POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Tesi di Laurea Magistrale

Pianificazione della traiettoria di un APR tramite mappatura digitale del terreno per il monitoraggio incendi

Relatore:
Prof. Ing. Manuela Battipede
Correlatore:
Ing. Francesco De Vivo

Candidata:
Sara Senzacqua

Anno Accademico 2018/2019

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Capitolo 1

Sommario

L'obiettivo di questa tesi è approfondire il lavoro di pianificazione della traiettoria di un aeromobile a pilotaggio remoto operante in terra ignota il cui obiettivo consiste nel monitoraggio di un incendio. La ricostruzione della mappa circostante avviene sfruttando i dati ottenuti con un sensore LIDAR a basso costo, la mappa digitale così ottenuta viene aggiornata in real-time permettendo così di garantire l'*obstacle avoidance*. La pianificazione della traiettoria risulta necessaria al fine di automatizzare il movimento dell'aeromobile e ottimizzare così la sua funzione di monitoraggio.

La struttura di questo elaborato ricalca le varie fasi che hanno composto il lavoro:

- Approfondimento sullo stato dell'arte della categoria di velivolo;
- Approfondimento del software autopilota installato sul drone, del protocollo per la comunicazione e dei software di simulazione utilizzati;
- Studio degli algoritmi di pathfinding esistenti e selezione dell'algoritmo più adatto al caso di studio, implementazione dell'algoritmo per la pianificazione della traiettoria sfruttando la mappatura digitale ricreata dal sensore;
- Test atti a verificare il corretto funzionamento dell'algoritmo per confermarne la validazione;
- Conclusioni.

Capitolo 2

Introduzione

Un Sistema Aeromobile a Pilotaggio Remoto (SAPR) è definito dall'ENAC (Ente Nazionale per l'Aviazione Civile) come un sistema costituito da un mezzo aereo senza persone a bordo, utilizzato per fini diversi da quelli ricreativi e sportivi, e dai relativi componenti necessari per il controllo e comando (stazione di controllo) da parte di un pilota remoto. I SAPR vengono generalmente identificati con il sostantivo drone, termine che trae le sue origini dal ronzio dei primi modelli somigliante al rumore che fa il fuco, il maschio dell'ape domestica, in tedesco *drohne*. Alcuni degli acronimi più noti per identificare i droni sono:

- **RPA** (*Remotely Piloted Aircraft*);
- **UAV** (*Unmanned Aerial Vehicle*);
- **RPV** (*Remotely Piloted Vehicle*).

2.1 Cenni storici

Gli Aeromobili a Pilotaggio Remoto vennero originariamente usati per missioni considerate troppo rischiose per l'Uomo e benché la loro origine risieda nelle applicazioni militari, ad oggi il loro uso si è espanso in molteplici applicazioni civili quali ad esempio la ricerca scientifica, l'agricoltura, la sorveglianza e le operazioni di prevenzione e intervento per un'emergenza incendio.

Il primo uso documentato di un veicolo aereo da guerra senza equipaggio di bordo risale al Luglio 1849 quando gli austriaci usarono dei palloni imbottiti di esplosivo per bombardare Venezia.

Un successivo prototipo di velivolo senza pilota a bordo fu il *Ruston Procton Aerial Target* che venne costruito nel 1916 su idea dell'ingegnere Archibald Low il quale lavorava al sistema radar dell'esercito britannico. L'idea innovativa di Low fu quella di sviluppare un piccolo aereo caricato di materiale esplosivo da telecomandare a distanza. La mancanza di uno stabilizzatore di volo su tale velivolo fece perdere subito il controllo dell'oggetto motivo per il quale tale progetto fu presto abbandonato. Sempre durante la Prima Guerra Mondiale, gli ingegneri Peter Cooper Hewitt e Elmer Sperry aggiunsero al prototipo di Low uno stabilizzatore giroscopico arrivando così a realizzare il *Hewitt-Sperry Automatic Airplane*, anche noto come "bomba volante", dimostrando così la fattibilità di un aereo

senza persone a bordo. Tale velivolo fu inteso come una versione antesignana dei missili da crociera. Durante lo stesso periodo l'ingegnere Charles Kettering realizzò il *Kettering Bug*, un biplano equipaggiato con lo stesso sistema di controllo del velivolo di Hewitt e Sperry con l'introduzione di un sistema di sgancio della bomba. Quest'ultima configurazione non fu completamente sviluppata per via della fine della guerra.

Figura 2.1: DH.82 Queen Bee

Figura 2.2: Pubblicizzazione del Radioplane OQ-2

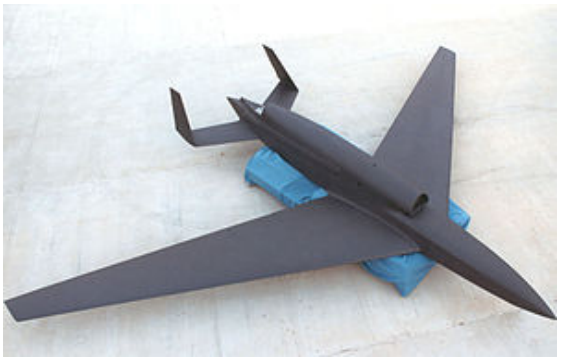
I primi successi a riguardo degli APR portarono allo sviluppo di velivoli radio-controllati

in Gran Bretagna e negli Stati Uniti nei decenni successivi al primo conflitto mondiale. Nel 1926 gli inglesi svilupparono i *Fairey Queen*, modificando tre *Fairey IIIF* come addestratori d'artiglieria radio-controllati, mentre nel 1935 venne sviluppato, a partire dal *de Havilland DH.82 Tiger Moth*, il primo vero drone della storia, il *DH.82 Queen Bee* (Figura 2.1), in quanto capace di decollare, svolgere una missione programmata e tornare alla base.

La prima produzione di droni su larga scala avvenne per opera di Reginald Denny, un appassionato di aeronautica e modellismo che fondò la compagnia *Radioplane Company.* Nel 1940 Denny vinse un contratto con l'esercito statunitense per il quale vennero prodotti 15000 velivoli *Radioplane OQ-2* (Figura 2.2).

Numerose configurazioni di aeromobili a pilotaggio remoto emersero durante la seconda guerra mondiale, in particolare dalla Germania, sia per addestrare l'artiglieria contraerea che per l'attacco aereo. Al termine della guerra entrarono in servizio droni con esoreattori, quali ad esempio il *GAF Jindivik* e il *Firebee I.* Gli APR rimasero poco più che aeromobili pilotati da remoto almeno fino alla guerra del Vietnam. La *U.S. Air Force* nel 1959, preoccupata a riguardo della perdita di piloti in territorio ostile, iniziò a valutare l'uso di aeromobili senza persone a bordo; in seguito all'abbattimento dell'U-2 per opera dell'Unione Sovietica gli Stati Uniti fecero partire il programma segretato *"Red Wagon"* per la produzione di APR, in particolare i *Ryan Model 147*, *Ryan AQM-91 Firefly* e *Lockheed D-21* (Figura 2.3) fecero la loro entrata nella guerra del Vietnam durante l'incidente del golfo del Tonchino nel 1964.

