelto di sviluppare varie *interface* e di farle implementare alla Main Activity. Così facendo i fragment sono in grado di richiedere alla Main Activity di eseguire del codice per conto loro ed espandere le proprie funzionalità.

Dal punto di vista della UI, la Main Activity mette a disposizione la toolbar in cima alla schermata, come in figura 3.4. Sempre presente in tutta la navigazione dell'app, la barra contiene il tasto dedicato alla richiesta di decollo dei droni ed un'icona che cambia in base ad uno dei tre stati di connessione con l'engine: connesso, in fase di connessione e non connesso. Cliccando sui tre puntini a destra appare un menù con altre funzionalità come l'atterraggio oppure l'accesso alla schermata delle impostazioni.

3.4.3 I Fragment

Dei quattro fragment presenti nell'app vedremo ora i tre più importanti. La vista delle impostazioni serve solo per impostare l'indirizzo locale ipv4 dell'engine e controllare l'indirizzo del cellulare stesso.

Module List View

Parte degli obiettivi del progetto è dare all'utente modo di controllare lo stato del traffico nella zona in cui intende utilizzare l'engine. Questa funzione

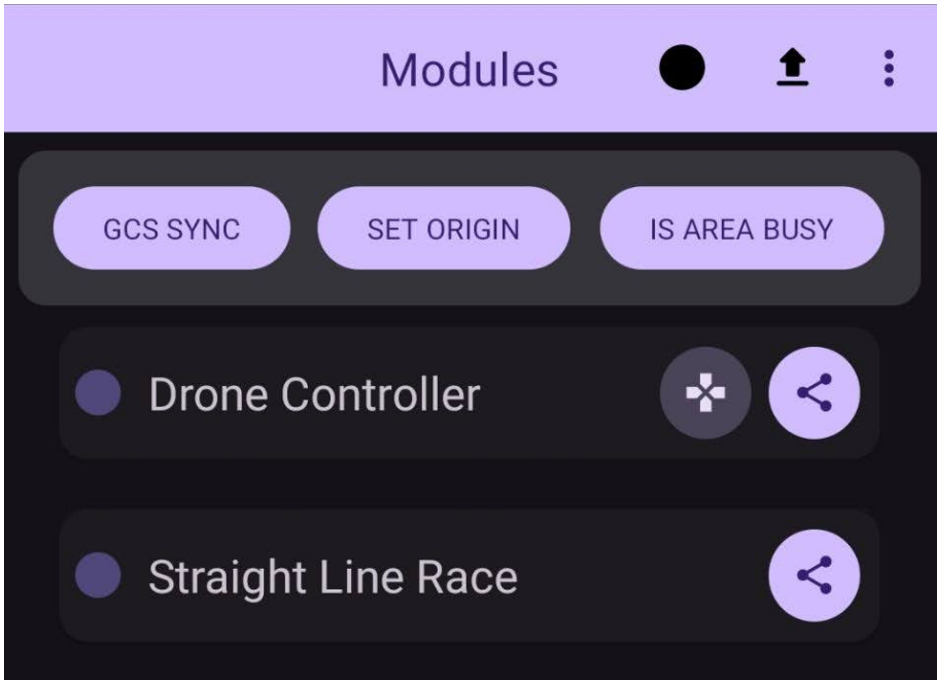

Figura 3.4: Schermata con la lista dei moduli

è a disposizione sotto forma di tasto (*IS AREA BUSY* in figura 3.4) nella Module List View. Il fragment richiede il numero di access point presenti attraverso la Main Activity e ne presenta la quantità tramite Toast.

La Module List View inoltre permette di inviare la posizione del cellulare all'engine tramite il tasto *SET ORIGIN*. Con quest'informazione l'engine in futuro potrà decidere dove si trova l'origine del piano su cui si basano gli spostamenti dei droni.

Sempre in questa schermata sono mostrati a lista i front end dei moduli eseguibili dall'engine. L'utente può vedere il titolo di un modulo, se è implementato all'interno di Heli attraverso un cerchietto viola (meglio discusso nel capitolo 6), e se il modulo prevede l'utilizzo del controller tramite l'icona del dpad.

Module View

La schermata di controllo di un modulo è il front end come descritto nella sezione 3.2.2. La schermata presenta direttamente il titolo, ed una side-sheet laterale attivabile premendo il tasto *?*, che mostra la descrizione del modulo. Questo comportamento è visibile nella figura 3.5.

Quando il modulo prevede un setup, l'input utente viene accolto tramite vari box di testo o switch. Una volta premuto, il tasto *SETUP* invia il contenuto degli input all'engine che li confronta con il back end del modulo. Sia che il setup risulti corretto o meno viene comunicato all'utente un messaggio; in base allo stato degli input il messaggio sarà di successo oppure di richiesta di correzione dei dati inseriti.

Per l'esecuzione del modulo sono previsti altri due comandi, *RUN* e *PAUSE*. Il primo lancia l'esecuzione principale; il secondo è opzionale e dipende dall'implementazione nel back end. Per esempio, il modulo nella figura 3.5 mostra il tasto di pausa disattivato, perché è ad esecuzione singola; questo dettaglio implementativo verrà approfondito ulteriormente nel capitolo 4.

