

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



**Sviluppo di un Protocollo di Rete Resiliente
per Sciami di Droni su Piattaforma Crazyflie**

Tesi di Laurea

Relatore

Prof. Corò Federico

Laureando

Valerio Solito

Matricola 2113178

ANNO ACCADEMICO 2025-2026

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, in cui è stato sviluppato e implementato un protocollo di rete Self-Healing per sciami di droni autonomi, ideato per garantire la continuità operativa in caso di perdita di connessione con la *base station*. A tal proposito è stato progettato un algoritmo decentralizzato che, tramite un modello a forze virtuali (molla-smorzatore), riposiziona fisicamente i droni nello spazio per ripristinare il segnale tramite un multi-hop routing.

Tale algoritmo è stato implementato in C su architettura FreeRTOS utilizzando la piattaforma embedded Crazyflie. I test sperimentali confermano la capacità del sistema nel rilevare i guasti di rete e attuare manovre correttive in tempo reale, così da riconfigurare dinamicamente la topologia della rete. Per isolare la logica di controllo dalle limitazioni della stima di distanza in ambienti indoor, l'algoritmo è stato inoltre validato in un ambiente di simulazione dedicato, che ne conferma la correttezza a prescindere dal rumore del segnale.

Indice

Acronimi e abbreviazioni	ix
Glossario	x
1 Introduzione	1
1.1 Il contesto	1
1.2 L'idea	1
1.3 Codice sorgente	2
1.4 Organizzazione del testo	2
2 Descrizione dello stage	4
2.1 Introduzione al progetto	4
2.2 Metodo di lavoro	5
2.3 Analisi preventiva dei rischi	6
2.4 Requisiti e obiettivi	7
2.5 Pianificazione	8
3 Analisi dei requisiti	10
3.1 Casi d'uso	10
3.2 Tracciamento dei requisiti	12
3.3 Tabelle dei requisiti	12
4 Progettazione e codifica	15
4.1 Tecnologie e strumenti	15
4.2 Ciclo di vita del software	16
4.3 Progettazione	17

4.4	Ambiente di simulazione	19
4.5	Codifica	20
5	Verifica e validazione	22
5.1	Verifica degli obiettivi obbligatori	22
5.1.1	Invio <i>broadcast</i> e aggiornamento della <i>Neighbor Table</i> (RFN-1)	22
5.1.2	Propagazione dell'hop count (RFN-2)	24
5.1.3	Calcolo delle forze e output telemetrico (RFN-3)	25
5.2	Verifica degli obiettivi desiderabili	27
5.2.1	Applicazione vettoriale delle forze ai controllori di volo (RFD-1)	27
5.2.2	Dimostrazione pratica in volo (RFD-2, parzialmente soddisfatto)	28
5.2.3	Confronto tra variante <i>weighted</i> e variante <i>centroid</i> (RFD-3, non soddisfatto)	29
5.3	Limiti riscontrati	29
6	Conclusioni	31
6.1	Consuntivo finale	31
6.2	Raggiungimento degli obiettivi	31
6.3	Conoscenze acquisite	32
6.4	Valutazione conclusiva	32
	Sitografia	ii

Elenco delle figure

4.1	Forza netta $F(d)$ agente su un vicino hop-adiacente in funzione della distanza d . A destra di D_{SAFE} (0.3m) agisce solo l'attrazione, che si azzera esattamente a D_{TARGET} (0.8m). Al di sotto di D_{SAFE} , la componente repulsiva si somma a quella attrattiva, generando una forza negativa (repulsione) che cresce esponenzialmente per evitare collisioni.	18
5.1	Avvio del drone ID 1 (primo test).	23
5.2	Log di ricezione <i>broadcast</i> e aggiornamento <i>Neighbor Table</i> (primo test).	23
5.3	Distanza stimata tramite <i>Received Signal Strength Indication (RSSI)_G</i> verso i vicini nel tempo, a fronte di una distanza fisica reale nota e costante di circa 1m tra i droni (test statico, senza <i>mission mode</i>).	24
5.4	Log dell' <i>hop count</i> corretto dopo l'introduzione del filtro topologico (secondo test).	25
5.5	Avvio del drone (terzo test).	26
5.6	Log della telemetria forze e del comportamento dell' <i>hop count</i> in seguito allo spegnimento reale della <i>base station</i> (terzo test). . .	26
5.7	Andamento temporale delle distanze in simulazione: a seguito della perdita di connessione (tempo 0), i droni si riposizionano attivamente fino a convergere e arrestarsi in corrispondenza della distanza di equilibrio D_{TARGET}	29

Elenco delle tabelle

3.1	Tabella del tracciamento dei requisiti funzionali.	13
3.2	Tabella del tracciamento dei requisiti qualitativi.	14
3.3	Tabella del tracciamento dei requisiti di vincolo.	14

Elenco dei codici sorgenti

4.1	Registrazione delle callback e delle API di comunicazione e parametri	16
4.2	Parametri tarabili del modello a forze virtuali	19
4.3	Selezione del vicino da sostituire (politica <i>Least Recently Updated</i> (LRU_G)	20
4.4	Calcolo delle forze attrattive e repulsive	20
4.5	Assegnazione deterministica dell'angolo verso un vicino	20
4.6	Calcolo della distanza geometrica nel simulatore in sostituzione all' $RSSI_G$	21

Acronimi e abbreviazioni

AoA Angle of Arrival. [i](#), [7](#), [14](#)

LRU Least Recently Updated. [i](#), [viii](#), [11](#), [18](#), [20](#)

P2P Peer-to-Peer. [i](#), [10–12](#), [14](#), [15](#), [17](#), [18](#), [24](#), [31](#)

ROS Robot Operating System. [i](#), [2](#), [5](#), [19](#), [21](#), [31–33](#)

RSSI Received Signal Strength Indication. [i](#), [vi](#), [viii](#), [2](#), [5–9](#), [11](#), [13](#), [14](#), [19](#),
[21–24](#), [28](#), [29](#), [31–33](#)

UWB Ultra-Wideband. [i](#), [2](#), [7](#), [14](#), [31](#)

Glossario

AoA Tecnica utilizzata in sistemi di localizzazione wireless per stimare la direzione di provenienza di un segnale radio, tipicamente tramite array di antenne, permettendo di ricavare informazioni angolari oltre alla sola distanza. [i](#)

LRU Politica di sostituzione utilizzata quando una struttura dati con capacità limitata è piena: l'elemento sostituito è quello aggiornato meno di recente tra quelli presenti. [i](#)

P2P Modello di comunicazione di rete in cui i nodi comunicano direttamente tra loro senza un server centrale intermediario, scambiandosi messaggi alla pari. [i](#)

ROS Framework open source per lo sviluppo di software robotico, che fornisce strumenti, librerie e convenzioni per la comunicazione tra processi (nodi) tramite un modello publish/subscribe. [i](#)

RSSI Indicatore della potenza del segnale radio ricevuto da un dispositivo wireless, espresso tipicamente in dBm. Viene spesso utilizzato come metrica indiretta per stimare la distanza tra due nodi di una rete, sfruttando il fatto che il segnale si attenua con la distanza. [i](#)

UWB Tecnologia di trasmissione radio a banda ultra-larga che utilizza impulsi di brevissima durata, permettendo di misurare con elevata precisione il tempo di volo (*Time-of-Flight*) di un segnale tra due dispositivi. Grazie a questa proprietà è comunemente impiegata per la localizzazione e il

calcolo di distanze in ambito indoor, con un'accuratezza superiore rispetto a tecniche basate sulla sola potenza del segnale. [i](#), [3](#)

Capitolo 1

Introduzione

1.1 Il contesto

Lo stage descritto in questo elaborato è stato svolto internamente presso l'Università degli Studi di Padova, anziché presso un'azienda esterna. Il progetto è stato proposto dal Prof. Federico Corò, tutor aziendale dello stage. L'attività si è svolta inizialmente da remoto, per lo studio preliminare dell'architettura hardware e software e per una prima stesura del codice, e in parte presso Torre Archimede, sede del Dipartimento di Matematica "Tullio Levi-Civita", dove è stato possibile interfacciarsi direttamente con i droni Crazyflie.

1.2 L'idea

L'algoritmo alla base di questo lavoro era stato sviluppato in precedenza dal Prof. Corò, e validato esclusivamente tramite simulazioni software, senza un'implementazione su hardware reale. Il modello è di tipo a forze virtuali: ciascun drone viene trattato come una particella soggetta a forze fittizie generate dai vicini, per analogia con un sistema fisico *molla-smorzatore*. Una componente attrattiva, simile a quella di una molla, richiama il drone verso una distanza di equilibrio dai nodi adiacenti nella catena; una componente repulsiva agisce invece da freno anti-collisione quando due droni si avvicinano troppo. La somma vettoriale di queste forze produce un comando di velocità che guida autono-

mamente il drone verso una posizione in grado di ripristinare la connettività, senza bisogno di una pianificazione centralizzata del percorso. Il modello è pensato per ripristinare la connettività di uno sciame di droni in caso di perdita del segnale con la *base station*, riorganizzando dinamicamente la topologia di rete in una catena *multi-hop*: una sequenza di collegamenti radio intermedi che permette a due nodi troppo distanti per comunicare direttamente di scambiarsi dati passando attraverso nodi intermedi che ritrasmettono il messaggio.