(a) *Ryan Model 147*



(b) *Ryan AQM-91 Firefly*

(c) *Lockheed D-21*

Figura 2.3: Droni militari utilizzati durante la guerra del Vietnam

Durante la guerra d'attrito per la prima volta vennero usati dall'intelligence israeliana degli APR tattici per la ricognizione, ottenendo foto del canale di Suez. Sempre Israele, durante la guerra dello Yom Kippur, svilupparono i primi APR capaci di sorveglianza in tempo reale. Le immagini e le esche radar prodotte da questi droni permisero la completa neutralizzazione delle difese aeree siriane all'inizio della prima guerra del Libano nel 1982.

Tra gli anni Ottanta e Novanta gli APR continuarono a dimostrare la loro capacità di essere macchine da combattimento più economiche e capaci, utilizzabili senza rischi per l'equipaggio. Nelle recenti guerre di pacificazione gli eserciti anglo-americani hanno iniziato a fare un sempre maggiore uso della tecnologia APR. I droni militari hanno il vantaggio di poter essere pilotati da remoto e, non dovendo tenere in conto i limiti della fisiologia umana, hanno prestazioni superiori e sono strutturalmente più semplici, economici e manutenibili. Oggi, infine, si ha una vera esplosione nel settore civile e ludico, con droni destinati agli utilizzi più svariati: riprese televisiva, gare di velocità (*Drone Racing League*) e persino trasporto di merci (Amazon, *Domino's Pizza*, UPS) e di persone (Airbus) in modalità a guida autonoma (Figura 2.4).

(a) Drone per il trasporto di Amazon

(b) Drone per il trasporto di *Domino's Pizza*

(c) Taxi-drone di Airbus

Figura 2.4: Utilizzi civili di UAV

2.2 Utilizzo degli APR

2.2.1 Ambito militare

Visto che gli aeromobili a pilotaggio remoto possono essere sfruttati per lavorare in condizioni in cui si presenta un elevato rischio per la vita umana, le applicazioni militari hanno trovato terreno fertile in questo mercato e vengono utilizzati per molteplici scopi:

- Ricognizione: gli APR utilizzati in ambito militare possono essere dedicati alla ricognizione in terra ostile, permettendo di ottenere informazioni in tempo reale anche a svariati chilometri di distanza; ne è un esempio il monitoraggio delle attività di Osama bin Laden da parte dell'esercito statunitense nel periodo antecedente la sua morte.

- Attacco aereo: i droni possono inoltre essere utilizzati anche per attaccare, qualora dotati di armamenti, evitando così la perdita ingente di forze militari umane in un attacco potenzialmente rischioso. Il loro utilizzo per attacchi aerei può inoltre colpire senza preavviso, fattore strategico determinante ad esempio nella lotta odierna contro il terrorismo.

- Sminamento: attraverso la cattura di immagini iperspettrali i droni sono anche usati per le operazioni di sminamento.

2.2.2 Ambito civile

Lo sviluppo tecnologico ha portato alla possibilità di equipaggiare gli APR con svariati sensori e telecamere che hanno aperto il mercato per l'utilizzo di questi velivoli per applicazioni civili. Ad esempio:

- Ricerca e soccorso (SAR, Search And Rescue): la ricognizione in tempo reale possibile grazie ai droni può essere sfruttata per facilitare le operazioni di ricerca e soccorso in situazioni di particolare criticità.

- Fotogrammetria aerea: con delle apparecchiature fotografiche installate sul drone risulta più semplice rispetto ad altre soluzioni aeree riprendere dall'alto un certo terreno per ottenere modelli digitali di elevazione (DEM, Digital Elevation Model), ortofoto e per il rilievo architettonico 3D. Con fotocamere più semplici si ottengono video o foto digitali più commerciali ma di largo uso e consumo anche non professionale.

- Controllo della biodiversità: i droni risultano essere un mezzo non invasivo per il monitoraggio di animali selvatici e dunque per il controllo della biodiversità di un ambiente.

- Telerilevamento: gli APR rendono possibile applicare la telerilevazione per monitorare zone agricole e lo stato di salute della vegetazione, inoltre è possibile volare in zone colpite da calamità naturali senza esporre a rischio delle persone e infine tenere sotto controllo le dispersioni termiche per incentivare misure contenitive volte alla ecosostenibilità.

- Sicurezza territoriale: gli aeromobili a pilotaggio remoto sono stati ingaggiati come strategia difensiva contro la criminalità, per esempio per disincentivare l'immigrazione clandestina, per segnalare e monitorare il traffico di sostanze stupefacenti e per il monitoraggio di siti archeologici contro la depredazione e il commercio illegale di reperti.
- Ampliamento della connessione Internet: diverse aziende stanno sfruttando gli APR per migliorare la copertura Internet nel mondo.

2.3 Classificazione e normativa

Secondo la classificazione del 2011 della UVS International (*Unmanned Vehicle System International*) i SAPR possono essere suddivisi in relazione a parametri come le dimensioni, il peso, la tipologia di motore, il payload, il range operativo, la quota e l'autonomia di volo. Secondo tali parametri è possibile distinguere tre grandi categorie di SAPR tattici, strategici e per operazioni speciali (Tabella 2.1). All'interno della categoria degli APR tattici rientrano praticamente tutti i velivoli utilizzati per le applicazioni in ambito civile, in particolare nella maggior parte dei casi vengono utilizzati gli APR appartenenti alla sotto-categoria Mini o Micro.

Tabella 2.1: Classificazione degli APR secondo la *UVS International*

Categoria	Acronimo	Range Operativo [km]	Quota di volo [km]	Durata del volo [h]	MTOW [kg]
Tattici					
Nano	H	<1	100	<1	<0,025
Micro	μ	<10	250	1	<5
Mini	Mini	<10	150 - 300	<2	<30
Close Range	CR	10 - 30	3000	2 - 4	150
Short Range	SR	30 - 70	3000	3 - 6	200
Medium Range	MR	70 - 200	5000	6 - 10	1250
Low Altitude Deep Penetration	LADP	>250	50 - 9000	0,5 - 1	350
Low Altitude Long Endurance	LALE	>500	3000	>24	<30
Medium Altitude Long Endurance	MALE	>500	14000	24 - 48	1500
Strategici					
High Altitude Long Endurance	HALE	>2000	20000	24 - 48	12000
Operazioni speciali					
Unmanned combat aerial vehicle	UCAV	1500	10000	2	10000
Lethal	LETH	300	4000	3 - 4	250
Decoy	DEC	0 - 500	5000	<4	250
Stratospheric	STRATO	>2000	>20000 and <30000	>48	Da definire
Exo-stratspheric	EXO	Da definire	<30000	Da definire	Da definire
Space	SPACE	Da definire	Da definire	Da definire	Da definire

L'ICAO (*International Civil Aviation Organization*), indipendentemente dalla presenza a bordo del pilota, definisce tutte le categorie di velivolo come "aeromobile" e di conseguenza i concetti normativi non subiscono alterazioni. In particolare, esiste un quadro normativo generale e gli APR di peso inferiore ai 150 kg sono di pertinenza delle singole autorità aeronautiche nazionali, l'ENAC (Ente Nazionale per l'Aviazione Civile) in Italia. L'ENAC stabilisce normative differenti a seconda delle diverse fasce di peso:

- SAPR inoffensivi (minori o uguali di 300 g);
- SAPR tra i 0,3 kg e i 2 kg;

- SAPR tra i 2 kg e i 25 kg;
- SAPR sopra i 25 kg.