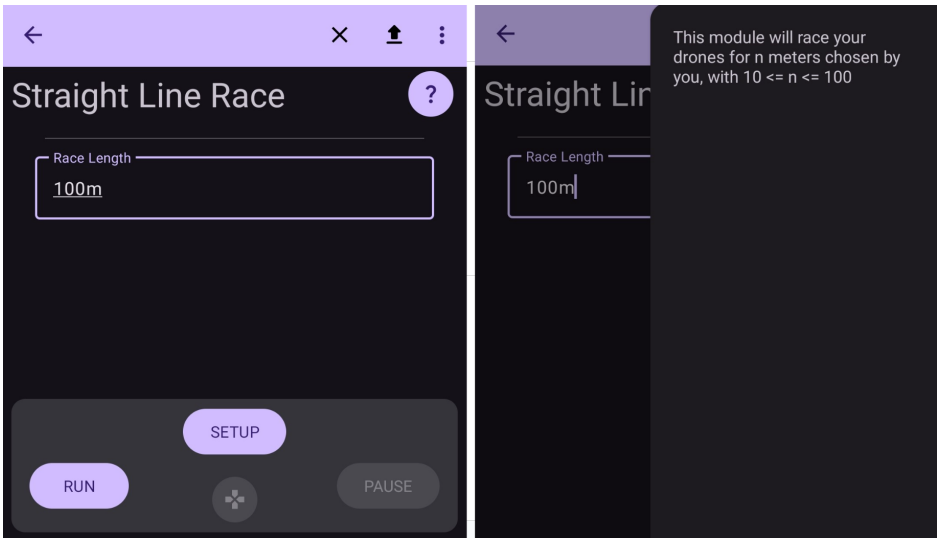

Figura 3.5: Schermata di un Modulo esempio

Controller View

L'ultimo tasto all'interno della Module View nella sezione inferiore apre la schermata del controller virtuale, mostrato in figura 3.6. Nella Controller View sono presenti tutti i tasti di un controller fisico, simulato dall'app che rileva e utilizza i corrispondenti comandi. Dei due joystick, solo il sinistro viene implementato attraverso la rotazione negli assi X ed Y del cellulare. Il destro di default non viene rilevato. I due valori decimali "0.99" e "-0.09" della stessa figura rappresentano il valore della rotazione: vanno da -1 a +1, con zero come valore neutro del dispositivo, perfettamente in piano. Di conseguenza il telefono al momento della cattura della schermata deve essere stato ruotato verso destra, quasi al valore massimo per l'asse X.

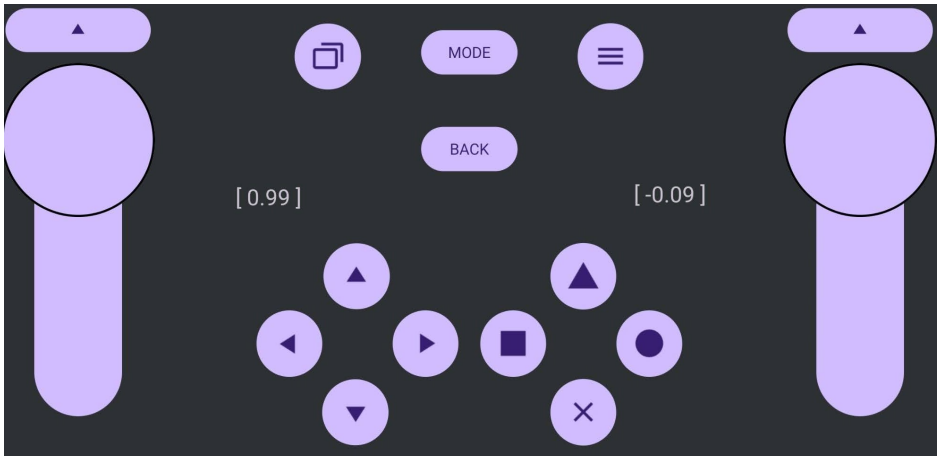

Figura 3.6: Schermata del controller virtuale

Capitolo 4

Implementazione

Nel capitolo precedente abbiamo visto l'architettura generale del progetto e quella specifica delle sue componenti. Vedremo invece adesso come alcune di queste componenti sono implementate, i problemi risolti e gli ostacoli incontrati.

Per prima discuteremo l'implementazione del sistema tridimensionale alla base del movimento e posizionamento dei droni; questo è un punto critico dato che vogliamo evitare l'utilizzo del GPS, e preferiamo piuttosto presentare i droni come su un piano piatto infinito con coordinate x,y e z per semplificare al massimo la programmazione dei moduli.

Successivamente discuteremo l'esecuzione del modulo e la differenza tra run singola e continua; vedremo come viene risolto il problema di gestione dei comandi causato dal fatto che vengono ricevuti su un thread diverso dal principale; infine verrà spiegata l'implementazione dei comandi *super* e perché sono necessari.

4.1 NED e l'origine del piano

Il design del sistema di movimento implementato nell'engine è ispirato a quello di un videogioco 3d. Per poter utilizzare un'unità di misura come i

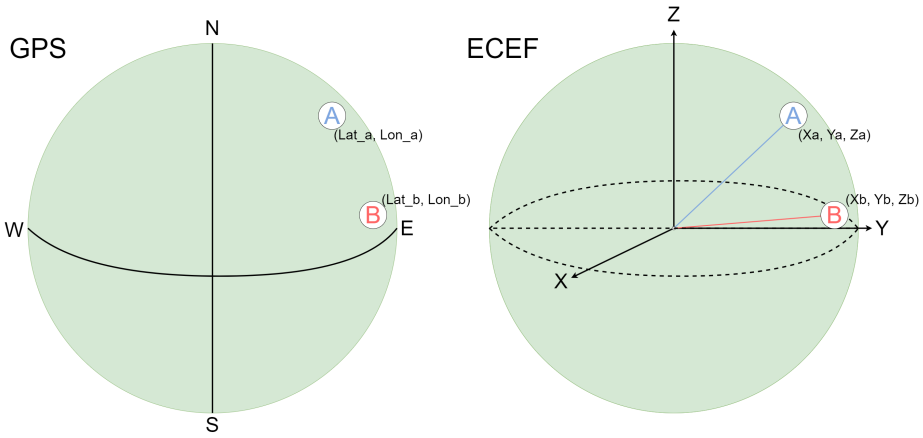

Figura 4.1: Due punti, A e B, espressi in GPS (sinistra) ed ECEF (destra)

metri e assumere che il piano di lavoro sia piatto, il tutto partendo da delle coordinate GPS, va fatta una conversione di sistema di riferimento.