Obiettivo di questo stage è stato portare tale algoritmo dalla simulazione a un'implementazione reale su piattaforma Crazyflie, sostituendo il livello di comunicazione e localizzazione originariamente previsto (basato su tecnologia *Ultra-Wideband (UWB)_G*, con stima delle distanze tramite tempo di volo) con un'infrastruttura Wi-Fi, stimando la distanza tra i droni tramite il segnale *RSSI_G* e valutandone la fattibilità pratica sull'hardware.

La validazione del sistema è stata condotta sia tramite test fisici su hardware reale, sia tramite un ambiente di simulazione basato su *Robot Operating System (ROS)_G 2* e Gazebo, sviluppato nel corso dello stage per isolare la logica di controllo dalle limitazioni della stima *RSSI_G* riscontrate in ambiente indoor.

1.3 Codice sorgente

Tutto il codice sorgente sviluppato e utilizzato per la validazione di questo elaborato è open-source ed è consultabile pubblicamente sulle seguenti repository GitHub:

- **Firmware Crazyflie (C):** <https://github.com/valeriosolito/crazyflie-firmware-stage>
- **Simulatore *ROS_G 2* (Python):** https://github.com/valeriosolito/self_healing_simulation

1.4 Organizzazione del testo

Il secondo capitolo descrive il contesto dello stage, i rischi individuati in fase di analisi, gli obiettivi del piano di lavoro e la pianificazione delle attività svolte.

Il terzo capitolo formalizza gli obiettivi dello stage in casi d'uso e requisiti funzionali, qualitativi e di vincolo.

Il quarto capitolo descrive le tecnologie e gli strumenti utilizzati, il ciclo di sviluppo seguito, l'architettura del sistema realizzato e alcuni estratti di codice significativi.

Il quinto capitolo verifica il raggiungimento degli obiettivi tramite le evidenze sperimentali raccolte durante i test svolti.

Nel sesto capitolo vengono riassunti i risultati raggiunti, le conoscenze acquisite durante lo stage e alcuni possibili sviluppi futuri del progetto.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

- gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
- per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente nomenclatura: *Ultra-Wideband***G**;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.
- i riferimenti a funzioni, variabili, macro, file o comandi sono evidenziati con il carattere a spaziatura fissa (**teletype**).

Capitolo 2

Descrizione dello stage

2.1 Introduzione al progetto

Il test del sistema è stato condotto utilizzando tre droni Crazyflie 2.x¹, ciascuno identificato da un indirizzo radio univoco e da un ruolo specifico all'interno della rete: un drone con ID 0, che funge da *base station* e rimane fermo a terra; un drone con ID 1, che funge da nodo *relay*; un drone con ID 2, che simula la perdita di connettività con la base.

I droni sono stati posizionati secondo una topologia lineare, con una separazione fisica di circa un metro l'uno dall'altro, tutti orientati nella stessa direzione. Solo i droni con ruolo di relay sono stati equipaggiati con un Flow Deck, per garantire stabilità sull'asse Z durante il volo, mentre il drone di base, dovendo restare a terra, ne era sprovvisto. Per migliorare la stabilità posizionale e ridurre la deriva causata da superfici del pavimento poco texturizzate o riflettenti (come segnalato anche dalla community ufficiale Bitcraze²), sono stati posizionati fogli di giornale lungo il percorso di volo, così da fornire pattern visivi ad alto contrasto per i sensori di flusso ottico.

¹Component explanation of the Crazyflie 2.x. URL: <https://www.bitcraze.io/documentation/system/platform/cf2-components/>.

²Optical flow deck and floor texture. URL: <https://forum.bitcraze.io/viewtopic.php?t=3425>.

2.2 Metodo di lavoro

Lo sviluppo si è articolato in due fasi principali. Una prima fase, condotta da remoto, dedicata allo studio preliminare dell'architettura hardware e software dei droni Crazyflie e a una prima stesura del codice. Una seconda fase, svolta presso Torre Archimede, in cui il codice veniva testato direttamente sui droni fisici e modificato sulla base dei risultati osservati in volo.

Disponendo inizialmente di un solo Flow Deck, il primo approccio di test sperimentato è stato quello di far muovere i droni a terra, mantenendo l'asse Z disattivato, anziché farli volare. Tale approccio è stato abbandonato sia per l'attrito tra i droni e la superficie del pavimento, che impediva un movimento fluido e prevedibile, sia per la medesima inaffidabilità della stima $RSSI_G$ in ambiente indoor discussa più avanti in questo capitolo. Si è quindi proceduto a equipaggiare con un Flow Deck aggiuntivo i droni con ruolo di relay, passando ai test in volo che costituiscono la base della sperimentazione descritta in questo elaborato.

Al termine di ogni giornata di lavoro in Torre Archimede, i droni venivano restituiti al Prof. Corò, con un breve confronto sulle attività svolte e sui prossimi passi. Con cadenza settimanale, inoltre, veniva effettuato un aggiornamento più strutturato sullo stato di avanzamento del progetto. Inoltre in ogni momento era possibile visionare lo stato del codice sulla repository di Github condivisa.

Nel corso dei test in volo, la difficoltà di isolare gli effetti del rumore $RSSI_G$ dal comportamento reale dell'algoritmo (Rischio 1) ha reso complesso validare in modo pulito la convergenza dello sciame verso la configurazione desiderata. Per questo motivo, su suggerimento del Prof. Corò, si è deciso di affiancare ai test fisici lo sviluppo di un ambiente di simulazione basato su ROS_G 2 e Gazebo, in cui la stima di distanza può essere sostituita da un valore esatto noto al simulatore, permettendo di isolare e validare la sola logica di controllo (Sezione 4.4).

2.3 Analisi preventiva dei rischi

Durante la fase di analisi iniziale sono stati individuati alcuni possibili rischi a cui si potrà andare incontro. Si è quindi proceduto a elaborare delle possibili soluzioni per far fronte a tali rischi.

1. Inaffidabilità della stima di distanza tramite $RSSI_G$ in ambiente indoor

Descrizione: Il fenomeno del *multipath fading*, tipico degli ambienti chiusi, causa forti variazioni del segnale ricevuto (fino a ± 10 dBm), rendendo la stima della distanza tramite $RSSI_G$ inaffidabile per il controllo motorio. In spazi di test di dimensioni ridotte, tale fenomeno può inoltre causare la ricezione diretta di pacchetti da nodi non adiacenti nella topologia, con conseguente calcolo errato dell'hop count.

Soluzione: Per la dimostrazione pratica dell'attuazione dei vettori di forza, si è scelto di utilizzare una distanza fissa nota a priori, mantenendo comunque la formula originale nel codice come riferimento; per il problema dell'hop count, è stato introdotto un filtro topologico che limita l'aggiornamento della *Neighbor Table* ai soli vicini diretti nella catena. Per una validazione più rigorosa e indipendente dal rumore $RSSI_G$, è stata inoltre sviluppata una simulazione dedicata (Sezione 4.4).

2. Difficoltà di connessione al drone durante i test in volo

Descrizione: La frequenza elevata dei messaggi *broadcast* scambiati tra i droni rende talvolta difficoltosa la connessione tramite `cfclient` per l'invio di comandi (come il parametro di `sig_loss` per simulare la perdita di connessione con la *base station*) o la modifica di parametri (ad esempio l'indirizzo radio o il canale).

Soluzione: Coordinamento attento nei tempi di connessione e disconnessione, ed esecuzione dei test in condizioni controllate. È stato inoltre necessario aumentare l'intervallo del loop di *broadcast* da 50 ms, previsto dall'algoritmo originale, a 250 ms, per garantire la stabilità della connessione `cfclient` senza comprometterne il funzionamento.

3. Sicurezza fisica durante i test in volo

Descrizione: Il posizionamento ravvicinato dei droni e i movimenti autonomi generati dal controllore in *mission mode* comportano un rischio di collisione.

Soluzione: Introduzione di una soglia di sicurezza (D_SAFE) nel calcolo delle forze repulsive, ed esecuzione dei test in ambiente controllato a bassa quota.

4. Impossibilità di determinare l'angolo verso i vicini

Descrizione: A differenza di un sistema basato su UWB_G , che permetterebbe tecniche di $Angle\ of\ Arrival\ (AoA)_G$ per stimare la direzione di un vicino, l'uso del solo $RSSI_G$ fornisce esclusivamente un'informazione di distanza, senza alcuna informazione angolare.

Soluzione: È stata adottata una mappatura deterministica dell'angolo basata sull'ID del drone, valida per la topologia lineare utilizzata nei test, ma non generalizzabile a configurazioni diverse.

5. Inesperienza pregressa con sviluppo embedded e FreeRTOS

Descrizione: Lo sviluppo su piattaforma Crazyflie richiede familiarità con concetti di programmazione embedded e con il sistema operativo real-time FreeRTOS, non affrontati in precedenza.