Tra i tanti impieghi per cui possono essere usati i droni risulta necessario distinguerne l'uso ludico da quello professionale; tale distinzione è evidenziata nei regolamenti ENAC anche con l'utilizzo di una diversa terminologia, infatti con il termine Aeromodello vengono chiamati i SAPR utilizzati per attività non professionali. Con i SAPR è possibile effettuare operazioni specializzate non critiche per le quali è obbligatorio richiedere il riconoscimento, e operazioni specializzate critiche per le quali è necessario chiedere l'autorizzazione.

2.4 Architettura del sistema aeromobile a pilotaggio remoto

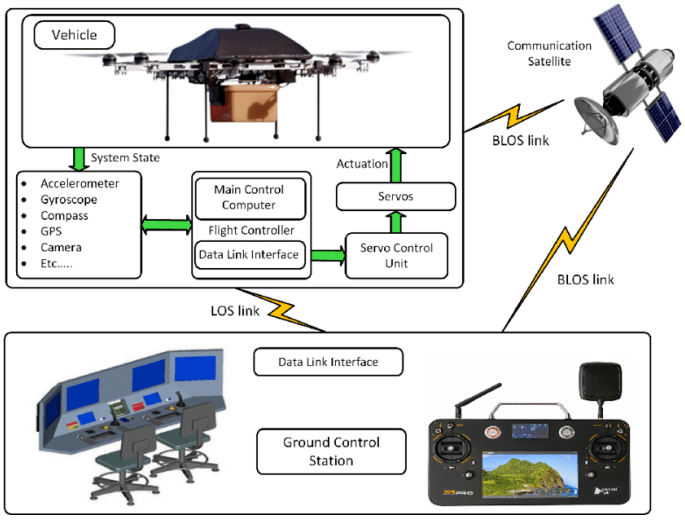


Figura 2.5: Architettura del sistema APR

Come sintetizzato in Figura 2.5, gli aeromobili a pilotaggio remoto sono un sistema complesso composto da molteplici parti; tali componenti verranno approfonditi in questa sezione del capitolo.

2.4.1 Corpo del velivolo

Gli aeromobili a pilotaggio remoto si presentano in due principali tipologie per ciò che riguarda il corpo del velivolo, quelli con struttura ad ala fissa e i multirotore (Figura 2.6); gli APR ad ala fissa oltre all'ala possono avere una o più code oppure non avere coda ed essere quindi velivoli tutt'ala, i multirotori presentano un numero variabile di rotori, solitamente 4, 6 o 8. Ciò che hanno in comune queste due tipologie è la mancanza del cockpit e dell'ECLSS (*Environmental Control and Life Support System*), caratteristica che sottolinea la mancanza di persone a bordo.

(a) APR ad ala fissa

(b) APR multirotore

Figura 2.6: Principali tipologie di APR

2.4.2 Hardware

Per ciò che riguarda l'hardware degli APR ad oggi, seguendo il trend di evoluzione informatica, si ha la *MicroController Unit* (MCU) in elettronica digitale, nata come alternativa al microprocessore. Dalla nascita dei microcontrollori si è implementato il single-board computer, una scheda elettronica che opera praticamente come un intero computer; tale configurazione si sviluppa attorno ai system-on-a-chip (SoC), circuiti integrati comunemente usati nelle applicazioni *embedded* per via della loro dimensione ridotta. Un esempio di single-board computer è il Raspberry Pi che è inoltre l'hardware installato nel velivolo oggetto di questo lavoro di tesi. Parte della dotazione hardware di un APR consiste nel *Flight Controller* (FC), responsabile del controllo e della stabilità del volo.

2.4.3 Software

Gli aeromobili a pilotaggio remoto sono sistemi che richiedono una rapida risposta al variare dei dati forniti dai sensori a bordo, per cui risulta fondamentale per il controllo del volo il software dell'autopilota. Alcuni software usati per utilizzi civili e disponibili in *open-source* sono:

- *PX4 autopilot*;
- *ArduCopter*;
- *LibrePilot*;
- *Paparazzi*.

Per quanto riguarda il velivolo approfondito in questo lavoro di tesi, il software autopilota considerato è *ArduCopter* e verrà approfondito nel prossimo capitolo.

2.4.4 Sensori e attuatori

Un sistema complesso come un drone possiede molteplici sensori a seconda delle informazioni necessarie da ottenere per il suo corretto funzionamento. Si possono dunque individuare delle categorie:

- Sensori propriocettivi: misurano variabili interne al velivolo quali la velocità dei rotori o il livello di carica della batteria;

- Sensori esterocettivi: misurano variabili esterne al velivolo come la distanza dagli ostacoli;

- Sensori di posizione e movimento: forniscono informazioni a riguardo della posizione e dell'assetto del velivolo. A seconda della quantità di gradi di libertà (DOF, *Degrees Of Freedom*) da valutare si definisce la quantità e dei sensori a bordo, 6 DOF necessitano di un giroscopio e di un accelerometro a 3 assi (una tipica IMU, *Inertial Measurement Unit*), per misurare 9 DOF si aggiunge all'IMU una bussola, con 10 DOF si necessita anche di un barometro mentre con 11 DOF si include un ricevitore GPS (*Global Positioning System*).

Nel drone approfondito in questa tesi si considera la presenza di molteplici sensori, tra i propriocettivi si annoverano quelli necessari al corretto funzionamento dell'apparecchio e al controllo del suo stato, tra i sensori esterocettivi risulta presente un sensore con tecnologia LIDAR (*Laser Imaging Detection and Ranging*), necessario per la ricostruzione della mappa del territorio e per garantire l'*obstacle avoidance*, mentre tra i sensori di posizione e movimento risultano presenti quelli precedentemente elencati atti a garantire una misurazione di 11 DOF.

Gli attuatori installabili su un aeromobile a pilotaggio remoto sono utili a svolgere vari compiti:

- Attuatori lineari e rotativi: necessari a convertire energia in movimento meccanico;

- Attuatori del *Flight Control System*: necessari a muovere le superfici di controllo per il controllo dell'assetto e della direzione di volo del velivolo.

- ESC (*Electronic Speed Controller*): necessario a controllare la velocità e il verso di rotazione dei motori.

2.4.5 Alimentazione

I droni di piccole dimensioni utilizzano come alimentatori delle batterie ricaricabili ai polimeri di Litio (LiPo), mentre i droni di grandi dimensioni presentano i motori tipici degli aeromobili convenzionali. Nonostante la densità energetica di un'alimentazione elettrica sia molto minore di quella a idrocarburi, spesso viene scelta in quanto i motori elettrici sono più silenziosi. Visto il campo di applicazione, i motori devono essere progettati affinché l'alimentazione sia sufficiente a garantire il decollo verticale. Al centro del drone solitamente si trova la PMU (*Power Management Unit*) che si occupa della distribuzione di corrente dalla batteria ai motori, ciascuno per ogni rotore presente sul velivolo. Fondamentale è la presenza per ogni motore dell'ESC, presentato nel paragrafo precedente, in modo tale che ogni motore risulti collegato al *Flight Controller*. La catena di alimentazione si sviluppa dunque a partire dalla batteria che trasmette corrente alla piastra di distribuzione che a sua volta la invia alla centralina di volo, alla ricevente radio, agli ESC e infine ai motori.