Prendiamo in considerazione il punto A nella figura 4.1 espresso con latitudine e longitudine. Per poter utilizzare un piano 3d ed una distanza come i metri, la posizione di A va convertita nel sistema di riferimento Earth Centered, Earth Fixed (ECEF). La conversione avviene attraverso la libreria Navpy di Python. Come da nome ECEF prende come origine del proprio piano il centro della terra; inoltre, similarmente a GPS, si tratta di un sistema in cui due punti, A e B in figura, sono esprimibili con delle coordinate assolute e non relative ai punti stessi.

Il comportamento desiderato dell'engine deve permettere di programmare con molta semplicità un movimento del tipo "sposta il drone X di 10 metri avanti". La sola conversione a ECEF non è dunque sufficiente: i valori X, Y e Z di un punto sono enormi dato che indicano la distanza dal centro della terra per ogni asse. Viene quindi utilizzato un sistema di riferimento locale al drone, nel nostro caso abbiamo scelto di utilizzare NED.

North East Down o NED soddisfa i requisiti imposti sul progetto. L'implementazione di NED è avvenuta prima in un modo che sarebbe potenzialmente risultato in un errore, poi nel modo corretto che ha tuttavia aggiunto

complessità all'engine. Li vediamo entrambi di seguito.

4.1.1 Origini indipendenti

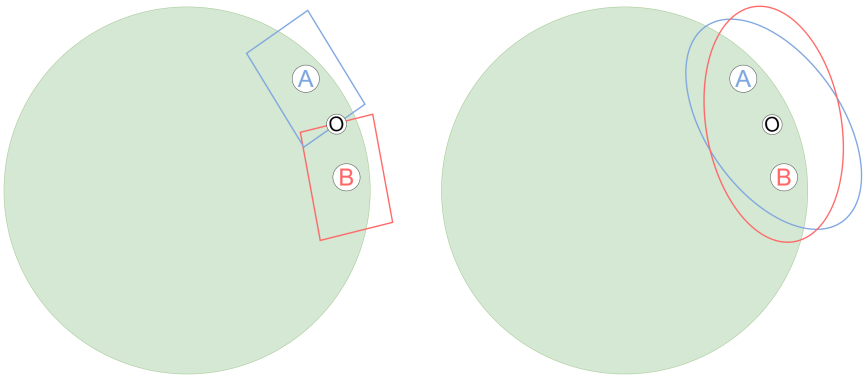

Figura 4.2: Tre punti, A, B ed o. A e B sono origini per il proprio piano tridimensionale dal colore corrispondente; o viene espresso in ciascuno (sinistra). Il movimento errato di due droni con decollo in A e B (destra)

Secondo il protocollo MAVLink le posizioni dei droni si possono richiedere anche nel sistema di riferimento NED, oltre che in coordinate GPS. L'origine del sistema per ciascuno dei droni è il punto *EKF Origin* impostato all'accensione del drone stesso; quindi, per un'eventuale richiesta di movimento, ogni drone agisce sul proprio piano, colorato in modo corrispondente in figura 4.2.

Con questa configurazione vediamo cosa succede se desideriamo far girare i droni in tondo attorno ad un terzo punto o:

1. Impostiamo per ciascun drone la posizione del punto o nel proprio piano. Questo risulta nell'avere due punti o, diciamo oA e oB.

2. Sia il drone A che il drone B ruotano correttamente intorno ad o secondo il loro sistema di riferimento locale NED. Tuttavia la visione di insieme risulta errata, come si può vedere nella stessa figura a destra: I droni percorrono ciascuno una circonferenza (di colore corrispondente) ruotata rispetto a quella dell'altro drone a causa della curvatura della terra.

Questo errore è poco grave, ma solo nel caso in cui il raggio di azione dei droni sia piccolo. Per test di algoritmi che richiedono alti livelli di precisione e prevedono spostamenti dei droni su lunghe distanze, l'engine diventerebbe inutile.

Di seguito vediamo l'implementazione corretta per l'uso del sistema di riferimento NED.

4.1.2 Origine GPS

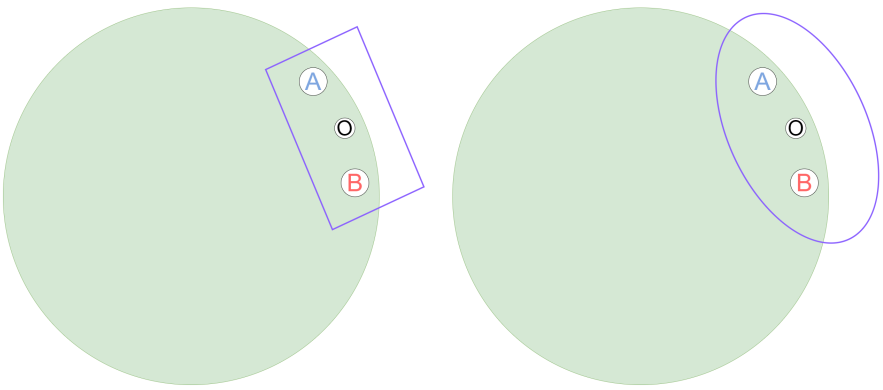

Figura 4.3: Tre punti, A, B ed o. o è l'origine dell'unico piano tridimensionale; A e B vengono espressi nel piano (sinistra). Il movimento corretto di due droni con decollo in A e B (destra)

Per ovviare al problema della doppia origine, che diventerebbe il problema delle n origini dati n droni da spostare, l'implementazione è stata cambiata. È stata presa la decisione di "ignorare" l'utilizzo del sistema di riferimento NED nativo dei droni e si è invece deciso di lavorare in GPS con conversione in-engine. In particolare all'avvio dell'engine un punto GPS viene scelto come origine del piano NED, come in figura 4.3. Il piano è rappresentato a sinistra con il colore viola.