Soluzione: Studio della documentazione ufficiale Crazyflie e apprendimento tramite approccio iterativo di sviluppo e test.

2.4 Requisiti e obiettivi

Il piano di lavoro dello stage ha definito i seguenti obiettivi.

Obiettivi obbligatori

1. Realizzazione del software per l'invio di messaggi in *broadcast* periodico, al fine di aggiornare la *Neighbor Table* tramite stime di distanza Wi-Fi;
2. Sviluppo del meccanismo per la propagazione dei messaggi necessari a calcolare la distanza in salti dalla base;

3. Implementazione dell'algoritmo per il calcolo a bordo delle forze attrattive verso i salti adiacenti e repulsive verso i vicini sotto la soglia di sicurezza, con produzione di output telemetrico.

Obiettivi desiderabili

1. Sviluppo dell'integrazione necessaria all'applicazione vettoriale delle forze calcolate ai controllori di volo, per attuare il movimento del drone;
2. Configurazione ed esecuzione di una dimostrazione pratica in volo comprendente un nodo base, un relay e un drone che perde la connessione per poi essere ri-aggiungato e recuperato;
3. Sviluppo ed esecuzione di test per confrontare la variante attrattiva *weighted* rispetto alla variante *centroid*.

Il grado di raggiungimento di ciascun obiettivo, con le relative evidenze sperimentali, viene discusso nel Capitolo 5. L'analisi formale dei requisiti, tradotti in requisiti funzionali, qualitativi e di vincolo, è invece riportata nel Capitolo 3.

2.5 Pianificazione

Le attività di stage sono state pianificate per un impegno complessivo stimato in 320 ore, suddivise nelle seguenti fasi:

- **Prima Fase – Studio, Setup e Familiarizzazione (50 ore):** Studio dell'architettura hardware/software dei Crazyflie 2.1, analisi del modello a molla-smorzatore virtuale, configurazione dell'ambiente di sviluppo e test preliminari;
- **Seconda Fase – Rete e Topologia Locale (80 ore):** Sviluppo dello scambio di pacchetti in rete ad-hoc, misurazione empirica del $RSSI_G$ e sviluppo del meccanismo di timeout;
- **Terza Fase – Logica Decentralizzata e Middleware (80 ore):** Implementazione della propagazione dell'hop count dalla base, traduzione

delle formule fisiche in codice eseguibile a bordo, simulazione a terra dei vettori calcolati;

- **Quarta Fase – Integrazione e Tuning in Volo (60 ore):** Traduzione dell’accelerazione vettoriale in setpoint di velocità, tuning dei parametri sul campo, esecuzione dello scenario self-healing in ambiente controllato;
- **Quinta Fase – Raccolta Dati (50 ore):** Estrazione della telemetria, validazione dell’efficacia del relay multi-hop.

Rispetto alla pianificazione originale, la fase di raccolta dati ha evidenziato limiti significativi legati all’inaffidabilità della stima $RSSI_G$ in ambiente indoor (Rischio 1), che hanno condizionato il raggiungimento completo di alcuni obiettivi desiderabili; tali scostamenti sono discussi nel dettaglio nel Capitolo 5.

Capitolo 3

Analisi dei requisiti

3.1 Casi d'uso

Trattandosi di un sistema embedded autonomo, gli attori coinvolti non sono utenti finali in senso tradizionale, ma il drone stesso, che agisce autonomamente sulla base dei dati ricevuti dalla rete, e l'operatore, che interagisce col sistema solo a scopo di configurazione e test tramite il tool *cfclient*. Per questo motivo i casi d'uso individuati sono pochi e concettualmente semplici.

UC0: Rilevamento perdita di connessione e attivazione *self-healing*

Attori Principali: Drone (nodo relay o nodo isolato).

Precondizioni: Il drone è in volo e ha stabilito comunicazione *Peer-to-Peer* (*P2P*)_G con gli altri nodi della rete.

Descrizione: Il drone rileva l'assenza di ACK dalla *base station* per un tempo superiore alla soglia di timeout, attiva la *mission mode* e calcola le forze attrattive e repulsive verso i vicini, convertendo il vettore risultante in un setpoint di velocità per riposizionarsi e ripristinare la connettività.

Postcondizioni: Il drone si sposta secondo il vettore calcolato; la *Neighbor Table* e l'*hop count* vengono aggiornati ad ogni ciclo di controllo.

UC1: Aggiornamento della *Neighbor Table*

Attori Principali: Drone.

Precondizioni: Il drone riceve un pacchetto $P2P_G$ da un vicino.

Descrizione: Il drone estrae ID, *hop count* e stato dal pacchetto ricevuto, stima la distanza tramite $RSSI_G$ e aggiorna la propria *Neighbor Table*.

Postcondizioni: La *Neighbor Table* riflette lo stato aggiornato dei vicini rilevati nell'ultimo intervallo di timeout.

Scenario Alternativo: Se la tabella è piena e il vicino non è già presente, viene sostituito il vicino aggiornato meno di recente (politica LRU_G).

UC2: Configurazione e test tramite `cfclient`

Attori Principali: Operatore.

Precondizioni: L'operatore ha connesso il Crazyradio al drone tramite `cfclient`.

Descrizione: L'operatore monitora i log di telemetria, modifica parametri quali indirizzo radio o canale (se necessario), e simula la perdita di segnale dalla *base station* iniettando il parametro `sig_loss`.

Postcondizioni: Il comportamento del drone viene osservato e validato tramite i log prodotti.

Scenario Alternativo: Se l'operatore tenta di stabilire o mantenere la connessione telemetrica via radio (Crazyradio PA) mentre i droni scambiano pacchetti $P2P_G$ ad alta frequenza, si verifica una forte saturazione del canale. Il `cfclient` può subire un grave degrado delle prestazioni, mostrando estrema latenza o avvisi di timeout; tuttavia, attendendo o forzando i tentativi di riconnessione senza chiudere il client, il collegamento riesce a stabilirsi, permettendo la lettura dei log pur con grandi ritardi che continueranno anche durante la lettura dei log. L'uso del cavo USB rimane relegato esclusivamente alle fasi di configurazione statica a terra a motori spenti.

UC3: Trasmissione periodica dello stato (*broadcast*)

Attori Principali: Drone.

Precondizioni: Il drone è acceso e il ciclo di controllo FreeRTOS è attivo.

Descrizione: Ogni 250 ms il drone impacchetta il proprio ID, il proprio *hop*

count calcolato e il proprio stato corrente (normale o emergenza) e lo trasmette in *broadcast* via radio $P2P_G$.

Postcondizioni: I droni vicini ricevono il pacchetto e possono aggiornare le loro *Neighbor Table*.

3.2 Tracciamento dei requisiti

Dall'analisi degli obiettivi dello stage e dei casi d'uso individuati è stata stilata la tabella che traccia i requisiti in rapporto agli use case.

Sono stati individuati diversi tipi di requisiti e si è quindi fatto utilizzo di un codice identificativo per distinguerli.

Il codice dei requisiti, dove ogni requisito è identificato con il carattere **R**, è così strutturato:

F: Funzionale.

Q: Qualitativo.

V: Di vincolo.

N: Obbligatorio (necessario).

D: Desiderabile.

Z: Opzionale.

Nelle tabelle 3.1, 3.2 e 3.3 sono riassunti i requisiti e il loro tracciamento con gli use case delineati in fase di analisi.

3.3 Tabelle dei requisiti

Requisito	Descrizione	Use Case
RFN-1	Invio di messaggi <i>broadcast</i> periodici per l'aggiornamento della <i>Neighbor Table</i> tramite stime di distanza basate su <i>RSSI_G</i>	UC3
RFN-2	Propagazione dei messaggi necessari al calcolo della distanza in salti (<i>hop count</i>) dalla <i>base station</i>	UC1, UC3
RFN-3	Calcolo a bordo delle forze attrattive verso i salti adiacenti e repulsive verso i vicini sotto la soglia di sicurezza, con produzione di output telemetrico	UC0
RFD-1	Applicazione vettoriale delle forze calcolate ai controllori di volo per l'attuazione del movimento del drone	UC0
RFD-2	Esecuzione di una dimostrazione in volo con nodo base, relay e drone che perde e recupera la connessione	UC0
RFD-3	Confronto sperimentale tra la variante attrattiva <i>weighted</i> e la variante <i>centroid</i>	UC0
RFN-4	Possibilità di simulare la perdita di segnale dalla <i>base station</i> tramite parametro configurabile, a scopo di test	UC2

Tabella 3.1: Tabella del tracciamento dei requisiti funzionali.

Requisito	Descrizione	Use Case
RQN-1	Il ciclo di controllo deve rispettare un periodo costante (es. 250 ms) per garantire un comportamento deterministico e prevedibile	UC0, UC1, UC3
Continua nella prossima pagina...		

Tabella 3.2 – Continuo della tabella

Requisito	Descrizione	Use Case
RQD-1	Il sistema deve produrre output telemetrico sufficientemente dettagliato da permettere la validazione a posteriori del comportamento dell'algoritmo	UC2

Tabella 3.2: Tabella del tracciamento dei requisiti qualitativi.