2.4.6 Comunicazione

Gli APR utilizzano la comunicazione radio per il controllo da remoto e per lo scambio di foto, video o altri dati. Il segnale radio inviato dall'operatore può provenire da fonti differenti:

- GCS (*Ground Control Station*): ricetrasmittente radio di natura differente (PC, smartphone, tablet) gestita da un operatore per mantenere il contatto con il velivolo;
- Un altro velivolo;
- Rete remota: applicazione militare di comunicazione bidirezionale via satellite.

All'inizio il canale di comunicazione era a banda stretta e disponibile esclusivamente dalla stazione di terra al drone, in seguito il canale è diventato bidirezionale. A seconda della distanza tra l'aeromobile e la stazione di terra la comunicazione è LOS (*Line Of Sight*) per distanze contenute oppure BLOS (*Beyond Line Of Sight*) per distanze molto grandi, quest'ultima comunicazione avviene tramite segnale satellitare e quando la distanza tra stazione di terra e il velivolo tale da non poter usare il canale radio classico. Dato che le applicazioni odierne richiedono uno scambio ingente di informazioni il canale di comunicazione è stato reso a banda larga, di modo da inviare in un unico canale molteplici dati. In particolare il protocollo che sta prendendo piede per le comunicazioni con i velivoli è il MAVLink (*Micro Air Vehicle Link*).

Capitolo 3

Autopilota

Come accennato precedentemente, il software autopilota installato sul drone oggetto di questa tesi è la versione per APR di ArduPilot, ovvero ArduCopter. ArduPilot è uno dei software open source più avanzati e completi in circolazione, è stato sviluppato in oltre 5 anni da un team di professionisti nel settore informatico; in quanto progetto open source è in continua evoluzione e in fase di sviluppo. Questo software è in grado di controllare diverse tipologie di veicoli:

- Velivoli ad ala fissa anche con capacità di decollo verticale;
- Droni multirotore;
- Elicotteri;
- Rover terrestri;
- Barche;
- Sottomarini;
- Antenna tracker.

Da tali veicoli derivano i nomi dei vari firmware: Copter, Plane, Rover, Sub e AntennaTracker.

I codici sorgente di ArduPilot sono archiviati e gestiti su GitHub, un servizio di hosting per progetti software. Sebbene ArduPilot non produca alcun hardware, il firmware ArduPilot funziona su dispositivi differenti per controllare veicoli a pilotaggio remoto di tutti i tipi; i vari pacchetti possono essere testati su hardware genericamente intesi come un microcontrollore collegato ai sensori necessari alla navigazione, quindi giroscopi, accelerometri, bussola, altimetro e antenna GPS, e ai sensori opzionali necessari al corretto svolgimento della missione in oggetto.

ArduPilot possiede svariate caratteristiche, tra cui si annoverano quelle comuni a tutti i veicoli del pacchetto:

- Modalità di volo completamente autonoma, semi-autonoma, e completamente manuale, missioni programmabili con waypoint 3D e geo-fencing opzionale.

- Possibilità di effettuare simulazioni con una grande varietà di simulatori, tra cui ArduPilot SITL.

- Sono supportati una grande quantità di sensori per la navigazione, inclusi vari modelli di RTK (*Real-Time Kinematic*) GPS, i tradizionali L1 GPS, i barometri, i magnetometri, i telemetri a laser e a sonar, i sensori a flusso ottico, i transponder ADS-B, i sensori ad infrarosso, i sensori per misurare la velocità, e infine i dispositivi necessari a catturare foto e video.

- Sono garantite le comunicazioni via SPI (*Serial Peripheral Interface*), I^2C (*Inter-Integrated Circuit*), CAN (*Controller Aera Network*) Bus, UART (*Universal Asynchronous Receiver-Transmitter*) e USART (*Universal Synchronous-Asyncronous Receiver-Transmitter*) e SMBus (*System Management Bus*).

- Progettazione fail-safe in caso di perdita di contatto radio, di GPS e violazione di un limite prestabilito e di livello minimo di batteria rimanente.

- Supporto alla navigazione in ambienti con accesso negato al GPS con posizionamento basato sulla visualizzazione, con flusso ottico, con tecniche SLAM (*Simultaneous Localization And Mapping*) e con posizionamento UWB (*Ultra Wide Band*).

- Supporto per attuatori come paracadute o pinza magnetica.

- Supporto per motori con e senza spazzole.

- Supporto e integrazione di sospensioni gimbal per foto e video.

- Integrazione e comunicazione con computer secondari, definiti in seguito "companion" computer.

- ArduPilot ha una vasta community online dedicata ad aiutare utenti tramite forum.

Tra le specifiche esclusive della categoria Copter si hanno:

- Modi di volo: Stabilize, Alt Hold, Loiter, Return-to-Launch, Auto, Acro, AutoTune, Brake, Circle, Drift, Guided, Guided NOGPS, Land, PosHold, Sport, Throw, Follow me, Simple, Super Simple, Avoid ADSB.

- Autotuning.

- Varie tipologie di architettura supportate tra cui tricotteri, quadricotteri, esacotteri, octocotteri sia piani che coassiali, e configurazioni personalizzate.

- Supporto per elicotteri elettrici e a propellente tradizionali, elicotteri monorotore e elicotteri tandem.