Eseguendo di nuovo la richiesta di movimento dei droni, l'esecuzione avviene come segue:

1. L'utente decide quale punto utilizzare per il centro della circonferenza nel piano NED simulato nell'engine, mantenendo tutti i vantaggi del sistema di riferimento locale.

2. L'engine converte il punto in ECEF ed infine in GPS e invia le coordinate di movimento a ciascun drone; ogni drone seguirà l'unica circonferenza (di colore viola in figura).

Quest'implementazione prevede anche la conversione dei valori di posizione dei droni da GPS a ECEF ed infine NED, per dare all'utente la posizione esatta dei droni all'interno del piano tridimensionale.

4.2 Run singola e continua

L'esecuzione principale di un modulo è stata pensata per due casi di utilizzo: il primo prevede che una volta premuto il tasto *RUN* dal cellulare (figura 3.5) l'engine esegua una sola volta il codice della corrispondente funzione *run()* nella programmazione del modulo; il secondo, invece, tratta il modulo come un loop, quindi sempre su richiesta dall'utente sul cellulare, in questo caso il modulo esegue la propria run continuamente.

È stato pensato un parametro da inserire all'interno del modulo chiamato `enablePause` per permettere quest'implementazione. Il comportamento di base di un modulo è quello della singola esecuzione, ed enablePause è `False` di default. Viceversa quando enablePause è `True` la funzione run del modulo viene eseguita ad ogni ciclo del Main Loop (figura 3.2); vanno inoltre modificate ed adattate le funzioni *isRunning()* e *pause()* nel codice del modulo, secondo la descrizione delle funzioni stesse.

4.3 Il loop dei comandi

L'engine è diviso in varie parti, come visto in figura 3.2. Ogni componente esegue il proprio compito sul main thread, tranne per i casi in cui si tratti

di lavori di ricezione. La ricezione dei dati dei droni nella componente GCS difatti avviene su un altro thread. Anche Mobile Handler, della stessa figura, agisce su un thread separato per ricevere le richieste del cellulare; una volta ricevuto un messaggio, il Mobile Handler lo incapsula in un comando e lo aggiunge alla lista *FIFO* adibita ai comandi. Anche il Main Loop accede alla lista; una volta per ogni ciclo di esecuzione infatti esegue il *pop()* della struttura e ne decodifica il contenuto, cedendo quando necessario il controllo al Module Handler.

Nel caso in cui la lista di comandi sia vuota il Main Loop ripete la propria esecuzione senza accedere ad altre componenti dell'engine. Tuttavia, se un modulo dalla run continua (descritta nella sezione precedente) è attualmente attivo, il Main Loop ne eseguirà il codice attraverso il Module Handler ad ogni ciclo. Questo comportamento dà comunque priorità a nuovi comandi; una volta incontrato il comando pause nella lista l'esecuzione continua viene stroncata.

Il funzionamento dei comandi di utilizzo del controller fisico o virtuale è molto simile: la ricezione avviene su un thread separato che inserisce delle strutture ControllerAction in una lista FIFO; la lista viene letta dal modulo in esecuzione continua attuale se supporta il controller, altrimenti viene svuotata al cambio del modulo corrente.

4.4 Comandi *super*

Per creare separazione tra quello che un modulo è in grado di fare e quello che invece è responsabilità dell'engine, il progetto richiede lo sviluppo di un sistema di comandi base e *super*. I comandi base descrivono quale funzione o componente di un modulo va eseguita e come. I comandi *super* invece, una volta ricevuti dal Module handler, vengono eseguiti direttamente come moduli dalla run singola. I comandi *super* rappresentano funzioni che un modulo non è predisposto per eseguire, alcune di queste sono il sync dei moduli tra l'engine ed il cellulare, il decollo, e l'atterraggio dei droni.

I comandi sono strutture dati rappresentate dal seguente codice:

```
{
    module: str,
    action: str,
    data: Any
}
```

Per un modulo normale il valore di `module` diventa lo *uuid4* corrispondente a quello della sua programmazione; l'`action` prende uno di 3 valori: *run*, *setup* o *pause*; il campo `data` contiene le risposte inserite negli input del cellulare da parte dell'utente, ed è rilevante solo con `action` di tipo *setup*.

Per un modulo *super* invece il campo `action` contiene sempre il valore di testo `"super"`; `module` contiene il nome codificato per quale codice va eseguito (i.e. `"decollo"`, `"atterraggio"`, etc); il valore di `data` è libero: al momento nessuno dei comandi *super* necessita di dati, ma se ciò dovesse cambiare in futuro con l'aggiunta di nuove funzionalità basterebbe inserire le strutture necessarie in questo terzo campo.

Capitolo 5

Validazione

In questo capitolo vedremo l'implementazione di un modulo dall'esecuzione ibrida all'interno dell'engine. Il modulo, in particolare, prende il titolo di "Circles Test" e, come da nome, rappresenterà un test di esecuzione per determinare se l'engine soddisfa o meno i requisiti imposti sul progetto.

La validazione consiste nel ruotare i droni in tondo seguendo vari punti equidistanti lungo una circonferenza di raggio 50 metri. Viene salvato il valore GPS di ogni UAV al massimo della frequenza di report. Viene inoltre registrato il tempo totale di percorrenza per compiere un giro lungo la circonferenza. La variabile in questo test è il numero di punti presi in considerazione: l'aspettativa è che più posizioni porteranno ad un aumento della precisione e del tempo di percorrenza.