Requisito	Descrizione	Use Case
RVN-1	Il sistema deve essere eseguibile sull'architettura FreeRTOS della piattaforma Crazyflie 2.x, nel rispetto dei vincoli di memoria e risorse del microcontrollore	-
RVN-2	La stima della distanza tra i droni deve basarsi esclusivamente sul segnale $RSSI_G$ ricevuto tramite comunicazione $P2P_G$ Wi-Fi, senza l'ausilio di hardware UWB_G	-
RVN-3	Il numero massimo di vicini tracciabili contemporaneamente nella <i>Neighbor Table</i> è limitato staticamente (<code>MAX_NEIGHBORS</code>)	-
RVN-4	In assenza di hardware UWB_G per il calcolo dell'Angolo di Arrivo (AoA_G), l'algoritmo vettoriale deve operare assumendo una topologia di rete lineare deterministica	-

Tabella 3.3: Tabella del tracciamento dei requisiti di vincolo.

Capitolo 4

Progettazione e codifica

Questo capitolo descrive gli strumenti e le tecnologie utilizzati per lo sviluppo, il ciclo di sviluppo seguito, l'architettura del sistema realizzato e alcuni estratti significativi del codice implementato.

4.1 Tecnologie e strumenti

Per lo sviluppo sono stati utilizzati la documentazione ufficiale di Crazyflie e il repository GitHub ufficiale del progetto, di cui è stato creato un fork in cui è stata aggiunta l'applicazione `app_self_healing` all'interno della cartella `examples/`. Come ambiente di sviluppo è stato utilizzato Visual Studio Code, mentre la compilazione e il flashing del firmware sono stati effettuati tramite la macchina virtuale ufficiale fornita da Crazyflie.

A livello di piattaforma, il firmware Crazyflie è basato su FreeRTOS¹, un sistema operativo real-time che gestisce la schedulazione dei task tramite un ciclo periodico. L'applicazione sviluppata sfrutta le primitive di FreeRTOS (`vTaskDelay`, `xTaskGetTickCount`) per implementare il ciclo di controllo a 250 ms.

La comunicazione tra i droni avviene tramite l'API di comunicazione *P2P_G* offerta dal firmware (`p2pRegisterCB`, `radiolinkSendP2PPacketBroadcast`), che permette lo scambio di pacchetti *broadcast* tra i nodi radio senza un coordi-

¹FreeRTOS – Real-time operating system for microcontrollers. URL: <https://www.freertos.org/>.

natore centrale². Per l'esposizione runtime di parametri configurabili da `cfclient` (ad esempio il parametro `sig_loss` utilizzato per simulare la perdita di segnale) è stato utilizzato il sistema dei parametri del firmware (`PARAM_GROUP_START/PARAM_ADD`). Infine, l'attuazione dei movimenti calcolati avviene tramite l'API del *commander*, che riceve i setpoint di velocità tramite `commanderSetSetpoint`.

```
p2pRegisterCB(p2pcallbackHandler);
...
radiolinkSendP2PPacketBroadcast(&p_reply);
...
commanderSetSetpoint(&sp, 3);

PARAM_GROUP_START(selfhealing)
PARAM_ADD(PARAM_UINT8, sig_loss, &force_signal_loss)
PARAM_GROUP_STOP(selfhealing)
```

Codice 4.1: Registrazione delle callback e delle API di comunicazione e parametri

4.2 Ciclo di vita del software

Lo sviluppo del codice ha seguito un modello iterativo di tipo agile. Aniché separare nettamente le fasi di analisi, progettazione, codifica e collaudo, ogni iterazione consisteva in un breve ciclo di scrittura o modifica del codice in Visual Studio Code, compilazione tramite `make` all'interno della macchina virtuale ufficiale, flashing del firmware sul drone tramite USB, esecuzione di un test e osservazione dei risultati, sia visivamente (comportamento fisico dei droni, LED di stato) sia tramite i log di telemetria stampati su `cfclient` (`DEBUG_PRINT`). Sulla base di quanto osservato, il codice veniva quindi corretto o i parametri ritirati, ripetendo il ciclo.

Non sono stati adottati metodi formali tipici di framework agili strutturati come Scrum (*sprint planning, daily standup, retrospective*), trattandosi di un progetto individuale; tuttavia, la cadenza settimanale degli aggiornamenti con

²Peer to Peer API. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/p2p_api/.

il Prof. Corò (Sezione 2.2) ricorda, su scala ridotta, la struttura di uno *sprint* settimanale con relativa revisione.

Questo approccio empirico e iterativo si riflette in diverse scelte implementative: ad esempio, l'intervallo del ciclo di *broadcast* è stato portato da 50 ms, previsto dall'algoritmo originale, a 250 ms, dopo aver osservato che l'intervallo più breve rendeva instabile la connessione tramite `cfclient` (cfr. Rischio 2). Analogamente, è stato introdotto un *grace period* di 5 secondi all'avvio, per evitare che il drone entrasse in *mission mode* prima che la rete si stabilizzasse. Inoltre un altro esempio potrebbe essere il filtro topologico inserito per assegnare il giusto *hop count*, dato che soltanto dopo alcuni test si è potuto osservare come il drone n.2 (dati gli spazi ristretti) ottenesse il segnale direttamente dalla *base station* e non dal precedente drone nella catena.

4.3 Progettazione

Il sistema è organizzato in tre moduli principali:

- `self_healing.c`: contiene il ciclo di controllo principale, la gestione dei messaggi *P2P_G* in ricezione e l'integrazione con il controllore di volo;
- `self_healing_state.c/h`: gestisce la *Neighbor Table*, la struttura dati che mantiene le informazioni sui droni vicini rilevati tramite *broadcast P2P_G*;
- `self_healing_math.c/h`: implementa il calcolo del vettore di forze virtuali e la sua conversione in setpoint di velocità.

Il ruolo di ciascun drone all'interno della rete (*base station*, relay o nodo che simula la perdita di segnale) è determinato dalla configurazione fisica del dispositivo: tramite `cfclient` viene assegnato staticamente l'indirizzo radio del drone (rispettivamente 0xE7E7E7E700, 0xE7E7E7E701 e 0xE7E7E7E702 per i droni con ID 0, 1 e 2). All'avvio, il firmware estrae l'ultimo byte dell'indirizzo tramite `configblockGetRadioAddress()` per determinare il proprio identificativo (`my_id`), da cui deriva il ruolo assunto durante il ciclo di controllo.

Alla ricezione di un pacchetto $P2P_G$, la callback `p2pcallbackHandler` applica un filtro topologico (per limitare l'aggiornamento ai soli vicini diretti nella catena) e aggiorna la *Neighbor Table* tramite `state_update_neighbor`. La tabella ha capacità fissa e adotta una politica LRU_G : quando è piena, il vicino aggiornato meno di recente viene sostituito.

Ad ogni ciclo di controllo (250 ms), il drone rimuove dalla tabella i vicini non più attivi, stima il proprio *hop count* come il minimo tra quelli dei vicini più 1, verifica il timeout di connessione con la *base station* e trasmette il proprio stato in *broadcast*. Se in *mission mode*, calcola il vettore di forze attrattive/repulsive tramite `compute_force_vector` e lo converte in velocità tramite `force_to_velocity`, secondo un'integrazione di Eulero basata sulla seconda legge di Newton. Più precisamente, la seconda legge di Newton ($a = F/m$, con F la componente di forza calcolata su un asse e m la massa del drone, `DRONE_MASS`) viene utilizzata per ricavare l'accelerazione, che viene poi integrata sulla velocità del passo precedente secondo il metodo di Eulero esplicito ($v_t = v_{t-1} + a \cdot \Delta t$, con Δt pari al periodo del ciclo di controllo, `DELTA_T`). La velocità risultante viene infine limitata al valore massimo consentito (`V_MAX`) per ciascun asse, per ragioni di sicurezza.

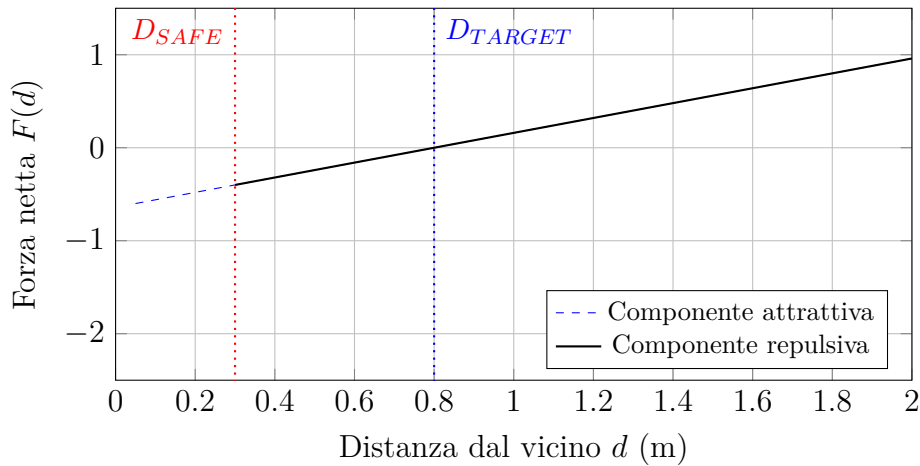


Figura 4.1: Forza netta $F(d)$ agente su un vicino hop-adiacente in funzione della distanza d . A destra di D_{SAFE} (0.3m) agisce solo l'attrazione, che si azzerà esattamente a D_{TARGET} (0.8m). Al di sotto di D_{SAFE} , la componente repulsiva si somma a quella attrattiva, generando una forza negativa (repulsione) che cresce esponenzialmente per evitare collisioni.