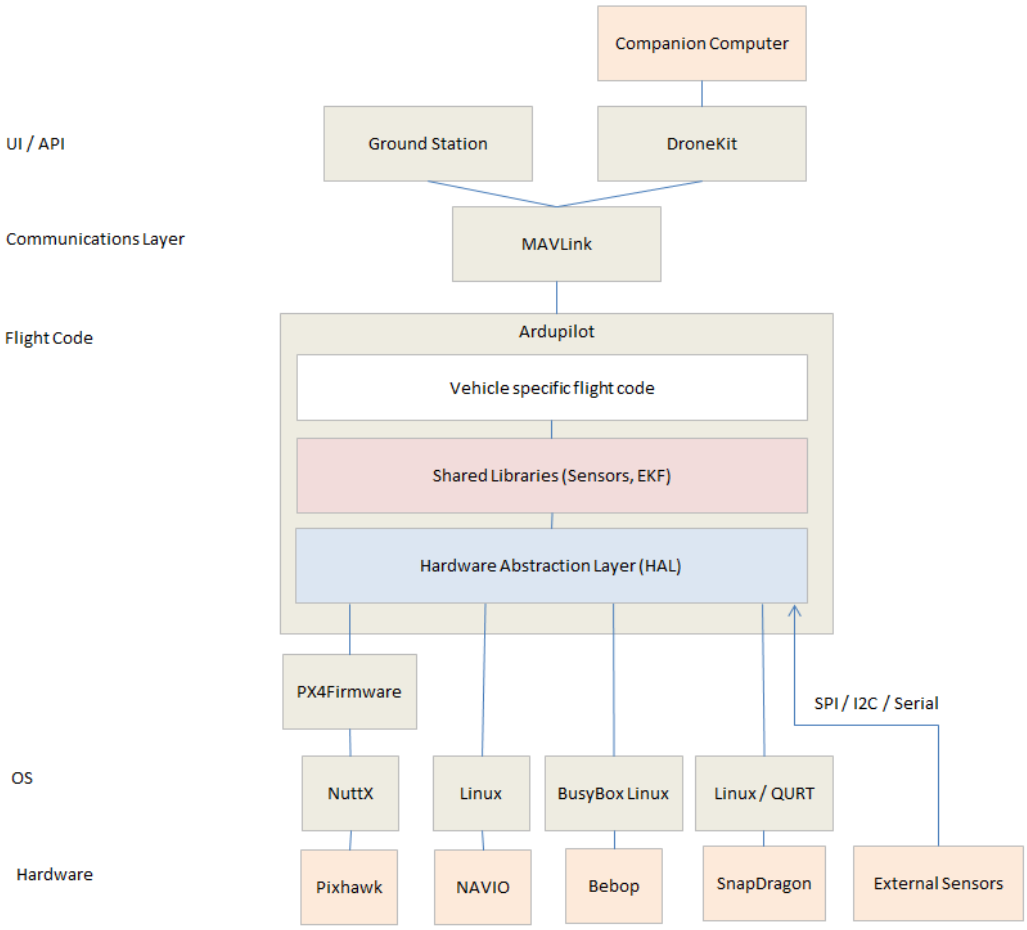


Figura 3.1: Architettura del firmware ArduCopter

3.1 ArduCopter

In Figura 3.1 è riportata l'architettura del firmware ArduCopter. Come si vede dall'immagine si hanno due possibilità di interfaccia utente, la Ground Station fisica oppure l'utilizzo di un Companion Computer, per esempio DroneKit una delle soluzioni più utilizzate. Data la natura sperimentale della tesi, per testare il corretto funzionamento del codice implementato la soluzione scelta è stata la seconda per procedere dunque in ambiente di simulazione. La comunicazione con il codice autopilota, indipendentemente dalla tipologia di sorgente, avviene tramite protocollo MAVLink che verrà approfondito nel prossimo paragrafo.

Il codice è strutturato con il codice Copter principale che si collega a svariate librerie; in tali librerie è compresa la sensoristica di bordo, le informazioni di posizione inerziale, l'altitudine barometrica e la localizzazione GPS sono elaborate dalla libreria EKF in cui è implementato l'algoritmo *Extended Kalman Filter*, da cui il nome. Questo algoritmo infatti usa le informazioni ottenute da giroscopi, accelerometro, bussola (o magnetometro), GPS, velocità dell'aria e pressione barometrica per stimare posizione velocità e assetto del

velivolo in volo. Il vantaggio dell'algoritmo EKF rispetto ad algoritmi di filtro più semplici sta nel fatto che, utilizzando contemporaneamente tutte le misurazioni disponibili, riesce ad individuare meglio le rilevazioni affette da errori significativi; in questo modo il velivolo diventa meno suscettibile ai guasti di un singolo sensore. Un'altra caratteristica dell'algorito EKF è che è in grado di stimare il campo magnetico terrestre, ciò lo rende meno sensibile agli errori di calibrazione della bussola. La logica dell'algoritmo si può riassumere come segue:

1. La misura inerziale delle velocità angolari viene integrata per calcolare la posizione angolare.

2. La misura inerziale delle accelerazioni viene ottenuta convertendo la posizione angolare da assi corpo X, Y, Z a assi NED (North, East, Down) e applicando la correzione per la forza di gravità.

3. Le accelerazioni vengono integrate per ottenere la velocità.

4. La velocità viene integrata per calcolare la posizione.

5. Viene stimato il rumore nelle misurazioni effettuate da giroscopi e accelerometri per calcolare la crescita dell'errore nelle misurazioni di angoli, velocità e posizioni. Queste stime di errore vengono conservate in una matrice delle covarianze.

6. Quando arriva una misurazione GPS, il filtro calcola la differenza tra la posizione stimata attraverso i primi 4 passaggi e la posizione del GPS. Questa differenza prende il nome di "Innovazione".

7. L'Innovazione, la matrice delle covarianze e l'errore nella misurazione GPS vengono combinati per calcolare una correzione di ciascuna rilevazione.

8. Il livello di incertezza nella determinazione dello stato del velivolo risulta ridotto, dunque l'algoritmo aggiorna la matrice delle covarianze e ricomincia dall'inizio.

La modalità di volo viene inizializzata all'inizio del codice e a seconda di tale modalità viene interpretato l'input del pilota e impostati i target da raggiungere. L'algoritmo EKF applicato alla sensoristica di bordo rende dunque più precisa la misurazione della posizione e dell'assetto del drone. Le librerie di controllo posizione, controllo assetto e controllo motori vengono dunque richiamate dal codice principale per garantire il controllo della navigazione del drone. La libreria di controllo posizione utilizza controllori PID per controllare separatamente il piano orizzontale (assi X e Y) e l'asse verticale (asse Z); i due controlli sono mantenuti separati perché alcune modalità di volo, per esempio AltHold, richiedono solo il controllo dell'asse Z. Per il controllo degli assi X e Y, il cui diagramma è riportato in Figura 3.2, viene usato un controllore P per convertire l'errore di posizione in velocità target, in seguito un controllore PID converte l'errore di velocità in accelerazione necessaria che a sua volta è convertita in angolazione e inviata alla libreria di controllo assetto. Per il controllo dell'asse Z viene usato un controllore P per convertire l'errore di posizione in target di velocità di salita che un ulteriore controllore P converte in target di accelerazione; un controllore PID converte tale accelerazione in manetta necessaria e invia il segnale alla

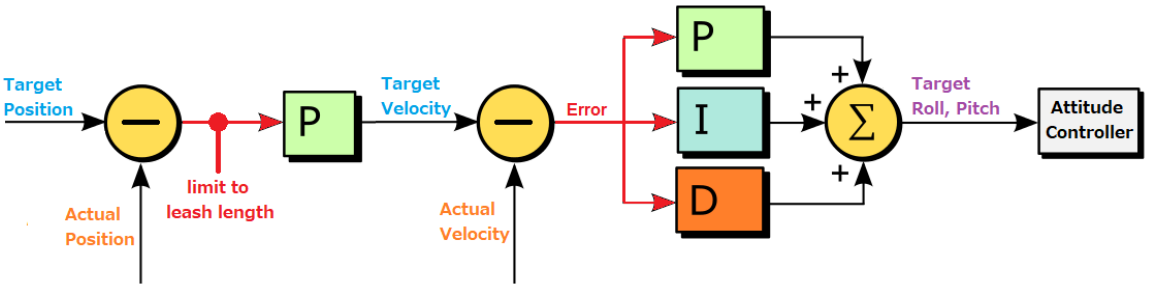


Figura 3.2: Diagramma di controllo dell'assetto nel piano orizzontale

libreria di controllo assetto. La libreria di controllo assetto utilizza i vari input per richiedere ai motori la manetta necessaria ad azzerare gli errori di angolazione. I valori di manetta necessaria vengono gestiti dalla libreria di controllo motore, le richieste assolute ai motori vengono inviate sotto forma di PWM (*Pulse-Width Modulation*) alle ESC, come anticipato nel capitolo precedente.