Di seguito vengono presentati i passi di codifica delle principali funzioni responsabili del corretto funzionamento del modulo.

5.1 ModuleData e front end

L'implementazione comincia con la definizione del modulo nell'engine. In particolare la variabile descrittiva del modulo è un'istanza della classe ModuleData e viene compilata come segue:

```
{
```

```
    uuid: "6c4b76d6-f921-4c05-9dcb-c58fe659eccf",
    title: "Circles Test",
    description: ...,
    enablePause: True,
    enableController: True,
    enableSetup: True,
    inputs: [
        {
            what: "Circle Radius",
            type: "text"
        },
        {
            what: "Position Density",
            type: "text"
        }
    ]
}
```

Quindi il modulo chiede all'utente due input: raggio del cerchio e densità dei punti di posizione lungo la circonferenza; nella descrizione vengono dati ulteriori dettagli sui valori da inserire ed istruzioni per utilizzare correttamente il modulo.

A questo punto dello sviluppo del modulo è già possibile avviare l'engine ed eseguire il sync con l'app. Il front end appare automaticamente nella lista dei moduli come in figura 5.1. Nella stessa figura è presente la schermata del modulo una volta aperto, che è tuttavia mancante di funzionalità.

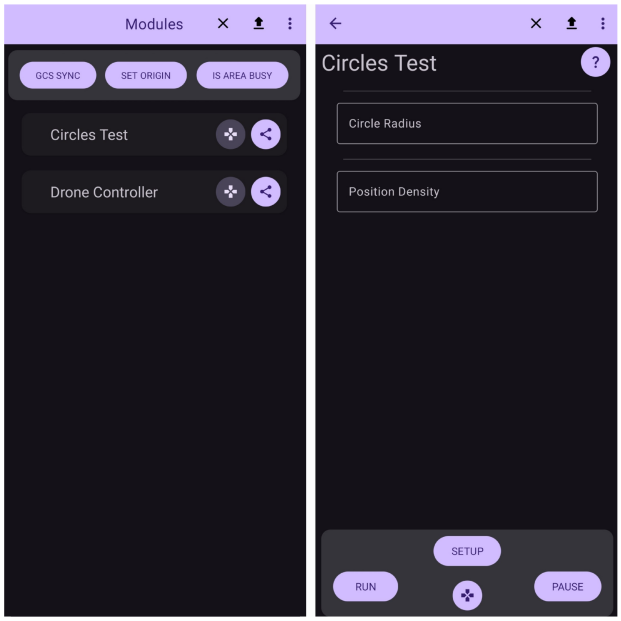

Figura 5.1: Schermata della lista dei moduli con Circles Test (sinistra). Schermata di Circles Test una volta aperto (destra)

5.2 Setup

All'interno del setup il modulo descrive quali condizioni o test devono essere eseguiti sui valori di input dell'utente. Il setup serve inoltre per salvare dati importanti per il funzionamento del resto del modulo; di seguito vediamo le variabili ricevute in input, calcolate e quindi salvate dalla funzione di setup.

- `r` è il raggio della circonferenza come da input dell'utente.
- `density` rappresenta il valore di densità delle posizioni lungo la circonferenza ed è sempre un input dell'utente.
- `n` è il numero dei droni, ed è un valore già in possesso del modulo al momento del setup.
- `count` corrisponde a $n \times density$.

- `pos` è un array contenente count coppie di coordinate x,y lungo la circonferenza del cerchio.
- `angle` è l'angolo che determina la distanza tra le coppie di pos lungo la circonferenza. È definito dalla formula $2\pi \div count$.
- `uavPos` è un array di dimensione n in cui ogni elemento rappresenta la posizione di un drone all'interno di pos. I droni hanno ciascuno un valore droneID che va da 0 a n-1; gli indici delle caselle di uavPos indicano l'id di ciascun drone.
- `runInCircle` è una variabile inizializzata a False che servirà per avviare il test.
- `startingPos` è un'intero utile come runInCircle per lo scopo del test.

Le due variabili r e density inserite devono soddisfare le seguenti condizioni:

$$r \geq n \div 2 \tag{5.1}$$

$$density \geq 1 \tag{5.2}$$

In questo modo i droni avranno sempre 3 metri di distanza l'uno dall'altro, e l'array pos non potrà avere lunghezza zero; se gli input non dovessero risultare corretti, verrebbe detto all'utente cosa va cambiato per renderli soddisfacenti.

Per il calcolo delle coppie x,y in pos viene utilizzata la libreria math. Di seguito il codice per la generazione delle x:

```
for i in range(count):
    pos[i].x = cos(angle * i) * r
```

Il calcolo delle y è possibile in due modi equivalenti: eseguire lo stesso codice inserendo il valore del seno piuttosto che quello del coseno, oppure calcolare per ciascuna delle x:

$$y = \sqrt{x^2 + r^2} \tag{5.3}$$

La scelta è ricaduta sull'utilizzo dell'equazione 5.3.

5.3 Run

Questo modulo è ad esecuzione continua (come descritto nel capitolo 4), quindi il codice della funzione *run()* deve essere scritto con la stessa logica di un codice in loop. Il modulo richiede l'utilizzo del controller, perciò la funzione riceve anche la lista dei tasti premuti dall'utente. Del controller utilizzeremo soltanto tre tasti, ma sono sufficienti per rendere il tipo di utilizzo dello swarm da indiretto ad ibrido. Ad ogni ciclo di esecuzione il modulo chiama la *pop()* della lista e decide cosa fare del tasto premuto.