I parametri del modello, tarati sperimentalmente nel corso dei test sul campo,

sono riportati di seguito:

```
#define DRONE_MASS 0.029f // Massa del Crazyflie (kg)
#define DELTA_T 0.05f    // Intervallo di controllo, pari al tick del loop (s
)
#define D_SAFE 0.3f      // Distanza minima di sicurezza anti-collisione (m)
#define V_MAX 0.15f     // Velocità massima consentita (m/s)
#define K_ATT 0.8f       // Guadagno della forza attrattiva
#define K_REP 0.5f       // Guadagno della forza repulsiva
#define RSSI_REF -55.0f  // RSSI di riferimento a 1 metro (dBm)
#define N_EXPONENT 3.0f  // Esponente del modello log-distance (indoor)
#define D_TARGET 0.8f    // Distanza di equilibrio della molla attrattiva (m)
#define TARGET_HEIGHT 0.4f // Quota di volo target per i droni relay (m)
```

Codice 4.2: Parametri tarabili del modello a forze virtuali

4.4 Ambiente di simulazione

Come descritto nella Sezione 2.2, la difficoltà di isolare gli effetti del rumore $RSSI_G$ nei soli test fisici ha portato allo sviluppo di un ambiente di simulazione complementare, basato su ROS_G ³ e Gazebo⁴.

Le funzioni `compute_force_vector`, `force_to_velocity`, `update_neighbor` e `remove_dead_neighbors` sono state riportate in Python all'interno di un nodo ROS_G , mantenendo la stessa logica e gli stessi parametri (`K_ATT`, `K_REP`, `D_TARGET`, `V_MAX`, `D_SAFE`) del firmware originale; il parametro `sig_loss_active`, impostabile tramite `ros2 param set`, replica il parametro `sig_loss` esposto via `cfclient` nei test fisici.

La differenza sostanziale rispetto ai test fisici è che, conoscendo Gazebo la posizione esatta dei modelli, la chiamata a `rss_i_to_distance()` viene sostituita dal calcolo diretto della distanza euclidea reale tra i droni, eliminando così la fonte di rumore che aveva compromesso la validazione completa della logica di controllo nei soli test fisici. I risultati di questa validazione sono discussi nella Sezione 5.2.2 del Capitolo 5.

³*ROS 2 Documentation*. URL: <https://docs.ros.org/en/jazzy/>.

⁴*Gazebo*. URL: <https://gazebo.org/>.

4.5 Codifica

Di seguito sono riportati alcuni estratti di codice ritenuti maggiormente rappresentativi delle scelte implementative descritte.

La politica di sostituzione LRU_G nella *Neighbor Table*, utilizzata quando la tabella è piena:

```
uint8_t find_best_candidate_to_replace() {
    uint8_t candidate_idx = 0;
    for (uint8_t i = 1; i < active_neighbors_count; i++){
        if (neighbor_table[i].last_updated < neighbor_table[candidate_idx].
            last_updated) {
            candidate_idx = i;
        }
    }
    return candidate_idx;
}
```

Codice 4.3: Selezione del vicino da sostituire (politica LRU_G)

Il calcolo del vettore di forze, con componente attrattiva verso i salti adiacenti e repulsiva verso i vicini sotto la soglia di sicurezza:

```
if (neighbors[i].hop_count == (my_hop - 1) || neighbors[i].hop_count == (
    my_hop + 1)) {
    force_magnitude += K_ATT * (d - D_TARGET);
}

if (d < D_SAFE) {
    force_magnitude -= K_REP / (d*d);
}
```

Codice 4.4: Calcolo delle forze attrattive e repulsive

In assenza di hardware in grado di stimare l'angolo verso un vicino (cfr. Rischio 4), è stata adottata una mappatura deterministica basata sull'ID, valida per la topologia lineare utilizzata nei test:

```
static float get_angle_for_neighbor(uint8_t my_id, uint8_t neighbor_id) {
    if (neighbor_id < my_id) {
        return 0.0f;
    }
}
```

```
    } else {  
        return 3.14159f;  
    }  
}
```

Codice 4.5: Assegnazione deterministica dell'angolo verso un vicino

Per l'ambiente di simulazione, l'architettura logica è stata replicata fedelmente in Python all'interno del nodo *ROS_G* 2. Come discusso nella Sezione 4.4, la differenza chiave risiede nel modo in cui viene ottenuta la distanza *d*: per isolare i calcoli vettoriali dai disturbi radio, la funzione basata sull'*RSSI_G* è stata sostituita dal calcolo diretto della distanza euclidea ricavata dalle posizioni esatte fornite da Gazebo.

Il seguente frammento illustra l'estrazione della distanza pura e la successiva applicazione della logica vettoriale, mantenuta identica a quella scritta in C:

```
# Estrazione della distanza reale (Euclidea) tramite coordinate Gazebo  
d = math.sqrt((drone.x - other.x)**2 + (drone.y - other.y)**2)  
  
if d < 0.01:  
    d = 0.01 # Prevenzione della divisione per zero nel calcolo repulsivo  
  
# Applicazione delle medesime forze del firmware originale  
force_magnitude = 0.0  
if n.hop_count == (drone.hop_count - 1) or n.hop_count == (drone.hop_count +  
    1):  
    force_magnitude += K_ATT * (d - D_TARGET)  
if d < D_SAFE:  
    force_magnitude -= K_REP / (d * d)  
theta = get_angle_for_neighbor(drone.id, n.id)  
fx += force_magnitude * math.cos(theta)  
fy += force_magnitude * math.sin(theta)
```

Codice 4.6: Calcolo della distanza geometrica nel simulatore in sostituzione all'*RSSI_G*

Capitolo 5

Verifica e validazione

Questo capitolo verifica, tramite evidenze sperimentali raccolte tramite i log di telemetria del `cfclient`, il raggiungimento degli obiettivi definiti nella Sezione 2.4 del Capitolo 2 e formalizzati come requisiti nel Capitolo 3.

5.1 Verifica degli obiettivi obbligatori

5.1.1 Invio *broadcast* e aggiornamento della *Neighbor Table* (RFN-1)

Il primo test (Figure 5.1 e 5.2) mostra il drone con ID 1 all'avvio, seguito dalla ricezione di pacchetti *broadcast* dai droni con ID 0 e ID 2, con relativa stima di distanza tramite *RSSI_G*:

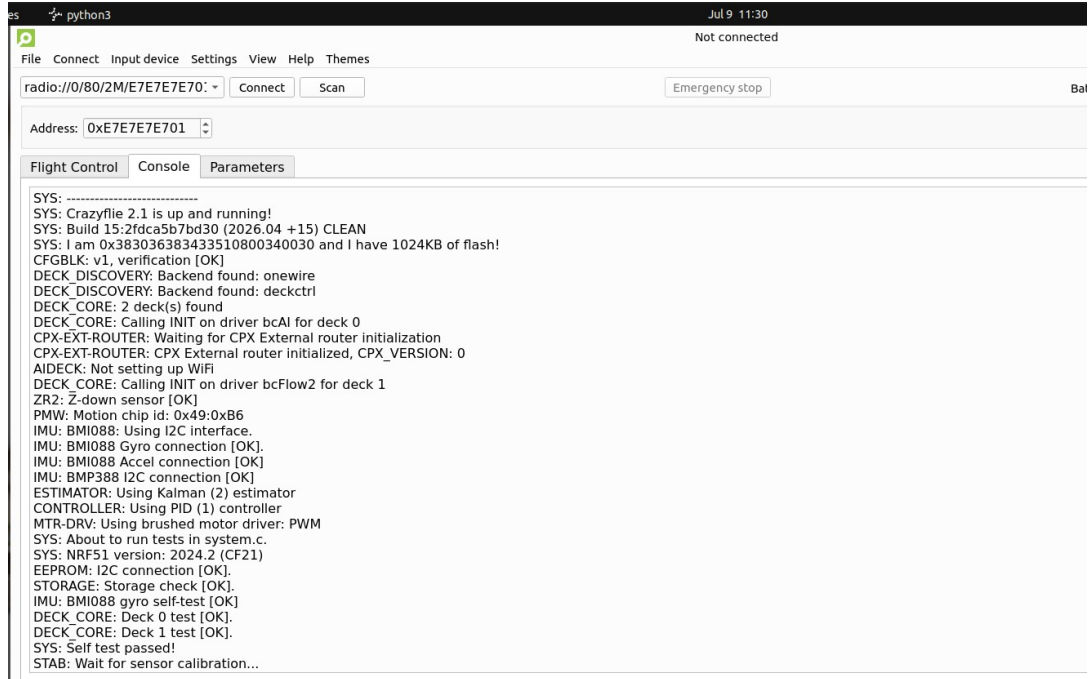
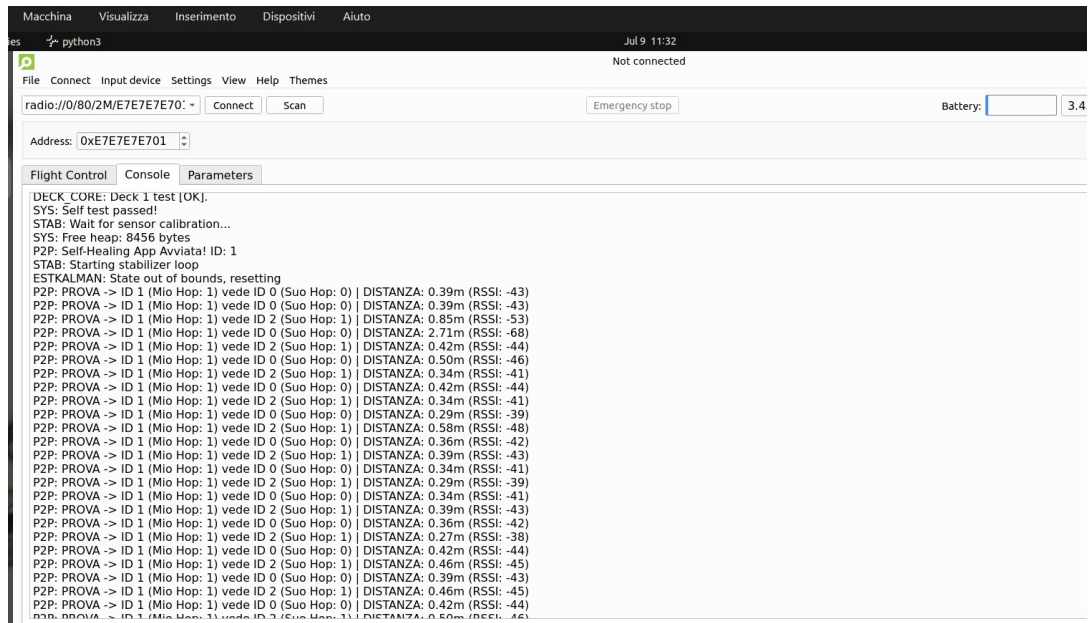