Il blocco HAL rappresentante lo strato di astrazione dall'hardware rende più versatile il colloquio I/O tra dispositivi fisici e codice. L'output, così come detto precedentemente, in questo lavoro di tesi è stato gestito in ambiente simulato senza fare riferimento ad un hardware fisico in particolare.

3.2 Ambiente di simulazione

SITL (*Software In The Loop*) è un simulatore che permette di eseguire ArduPilot sul PC senza l'utilizzo di alcun hardware. L'eseguibile di questo software deriva dalla compilazione del codice sorgente scritto in C++. Durante una simulazione con tale software i dati di volo riportati derivano da modelli di dinamica del volo, dato che ArduPilot integra un vasto numero di simulatori anche con SITL è possibile simulare:

- Velivoli multirotore;
- Velivoli ad ala fissa;
- Rover terrestri;
- Veicoli subacquei;
- Videocamere con sostegno gimbal;
- Antenna tracker;
- Molteplici sensori opzionali.

Uno dei vantaggi sostanziali dell'utilizzo di ArduPilot su SITL è la possibilità di sfruttare gli strumenti di sviluppo desktop per C++, per esempio debugger interattivi e strumenti di analisi statica e dinamica, permettendo così lo sviluppo e il test di nuove funzionalità dell'autopilota.

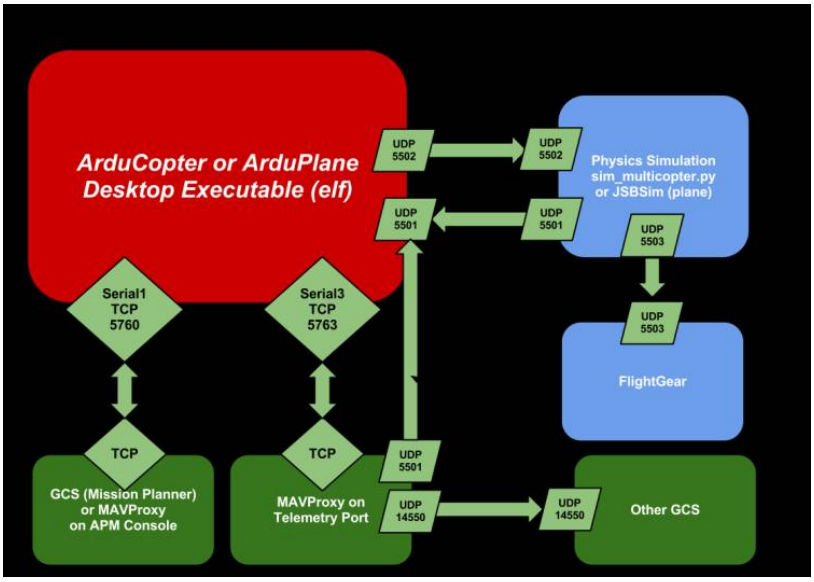
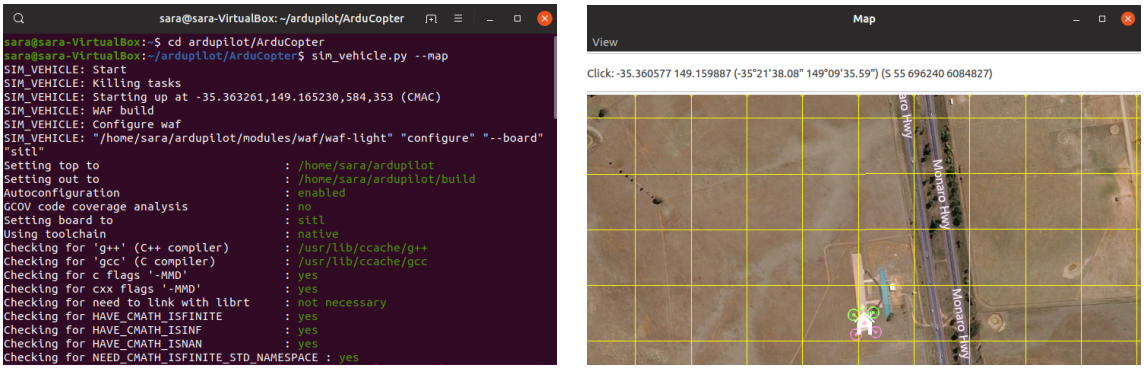


Figura 3.3: Architettura del software SITL

SITL può essere usato su Linux e Windows, per questa tesi è stato installato in ambiente macchina virtuale su Ubuntu versione 19.04. Una volta effettuata l'installazione, il software autopilota viene fatto partire da terminale, una volta nella directory ArduCopter, con il comando

$$sim\_vehicle.py\ --map$$

Tale comando permette la visualizzazione della mappa per controllare il comportamento del drone. In Figura 3.4 si può vedere il risultato del comando per eseguire il software e la mappa nella quale il drone, raffigurato come un quadrirotore, si trova in corrispondenza della posizione di partenza, indicata col simbolo di una casa.



(a) Autopilota (b) Mappa

Figura 3.4: Esecuzione dell'autopilota tramite SITL e visualizzazione del drone su mappa

3.3 Protocollo MAVLink

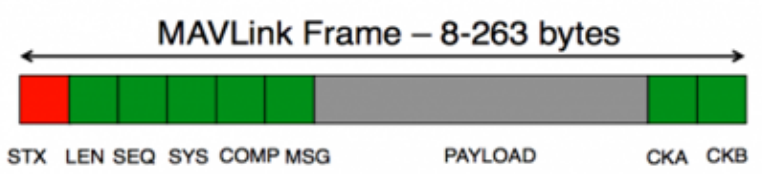


Figura 3.5: Struttura di un pacchetto dati MAVLink

Tabella 3.1: Struttura di un pacchetto dati MAVLink

Indice (byte)	**Contenuto**	**Descrizione**
0	Inizio del frame	Indica l'inizio di un nuovo pacchetto e la versione del protocollo
1	Dimensione del payload	Indica la dimensione del messaggio (payload)
2	Sequenza del pacchetto	Ciascun componente incrementa la propria sequenza di invio, consente di rilevare la perdita di pacchetti
3	ID sistema	Identificativo del sistema che invia il pacchetto, permette di differenziare diversi sistemi nella stessa rete
4	ID componente	Identificativo del componente che invia il messaggio, permette di differenziare diversi componenti in uno stesso sistema (ad esempio l'IMU e l'autopilota)
5	ID messaggio	Identificativo del messaggio, definisce il significato del payload e le modalità per decodificarlo correttamente
6 to (n+6)	Payload	I dati nel messaggio, dipendenti dall'ID del messaggio
(n+7) to (n+8)	Controllo di Ridondanza Ciclico	Check-sum dell'intero pacchetto e rilevazione di errori di trasmissione. Inoltre assicura che mittente e destinatario siano d'accordo sul messaggio inviato

Le comunicazioni tra autopilota ArduPilot e stazione di terra (o companion computer) e le inter-comunicazioni di sistemi a bordo del velivolo avvengono attraverso l'uso del protocollo MAVLink (*Micro Air Vehicle Link*), protocollo tipico per le comunicazioni con piccoli

aeromobili a pilotaggio remoto. MAVLink, all'interno della propria documentazione, definisce una grande quantità di messaggi che possono essere inviati tramite qualsiasi tipologia di trasmissione seriale. Non è garantito che tutti i messaggi vengano recapitati, per questo motivo le stazioni di terra o i companion computer devono controllare lo stato del velivolo per determinare se il comando è stato eseguito o meno.