Vediamo ora una panoramica generale della funzione:

```
btn = controller.pop()

if btn != None:
    match btn:
        case ctrl.A:
            distribute(swarm)
        case ctrl.B:
            shift(swarm)
        case ctrl.Y:
            runInCircle = not runInCircle
            saveStartingPos()

if runInCircle:
    test(swarm)
```

5.3.1 Tasto A

Quando il tasto rilevato è il tasto A del controller viene chiamata *distribute()* sempre definita nella classe del modulo. Lo scopo è quello di distribuire in modo equidistante i droni sulle varie posizioni della circonferenza calcolate nel setup. Per fare ciò vengono manipolati i valori di uavPos come segue:

```
for i in range(n):
    uavPos[i] = i*density
```

Infine viene chiamata, una volta per drone, la funzione Goto(x, y, z) che nel nostro caso riceve i parametri omonimi e $z = -20$ (-20 rappresenta una distanza di 20 metri da terra in quanto il sistema NED ha valori negativi di altezza). Il codice corrispondente è:

```
for i, drone in enumerate(swarm.drones):
    drone.Goto(pos[uavPos[i]].x, pos[uavPos[i]].y, -20)
```

5.3.2 Tasto B

Il tasto B è responsabile per la chiamata di un'altra funzione interna alla classe creata in questo modulo, *shift()*. Come da nome, lo scopo è quello di fare uno *shift* delle posizioni dei droni all'interno di pos. Il seguente codice si occupa della riassegnazione delle posizioni:

```
for dronePosition in uavPos:
    dronePosition = (dronePosition + 1) % count
```

A questo punto la chiamata corrispondente a Goto viene eseguita: il codice è del tutto identico a quello mostrato in precedenza.

Il tasto B è stato inizialmente una prova di implementazione della rotazione dei droni per le varie posizioni. Successivamente è servito invece per orientare approssimativamente ogni drone nella direzione del prossimo punto da raggiungere all'interno del cerchio, la posizione finale prima dell'inizio della rotazione con test.

5.3.3 Tasto Y

Questo tasto si comporta da *switch*, cambiando il valore della variabile runInCircle da True a False e viceversa . Salva inoltre la posizione del drone con droneID zero con una chiamata a *saveStartingPos()*:

```
startingPos = uavPos[0]
```

5.3.4 Test

La funzione di test è divisibile in 2 sezioni principali: l'effettivo giro in tondo ed il salvataggio dei dati alla sua terminazione.

Rotazione

Per eseguire la rotazione viene utilizzata la funzione *shift()* descritta in precedenza ed un controllo dello swarm dato da *isOnTarget()*. *isOnTarget()* ritorna True solo quando l'intero swarm ha completato uno shift. Durante ogni ciclo di rotazione tutti i dati di posizione ed i tempi di registrazione di tali dati sono salvati in delle variabili di supporto.

Termine della rotazione

La rotazione termina quando due condizioni sono vere al contempo:

- Il drone zero riceve il comando di tornare alla startingPos.
- L'intero swarm raggiunge il target dato dallo shift finale.

Una volta raggiunto questo punto runInCircle è posta a False. Tutti i dati salvati vengono racchiusi in un dizionario Python e salvati su un file csv attraverso la libreria Pandas. Le variabili contenenti la posizione GPS dei droni nel tempo vengono azzerate per la prossima esecuzione.

Vediamo ora come avviene l'esecuzione del modulo.

5.4 Esecuzione

Una volta salvato il modulo viene avviata la simulazione con SITL e Gazebo. L'engine è stato avviato con i parametri 5 (numero dei droni n) e l'indirizzo locale del cellulare su cui è installata la companion app. Infine è stata aperta l'app ed eseguito il sync dei moduli. La vista dal cellulare è sempre quella in figura 5.1.

Nei campi Circle Radius e Position Density sono stati inseriti i valori 50 e 1 rispettivamente. Sono stati fatti decollare i droni con il tasto lungo la toolbar in alto, premuto il tasto *SETUP* ed infine il tasto *RUN*. Una volta determinato che i dati inseriti rispettavano le condizioni imposte dal modulo è stato aperto il controller virtuale attraverso il tasto piccolo e tondo in basso nella schermata del modulo. Così facendo, si è aperta la schermata della figura 3.6. Una volta qui è stato sufficiente premere il bottone A, attendere che i droni prendessero il proprio posto lungo la circonferenza; successivamente è stato premuto il tasto B per due volte, per orientare i droni correttamente lungo il cerchio; infine il tasto Y per iniziare il test.

5.5 Calcolo dei dati

Il test è stato eseguito per valori di Position Density da 1 a 8 e con raggio costante di 50 metri.

Per tutti i calcoli sono state prese in considerazione le posizioni GPS nel tempo date dal drone zero.

Calcolo dell'errore

La posizione del drone è già convertita secondo il sistema di riferimento NED al momento del salvataggio dei dati. Della posizione in NED vengono presi in considerazione gli assi X ed Y e quindi i valori corrispondenti.

L'errore sulla distanza dalla circonferenza perfetta per ogni valore di posizione dato dal drone è stato calcolato come segue:

$$Err = |r - dist| \tag{5.4}$$

Dove r è il raggio di 50 metri, mentre dist è dato dalla formula della distanza tra due punti: il centro del cerchio e la posizione del drone. Dato che il centro del cerchio corrisponde all'origine del piano dist può essere calcolato come:

$$dist = \sqrt{x^2 + y^2} \tag{5.5}$$

L'errore medio su tutte le posizioni date dal drone è quindi la media dei valori di Err per ogni coppia x, y.

Calcolo del tempo di percorrenza

Il tempo di percorrenza della circonferenza è pari a:

$$\Delta T = t1 - t0 \tag{5.6}$$

Con t1 tempo al momento del ritorno alla posizione di partenza e t0 tempo al momento della partenza.