Figura 5.1: Avvio del drone ID 1 (primo test).

Figura 5.2: Log di ricezione *broadcast* e aggiornamento *Neighbor Table* (primo test).

Si osserva come le distanze stimate tramite $RSSI_G$ varino sensibilmente anche a parità di posizione fisica (ad esempio, valori tra 0.27m e 2.71m nello stesso breve intervallo), a conferma del fenomeno di *multipath fading* discusso nel Rischio 1. Il requisito RFN-1 può quindi considerarsi soddisfatto.

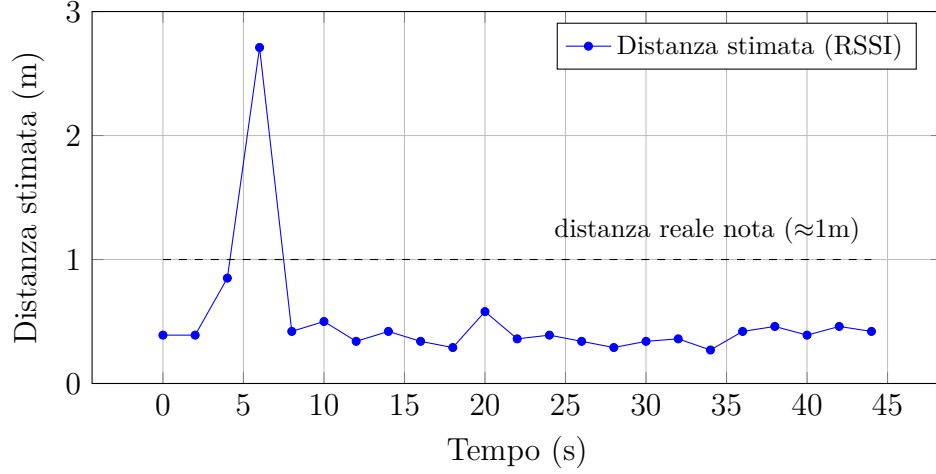


Figura 5.3: Distanza stimata tramite $RSSI_G$ verso i vicini nel tempo, a fronte di una distanza fisica reale nota e costante di circa 1m tra i droni (test statico, senza *mission mode*).

5.1.2 Propagazione dell'hop count (RFN-2)

Lo stesso primo test evidenzia però un'anomalia: il drone ID 2 viene riportato con *hop count* pari a 1, identico a quello del drone ID 1 stesso, anziché 2 come atteso nella topologia lineare Base-Relay-Drone. Questo comportamento, riscontrato prima dell'introduzione del filtro topologico descritto nella Sezione 4.3, ha motivato l'aggiunta del filtro nella callback di ricezione $P2P_G$.

Il secondo test, eseguito dopo l'introduzione del filtro, conferma la correzione:

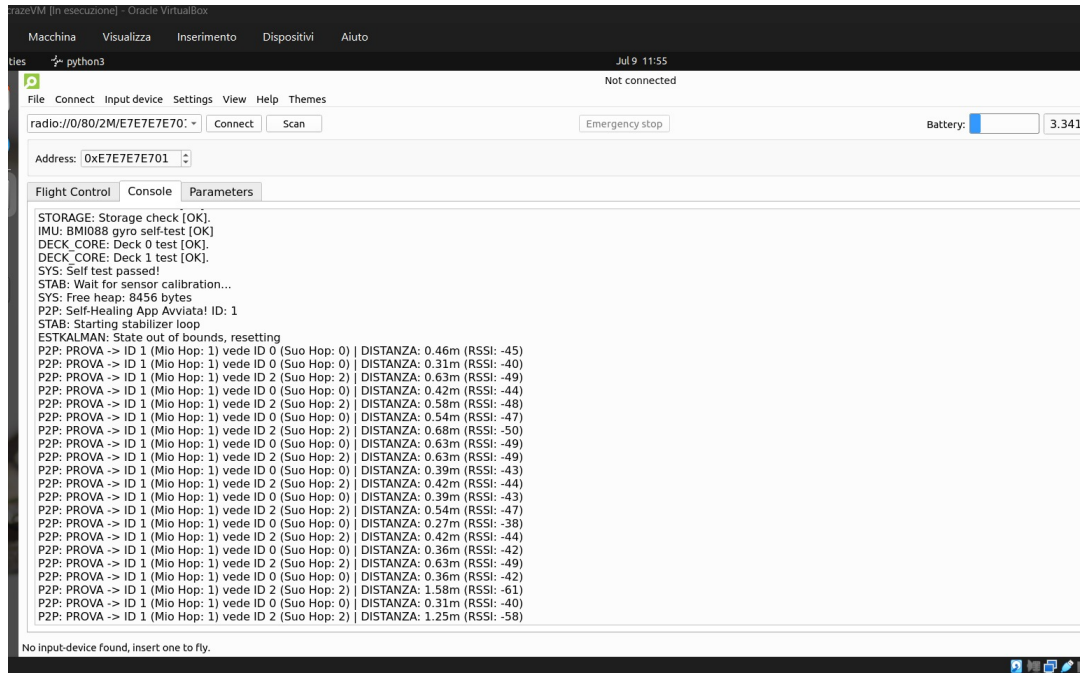


Figura 5.4: Log dell'hop count corretto dopo l'introduzione del filtro topologico (secondo test).

Qui il drone ID 0 viene correttamente riportato con hop 0 e il drone ID 2 con hop 2, coerentemente con la topologia lineare a tre nodi. Il requisito RFN-2 può quindi considerarsi soddisfatto.

5.1.3 Calcolo delle forze e output telemetrico (RFN-3)

Il terzo test (Figure 5.5 e 5.6) mostra l'avvio del drone e, successivamente, la produzione di telemetria relativa al calcolo delle forze:

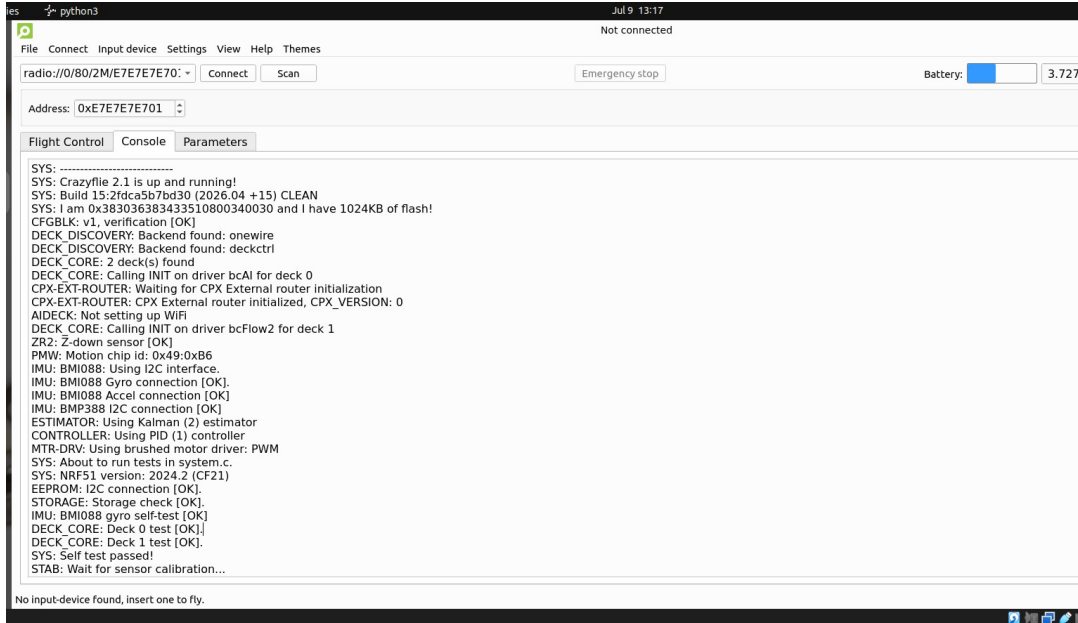


Figura 5.5: Avvio del drone (terzo test).