Un messaggio MAVLink consiste in un flusso di byte codificato dalla stazione di terra ed inviato all'autopilota, o viceversa. Tale messaggio viene emesso in una struttura di dati, un pacchetto, che può avere una dimensione massima di 263 byte e una minima di 8 byte. La struttura del pacchetto è schematizzata in Figura 3.5 mentre è descritta in Tabella 3.1. Il mittente riempie sempre i campi "ID sistema" e "ID componente" in maniera tale che il destinatario sappia sempre da dove arriva il pacchetto. Il campo "ID sistema" ha un unico identificativo per ciascun velivolo (di default "1") o stazione di terra (solitamente "255"). Il campo "ID componente" della stazione di terra è generalmente "1". Tutti i componenti capaci di inviare un pacchetto MAVLink usano il medesimo "ID sistema" del Flight Controller ma usano un differente "ID componente". Il campo "ID messaggio" può avere varie forme a seconda del messaggio che vuole essere inviato (ad esempio l'Heartbeat come "ID messaggio" ha "0"), le alternative sono elencate nella documentazione del protocollo. In Figura 3.6

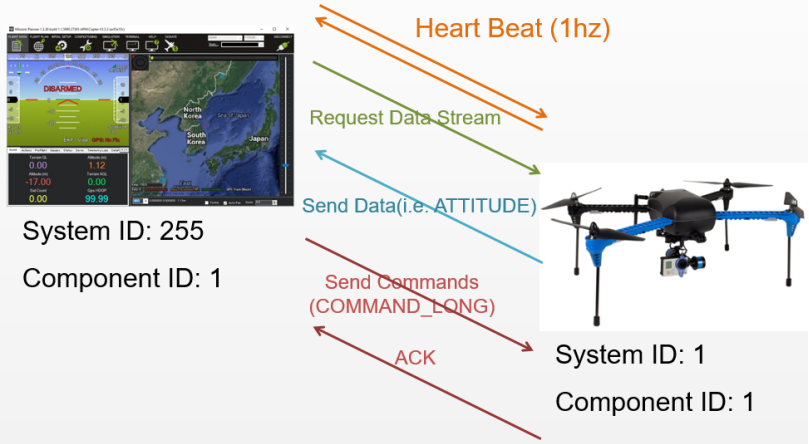


Figura 3.6: Esempio di flussi di messaggi con protocollo MAVLink

è riportato un esempio di flusso di dati tramite protocollo MAVLink. Una volta che la connessione è stabilita ciascun sistema invia il messaggio Heartbeat alla frequenza di 1 Hz. La stazione di terra, o il companion computer, richiede i dati di cui necessita e invia comandi al velivolo.

I comandi MAVLink che il firewall Copter accetta sono molteplici, in questa tesi il messaggio MAVLink necessario è stato il

$$SET\_POSITION\_TARGET\_GLOBAL\_INT$$

, messaggio appartenente alla categoria di comandi per il movimento, la cui descrizione è riportata in Tabella 3.2. Attraverso la funzione

$$goto\_position\_target\_global\_int(LocationGlobal)$$

è possibile inviare tale comando per richiedere al velivolo di volare verso una determinata posizione, da specificare tra parentesi.

Tabella 3.2: Struttura del messaggio SET_POSITION_TARGET_GLOBAL_INT

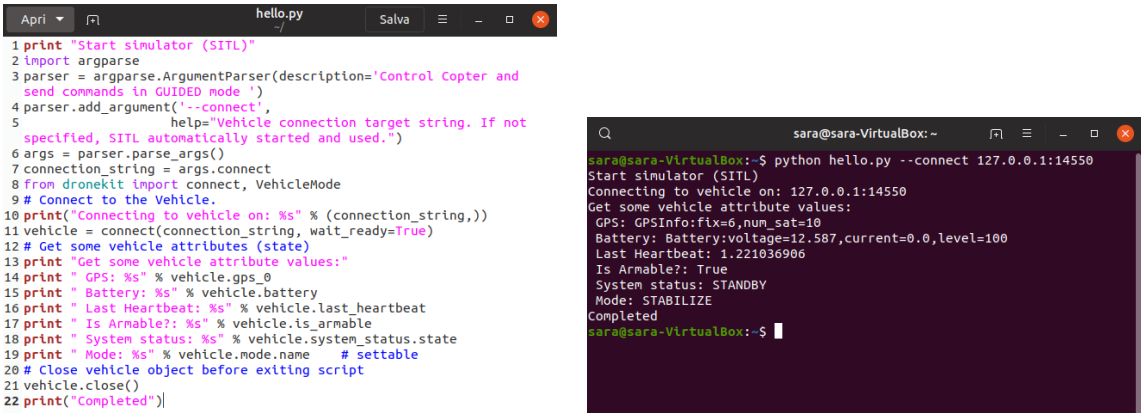
Tipologia di comando	Descrizione
time_boot_ms	Tempo di sistema del mittente in millisecondi dall'avvio
target_system	ID sistema del velivolo
target_component	ID componente del Flight Controller
coordinate_frame	A scelta tra le opzioni: MAV_FRAME_GLOBAL_INT: altitudine in metri slm MAV_FRAME_GLOBAL_RELATIVE_ALT: altitudine in metri rispetto all'altitudine di decollo MAV_FRAME_GLOBAL_RELATIVE_ALT_INT: altitudine in metri rispetto all'altitudine di decollo MAV_FRAME_GLOBAL_RELATIVE_TERRAIN_ALT: altitudine in metri rispetto al terreno MAV_FRAME_GLOBAL_RELATIVE_TERRAIN_ALT_INT: altitudine in metri rispetto al terreno
type_mask	Sequenza di bit indicante quali campi il velivolo deve leggere: Solo posizione Solo velocità Posizione e velocità
lat_int	Latitudine · 1e7
lon_int	Longitudine · 1e7
alt	Altitudine (misurata a seconda del coordinate_frame)
vx	Velocità lungo l'asse X in m/s (positiva verso Nord)
vy	Velocità lungo l'asse Y in m/s (positiva verso Est)
vz	Velocità lungo l'asse Z in m/s (positiva verso il centro della terra)
afx, afy, afz	Le accelerazioni non sono supportate
yaw	Angolo di imbardata in radianti (0 corrisponde al Nord)
yaw_rate	Velocità di imbardata in rad/s

3.4 Companion Computer

I comandi MAVLink, come già anticipato, possono essere inviati tramite stazione di terra oppure tramite Companion Computer, tra cui ad esempio:

- APSync;
- DroneKit;
- FlytOS;
- Maverick;
- ROS;
- Raspberry Pi;
- ODroid;
- Intel Edison;
- Nvidia TX2;
- Nvidia TX1;
- BeaglePilot Project;
- Turnkey Companion Computer Solutions.