5.6 Risultati

Una volta eseguiti i test per le varie quantità di posizioni abbiamo ottenuto i risultati rappresentati in figura 5.2: all'aumentare delle posizioni da raggiungere, l'errore di locazione istantaneo medio diminuisce. Sempre all'aumentare del numero di posizioni, il tempo di percorrenza aumenta. I dati confermano l'intuizione su cui si è basato l'esperimento.

Va considerato che l'errore si avvicina molto agli zero metri, già dalle 25 posizioni a salire. La relazione tra l'errore ed il numero delle posizioni è logaritmica.

Per quanto riguarda invece il tempo di percorrenza, questo aumenta all'aumentare delle posizioni con una relazione lineare diretta. Per i valori di

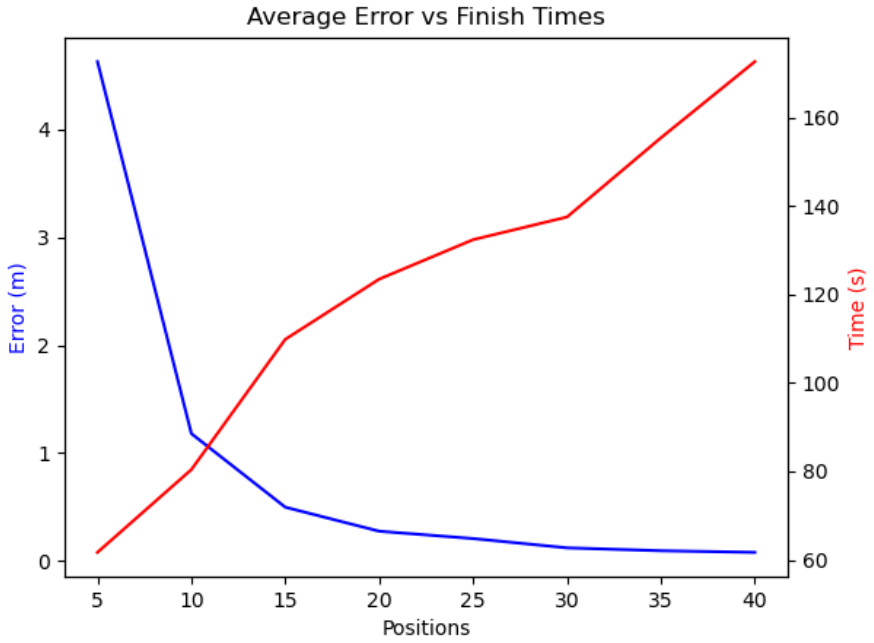

Figura 5.2: Tempo di percorrenza (in rosso) ed errore medio sulla posizione (in blu) di un drone che percorre vari punti di una circonferenza di raggio 50 metri.

posizione dal 10 al 30 compresi, il tempo di arrivo fluttua di molto. Questo comportamento potrebbe essere il risultato di vari delay di comunicazione tra GCS e droni e della frequenza alla quale i droni inviano i propri dati di posizione. La funzione `isOnTarget()` è responsabile dell'attesa tra uno shift di posizione ed il prossimo, e si basa sui report di posizione dei singoli droni, diventando quindi soggetta a possibili rallentamenti casuali.

Capitolo 6

Conclusioni

Questa tesi ha avuto come obiettivo lo sviluppo di un engine per semplificare la gestione di uno swarm di droni. Per fare ciò si è considerato l'utilizzo della suite di SDK dell'azienda francese Parrot; tuttavia questo avrebbe obbligato gli utenti dell'engine ad acquistarne i droni per poter utilizzare i moduli creati.

Per quanto riguarda il linguaggio di programmazione, la scelta era divisa tra Python e C++. Python non è riconosciuto come un linguaggio particolarmente performante, tuttavia, comparato a C++, è più semplice e di alto livello: non richiede all'utente di comprendere ed utilizzare i puntatori e si prende invece cura del garbage collection. Con queste caratteristiche diventa sicuramente una scelta preferita per abbassare la barriera tecnica discussa in precedenza. Inoltre il caso d'uso dell'engine comprende il testing di algoritmi che prevede la raccolta e l'analisi di dati. Python è il linguaggio di Data Analysis per eccellenza; proprio nel capitolo 5 di questa tesi abbiamo visto come la libreria open source Pandas è stata utile per testare un semplice algoritmo di validazione dell'engine.

I risultati della validazione hanno confermato che l'engine è viabile per lo sviluppo ed il testing di algoritmi per la gestione dei droni; Heli permette inoltre la creazione di moduli dal controllo di swarm diretto, indiretto ed ibrido. Infine, l'engine utilizza il sistema di riferimento NED di cui abbiamo

visto l'implementazione nel capitolo 4 per rendere il più semplice possibile il posizionamento ed il movimento dei droni nei moduli.

Di seguito vi sono alcune possibili espansioni per ampliare le capacità dell'engine e diffonderne l'utilizzo.

6.1 Sharing e repository

In figura 3.4, ogni modulo della lista presenta un tasto laterale destro adibito ad una funzione di condivisione. La condivisione dei moduli al momento funziona, ma si tratta di una condivisione unicamente del front end. Se un modulo non dovesse essere implementato all'interno dell'engine connesso, il front end non sarebbe utilizzabile. Per questo motivo i moduli che invece sono implementati presentano il cerchietto viola sulla sinistra, nella stessa figura. In futuro il modulo così come programmato nell'engine si potrebbe inviare al cellulare, in modo da poterlo effettivamente condividere per intero ad un altro utente in possesso dell'applicazione. I moduli, dopotutto, sono documenti di testo con estensione .py, quindi, tale funzionalità è già presente con le tecnologie attuali.

Un altro ottimo modo per condividere i moduli potrebbe essere quello di avere un sito, una repository, dove si possano caricare e dalla quale si possano scaricare.