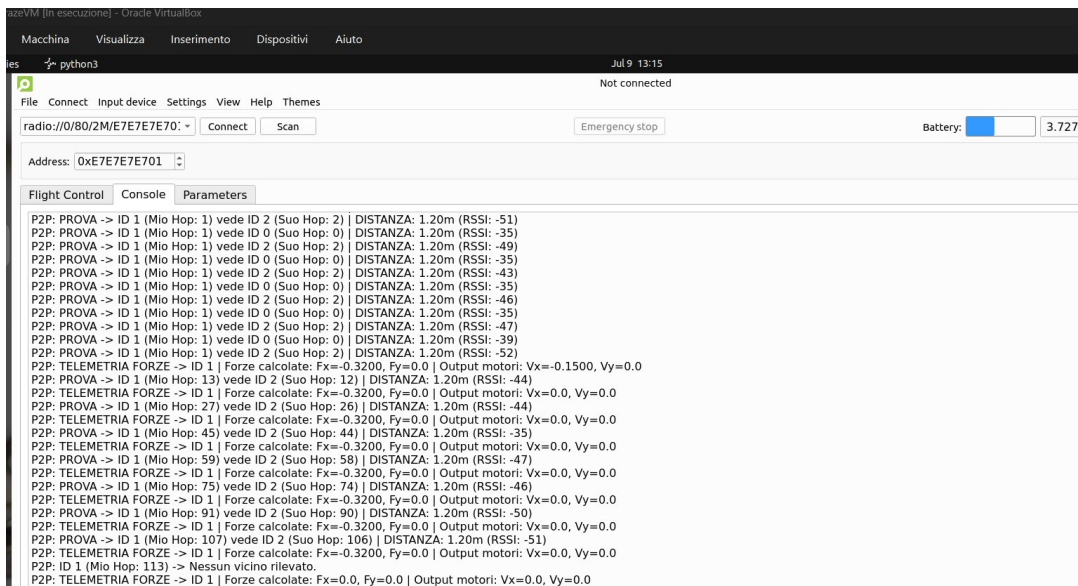


Figura 5.6: Log della telemetria forze e del comportamento dell'hop count in seguito allo spegnimento reale della base station (terzo test).

La riga "TELEMETRIA FORZE -> Fx=-0.3200, Fy=0.0 | Output motori: Vx=-0.1500, Vy=0.0" conferma che il vettore di forza viene calcolato e correttamente convertito in un setpoint di velocità non nullo, soddisfacendo il requisito RFN-3.

In questo stesso test si osserva un fenomeno non anticipato in fase di analisi dei rischi: a seguito dello spegnimento reale della *base station* (anziché tramite il

parametro `sig_loss`), l'*hop count* riportato dal vicino cresce progressivamente in modo incontrollato (13, 27, 45, 59, 75, 91, 107, 113), fino alla perdita completa del vicino (*Nessun vicino rilevato*). Questo comportamento è riconducibile al fenomeno noto in letteratura come *count-to-infinity*, tipico dei protocolli di routing a vettore di distanza: venendo meno il nodo di riferimento assoluto (hop 0), i due droni rimasti in *mission mode* derivano reciprocamente il proprio hop count da quello del vicino, incrementandolo senza un limite superiore, in assenza di meccanismi correttivi quali una soglia massima o un *hold-down timer*¹. Il problema è strutturale nei protocolli di routing a vettore di distanza perché ogni nodo conosce solo l'hop count dichiarato dai propri vicini diretti, non la topologia completa della rete: non potendo distinguere un aumento legittimo di distanza da una dipendenza circolare tra nodi, ciascun nodo continua a fidarsi del valore ricevuto e lo incrementa a propria volta. Il fenomeno è discusso ulteriormente nella Sezione 5.3.

5.2 Verifica degli obiettivi desiderabili

5.2.1 Applicazione vettoriale delle forze ai controllori di volo (RFD-1)

Come mostrato in Figura 5.6, il setpoint di velocità calcolato ($V_x = -0.1500$) viene effettivamente prodotto e inviato al controllore di volo tramite `commanderSetSetpoint`. Una dimostrazione video del movimento fisico risultante è disponibile al seguente [link](#). Si noti che, utilizzando in questo test una distanza fissa hardcoded (cfr. soluzione al Rischio 1) anziché una stima reale, il moto risultante è corretto in direzione ma privo di un meccanismo naturale di arresto: non variando la distanza percepita con l'avvicinamento reale dei droni, la forza attrattiva non si annulla mai, e i droni non convergono a una distanza di equilibrio stabile come avverrebbe con una stima di distanza accurata.

¹Charles Hedrick. *Routing Information Protocol*. URL: <https://www.rfc-editor.org/rfc/rfc1058>.

Il requisito RFD-1 può considerarsi soddisfatto per la parte di calcolo e attuazione; l'efficacia del movimento risultante ai fini del ripristino della connettività è discussa nel punto seguente.

5.2.2 Dimostrazione pratica in volo (RFD-2, parzialmente soddisfatto)

È stata eseguita una dimostrazione in volo con i tre droni nei rispettivi ruoli di base, relay e nodo che perde la connessione, documentata nel seguente [video](#). Tutte le componenti dell'algoritmo (rilevamento del timeout, calcolo delle forze, attuazione del movimento) risultano funzionanti e osservabili sia dal comportamento fisico dei droni sia dai log di telemetria. Tuttavia, a causa dell'inaffidabilità della stima $RSSI_G$ in ambiente indoor di dimensioni ridotte (Rischio 1), la direzione del movimento risultante non rifletteva correttamente l'obiettivo previsto: anziché avvicinarsi l'uno verso l'altro per ripristinare la connettività, i droni si sono mossi nella medesima direzione, proprio per colpa delle stime date dal segnale $RSSI_G$. L'esperimento è stato ripetuto più volte anche utilizzando l'apposito parametro `sig_loss`, ma i risultati osservati sono stati gli stessi.

Per verificare se tale esito fosse imputabile alla logica di controllo o esclusivamente al rumore della stima $RSSI_G$, l'algoritmo è stato eseguito anche nell'ambiente di simulazione descritto nella Sezione 4.4, in cui la distanza tra i droni è nota con precisione. Una dimostrazione video della simulazione è disponibile al seguente [link](#). A differenza del test fisico con distanza hardcoded (Sezione 5.2.1), in simulazione la forza attrattiva si annulla correttamente quando i droni raggiungono la distanza di equilibrio `D_TARGET`, permettendo loro di convergere e arrestarsi stabilmente. Questo risultato conferma che la logica di controllo implementata è corretta, e che l'esito solo parzialmente soddisfacente del requisito nei test fisici era imputabile esclusivamente all'inaffidabilità della stima $RSSI_G$ in ambiente indoor.

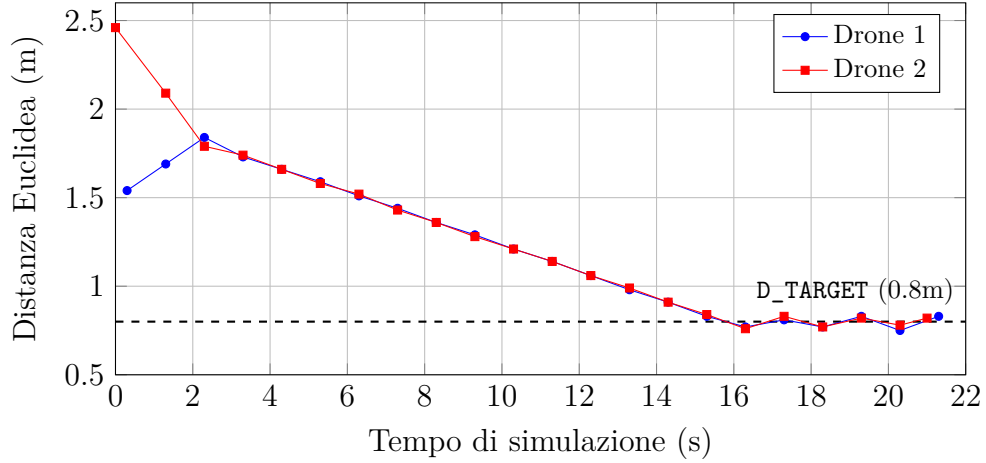


Figura 5.7: Andamento temporale delle distanze in simulazione: a seguito della perdita di connessione (tempo 0), i droni si riposizionano attivamente fino a convergere e arrestarsi in corrispondenza della distanza di equilibrio D_TARGET .