6.2 Utilizzo di droni reali e messaggi

Il progetto è basato sulla simulazione dei droni attraverso le tecnologie di SITL e Gazebo. Un ottimo punto di espansione del progetto potrebbe essere l'utilizzo di droni reali. Il controllo di droni virtuali non verrebbe tuttavia completamente eliminato, ma diventerebbe invece possibile eventualmente su richiesta tramite un parametro all'avvio dell'engine.

I messaggi scambiati tra i droni e l'engine sono al momento limitati: un'ulteriore espansione di compatibilità potrebbe essere il supporto di tutti

i messaggi MAVLink, compresi quelli di controllo della telecamera e altri sensori, possibilmente presenti su ciascun drone.

6.3 MAVLink e compatibilità

L'engine è in grado di comandare droni controllati da Ardupilot. Dato che la comunicazione drone-GCS avviene tramite il protocollo MAVLink, in futuro anche droni controllati da altri software autopilota come PX4 potrebbero essere utilizzati. Se l'ampliamento di compatibilità non fosse immediato, sarebbe per lo meno semplice adattare l'engine, dato che la tecnologia utilizzata è la stessa.

L'azienda Parrot è stata presa in considerazione più volte per alcune componenti di questo progetto, ma nella sua versione finale il progetto non è compatibile con i droni venduti da essa. Della suite di software messi a disposizione dalla Parrot non abbiamo parlato di AirSDK: si tratta di un software che permette di scrivere codice che viene direttamente eseguito dal firmware del drone. È quindi possibile tentare di creare, proprio attraverso AirSDK, un layer di compatibilità per il controllo di droni costruiti dalla Parrot.

6.4 Controllo collisioni ed altre funzionalità

Lo sviluppo dell'engine potrebbe continuare con l'aggiunta di funzionalità native in modo che gli utenti possano utilizzare algoritmi di collision avoidance e simili direttamente nei loro moduli, senza doverli implementare da zero. Questi algoritmi possono essere prima costruiti, poi testati come moduli a loro volta. Dopo lo studio ed il testing della loro funzionalità, potrebbero essere integrati in un aggiornamento, così che tutti possano farne uso.

Bibliografia

[1] Ardupilot, 2024. https://ardupilot.org/.

[2] Ardupilot sitl, 2024. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[3] Dji ground station pro, 2024. https://www.dji.com/it/ground-station-pro/.

[4] Dji gs pro faqs, 2024. https://www.dji.com/it/ground-station-pro/faq.

[5] Dji selfie drone, 2024. https://store.dji.com/it/content/selfie-drone.

[6] Drone market value report, 2024. https://www.skyquestt.com/report/drone-market.

[7] Mavlink, 2024. https://mavlink.io/en/.

[8] Mavlink messages, 2024. https://mavlink.io/en/messages/common.html.

[9] Mavproxy, 2024. https://ardupilot.org/mavproxy/index.html#home.

[10] Parrot ground sdk, 2024. https://developer.parrot.com/docs/groundsdk-android/overview.html.

[11] Parrot olympe sdk, 2024. https://developer.parrot.com/docs/olympe/overview.html.

[12] Parrot open software development kit, 2024. https://www.parrot.com/en/open-source-drone-software.

[13] Parrot sphinx sim, 2024. https://developer.parrot.com/docs/sphinx/index.html.

[14] Px4, 2024. https://px4.io/.

[15] Qgroundcontrol, 2024. https://qgroundcontrol.com/.

[16] Unreal engine 5.5 documentation, 2024. https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-5-documentation.

[17] Sukhrob Atoev, Ki-Ryong Kwon, Suk-Hwan Lee, and Kwang-Seok Moon. Data analysis of the mavlink communication protocol. In *2017 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–3, 2017.

[18] Gregory M. Dering, Steven Micklethwaite, Samuel T. Thiele, Stefan A. Vollgger, and Alexander R. Cruden. Review of drones, photogrammetry and emerging sensor technology for the study of dykes: Best practises and future potential. *Journal of Volcanology and Geothermal Research*, 373:148–166, 2019.

[19] Milan Erdelj and Enrico Natalizio. Drones, smartphones and sensors to face natural disasters. In *Proceedings of the 4th ACM Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, DroNet'18, page 75â86, New York, NY, USA, 2018. Association for Computing Machinery.

[20] JesÃºs JimÃ©nez LÃ³*pezandMargaritaMulero* – *PÃ!'zmÃ!'ny*. *Dronesforconservationinprotectedareas* : *Presentandfuture.Drones*, 3(1), 2019.

[21] Maja Kucharczyk and Chris H. Hugenholtz. Remote sensing of natural hazard-related disasters with small drones: Global trends, biases, and research opportunities. *Remote Sensing of Environment*, 264:112577, 2021.

[22] Woonghee Lee, Joon Yeop Lee, Jiyeon Lee, Kangho Kim, Seungho Yoo, Seongjoon Park, and Hwangnam Kim. Ground control system based routing for reliable and efficient multi-drone control system. *Applied Sciences*, 8(11), 2018.

[23] Young Seung Lee, Dong Gook Lee, Young-Geol Yu, and Hyun-Jik Lee. Application of drone photogrammetry for current state analysis of damage in forest damage areas. *Journal of Korean Society for Geospatial Information System*, 24(3):49–58, 09 2016.

[24] Fabrice Saffre, Hanno Hildmann, and Hannu Karvonen. The design challenges of drone swarm control. In Don Harris and Wen-Chin Li, editors, *Engineering Psychology and Cognitive Ergonomics*, pages 408–426, Cham, 2021. Springer International Publishing.

Ringraziamenti

Ringrazio i miei genitori per il supporto e la libertà che mi hanno lasciato nello scegliere la mia strada. Ringrazio specialmente la mia compagna per aver sempre creduto in me ed avermi spronato ad intraprendere questo percorso di studi.