Il requisito RFD-2 può quindi considerarsi soddisfatto tramite validazione in simulazione, pur restando solo parzialmente dimostrato in condizioni fisiche reali.

5.2.3 Confronto tra variante *weighted* e variante *centroid* (RFD-3, non soddisfatto)

Questo obiettivo non è stato realizzato nell'ambito dello stage, principalmente per vincoli di tempo e per le difficoltà riscontrate con la stima $RSSI_G$ in ambiente indoor, che hanno reso prioritario concentrare gli sforzi sulla dimostrazione degli obiettivi obbligatori e nell'effettuare molteplici prove per tentare di realizzare a pieno il requisito RFD-2.

5.3 Limiti riscontrati

I test condotti confermano la correttezza funzionale dell'algoritmo implementato, ma evidenziano due limiti principali:

- l'inaffidabilità della stima $RSSI_G$ in ambiente indoor di piccole dimensioni (Rischio 1), che ha impedito la corretta convergenza spaziale dei droni nello scenario completo;

- il fenomeno di *count-to-infinity* osservato in seguito alla perdita totale (anziché parziale) della *base station*, che rappresenta un limite strutturale dell'algoritmo di hop discovery non completamente esplorato nella pianificazione iniziale.

Capitolo 6

Conclusioni

6.1 Consuntivo finale

Lo stage ha portato alla realizzazione di un protocollo di rete *self-healing* per sciame di droni Crazyflie, a partire da un algoritmo precedentemente validato solo tramite simulazione, sostituendo il livello di localizzazione originariamente basato su UWB_G con una stima di distanza basata sul segnale $RSSI_G$ su infrastruttura Wi-Fi. Il lavoro ha richiesto lo studio dell'architettura hardware e software della piattaforma Crazyflie, lo sviluppo di un protocollo di comunicazione $P2P_G$ ad-hoc, e l'implementazione a bordo del modello a forze virtuali che governa il comportamento di *self-healing* dello sciame, e la realizzazione di un ambiente di simulazione complementare basato su ROS_G 2 e Gazebo, sviluppato per validare la logica di controllo indipendentemente dal rumore della stima $RSSI_G$.

6.2 Raggiungimento degli obiettivi

Come discusso nel dettaglio nel Capitolo 5, tutti gli obiettivi obbligatori definiti nel piano di lavoro (invio *broadcast* periodico, propagazione dell'*hop count*, calcolo a bordo delle forze) sono stati soddisfatti, con evidenza sperimentale tramite log di telemetria.

Tra gli obiettivi desiderabili, l'integrazione con i controllori di volo è stata realizzata e verificata, così come una dimostrazione pratica in volo con tutti e tre i ruoli (base, relay, drone che perde la connessione); quest'ultima è tuttavia da considerarsi solo parzialmente soddisfatta in condizioni fisiche reali, poiché l'inaffidabilità della stima $RSSI_G$ in ambiente indoor di dimensioni ridotte ha impedito la corretta convergenza spaziale dei droni. Tale ipotesi è stata successivamente confermata da una validazione in ambiente di simulazione, in cui la stessa logica di controllo ha prodotto una convergenza corretta (cfr. Capitolo 5). Il confronto sperimentale tra la variante attrattiva *weighted* e la variante *centroid* non è stato invece realizzato, per vincoli di tempo dovuti alla necessità di ripetere più volte i test sul campo a causa delle difficoltà legate alla stima $RSSI_G$.

6.3 Conoscenze acquisite

Il progetto ha permesso di approfondire la programmazione embedded in linguaggio C su architettura FreeRTOS, con particolare attenzione alla gestione di task periodici e strutture dati condivise in un contesto concorrente. Sono state inoltre sviluppate competenze pratiche sulle reti wireless ad-hoc, sulla stima di distanza tramite $RSSI_G$ e sui suoi limiti in ambiente indoor, nonché sui limiti strutturali degli algoritmi di routing a vettore di distanza, come il fenomeno di *count-to-infinity* riscontrato durante i test. Sono state inoltre acquisite competenze di base nell'ecosistema ROS_G 2 e nel simulatore Gazebo, utilizzati per validare l'algoritmo in un ambiente privo di interferenze radio. Non meno rilevante è stata l'esperienza di sviluppo secondo un ciclo iterativo di tipo agile, basato sull'osservazione empirica del comportamento fisico del sistema e sull'adattamento continuo del codice sulla base dei risultati ottenuti.

6.4 Valutazione conclusiva

Portare un algoritmo validato solo in simulazione fino a un'implementazione funzionante su hardware reale si è rivelata un'esperienza estremamente forma-

tiva. Prima di questo stage il mondo dello sviluppo embedded, di FreeRTOS e delle reti wireless risultava un ambito completamente sconosciuto, non essendoci basi pregresse in tale settore.

Al di là delle competenze tecniche acquisite, discusse nella sezione precedente, l'aspetto più coinvolgente di questo stage è stato il riscontro pratico e fisico dello sviluppo: a differenza dello sviluppo software puro, dove il risultato del proprio lavoro resta confinato a uno schermo, qui ogni riga di codice si traduceva in un comportamento osservabile direttamente su un drone reale, comprensibile anche a chi non ha alcuna conoscenza di ciò che avviene internamente. Vedere per la prima volta un drone muoversi autonomamente sulla base delle forze calcolate dal codice, o riuscire finalmente a stabilire una connessione stabile dopo aver individuato la causa di un problema che inizialmente sembrava incomprensibile, sono stati alcuni dei momenti di maggiore soddisfazione dell'intero percorso.

In ogni caso non sono mancate le difficoltà. Oltre ai primi ostacoli nella configurazione della connessione radio, la parte più impegnativa del lavoro è stata di gran lunga la fase di sperimentazione pratica sul campo. Un primo approccio, in cui i droni tentavano di muoversi mantenendo il contatto con il suolo anziché volare, è stato abbandonato sia a causa della medesima inaffidabilità della stima $RSSI_G$ in ambiente indoor già discussa precedentemente, ma anche per l'attrito che i droni dovevano affrontare strisciando al posto di essere liberamente in volo; la quasi totalità del tempo restante è stata quindi dedicata a ripetuti tentativi di test in volo, alla ricerca di una configurazione capace di produrre un comportamento corretto nonostante il rumore del segnale. Proprio per superare in modo più rigoroso questo limite, isolando la logica di controllo dal rumore del segnale, è stata infine sviluppata anche una simulazione dedicata in ROS_G 2 e Gazebo (Sezione 4.4), che ha permesso di dimostrare la correttezza dell'algoritmo indipendentemente dalla qualità della stima di distanza disponibile.

Diversi aspetti del lavoro meriterebbero un approfondimento futuro. Anzitutto, ripetere i test in uno spazio fisico più ampio, in cui il fenomeno del *multipath fading* abbia un impatto minore, permetterebbe di valutare più correttamente l'efficacia del modello a forze virtuali anche in condizioni reali. Il

fenomeno di *count-to-infinity* osservato in seguito alla perdita totale della *base station* potrebbe inoltre essere mitigato introducendo un meccanismo di soglia massima sull'*hop count* o un *hold-down timer*, in linea con le soluzioni note in letteratura per i protocolli di routing a vettore di distanza. Inoltre, l'ambiente di simulazione realizzato potrebbe essere esteso per eseguire il confronto tra la variante attrattiva *weighted* e la variante *centroid* (RFD-3), non svolto nei test fisici, oltre che per osservare il comportamento di sciame composti da un numero maggiore di droni o in composizioni diverse. Infine, nell'ambiente di simulazione sarebbe possibile implementare una stima dell'angolo precisa calcolata tramite la posizione geometrica fornita dal simulatore, ma per attenersi il più possibile all'esperimento condotto in condizioni fisiche reali si è deciso di riportare l'esperimento nella stessa identica maniera e disposizione.

Sitografia

Component explanation of the Crazyflie 2.x. URL: <https://www.bitcraze.io/documentation/system/platform/cf2-components/> (cit. a p. 4).

FreeRTOS – Real-time operating system for microcontrollers. URL: <https://www.freertos.org/> (cit. a p. 15).

Gazebo. URL: <https://gazebo.org/> (cit. a p. 19).

Hedrick, Charles. *Routing Information Protocol.* URL: <https://www.rfc-editor.org/rfc/rfc1058> (cit. a p. 27).

Optical flow deck and floor texture. URL: <https://forum.bitcraze.io/viewtopic.php?t=3425> (cit. a p. 4).

Peer to Peer API. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/p2p_api/ (cit. a p. 16).

ROS 2 Documentation. URL: <https://docs.ros.org/en/jazzy/> (cit. a p. 19).