In questa tesi come Companion Computer è stato utilizzato DroneKit, che permette di controllare l'autopilota ArduPilot usando il linguaggio di programmazione Python. Per comunicare con l'autopilota su SITL, una volta che questo viene eseguito da terminale come mostrato precedentemente, su un altro terminale viene lanciato un codice scritto in Python contenente i messaggi MAVLink da inviare all'APR. In Figura 3.7 si può osservare il



(a) Codice (b) Esecuzione

Figura 3.7: Esecuzione di un codice in Python di prova per verificare il colloquio con l'autopilota

risultato di una simulazione eseguendo un codice di prova dal nome "Hello.py". Da terminale viene lanciato il codice aggiungendo la stringa di connessione

$$--\mathit{connect}\ 127.0.0.1:14550$$

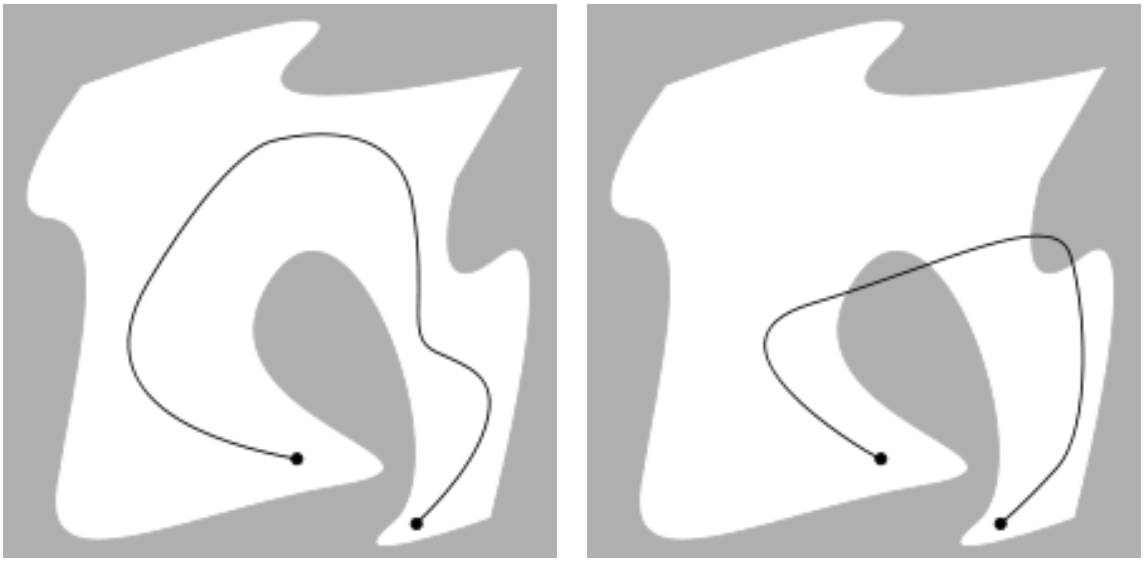
per assicurare la comunicazione con l'autopilota e SITL; una volta partita la comunicazione viene eseguito il codice correttamente, come si può osservare dai parametri stampati a video.

La pianificazione di traiettoria del drone studiata in questa tesi è stata perciò implementata in Python e testata colloquiando con l'autopilota Copter in ambiente di simulazione SITL.

Capitolo 4

Path planning

Il path planning è un termine utilizzato in robotica per trovare una sequenza di configurazioni tali per cui il robot si sposti da una configurazione start ad una configurazione goal evitando gli ostacoli durante il cammino, in Figura 4.1 sono riportati due esempi in cui si nota cosa si intende con percorso valido; la mappa e il movimento del robot possono essere descritti in 2D o in 3D a seconda della casistica in oggetto, nel primo caso si può avere un rover terrestre capace di muoversi in un piano mentre nel secondo caso si può avere un velivolo in grado di spostarsi in uno spazio tridimensionale. Argomento centrale per queste applicazioni è la determinazione del *free space*, ovvero lo spazio libero da ostacoli nel quale il robot può spostarsi. La corretta navigazione fino al target viene realizzata con l'utilizzo di algoritmi di path planning che avendo in input la descrizione della missione producono come output comandi di velocità e rotazione da inviare alle ruote del robot. La pianifi-



(a) Percorso valido (b) Percorso non valido

Figura 4.1: Esempi di applicazione del path planning

cazione di traiettoria ha numerose applicazioni in robotica quali ad esempio l'autonomia, l'automazione, i videogiochi, l'intelligenza artificiale e la chirurgia robotica.

L'obiettivo del drone presentato in questa tesi è quello di navigare in terra ignota per il monitoraggio incendi; dato che la mappa non è nota a priori l'APR è equipaggiato con un sensore LIDAR a basso costo, in particolare si è fatto riferimento al modello Leddar Vu8 con FOV 48° x 0,3° riportato in Figura 4.2, il cui compito consiste nella creazione di una mappatura digitale del terreno. Il comportamento del sensore è stato simulato e per

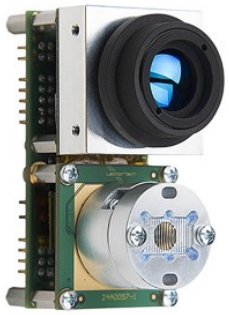


Figura 4.2: Leddar Vu8 48° x 0,3°

una comprensione più specifica del processo di simulazione si rimanda alla tesi "Mappatura digitale del terreno in tempo reale tramite LIDAR per un sistema di monitoraggio incendi." di Luigi Lombardi Vallauri.

4.1 Algoritmi di pathfinding

La pianificazione di un percorso che colleghi efficientemente due punti può essere estesa alla ricerca del percorso più breve nella teoria dei grafi. In matematica con teoria dei grafi si intende lo studio dei grafi, strutture matematiche usate per confrontare a coppie degli oggetti. Con grafo si intende una struttura composta da:

- Oggetti semplici, detti vertici o nodi;
- Collegamenti tra i vertici, che possono essere:
  - Non orientati, collegamenti dotati di direzione ma non di verso. In questo caso il grafo è detto "non orientato";
  - Orientati, collegamenti dotati sia di direzione che di verso. In questo caso il grafo è detto "orientato";
  - Pesati, collegamenti a cui sono associati valori numerici. In questo caso il grafo è detto "pesato".

I grafi vengono raffigurati con cerchi a rappresentazione dei nodi e segmenti che collegano i vari cerchi a rappresentazione dei collegamenti. Nel caso di grafo orientato, un collegamento

può essere percorso esclusivamente nel verso della freccia con cui viene rappresentato, in Figura 4.3 è riportato come esempio un grafo non orientato e pesato. Applicando la teoria

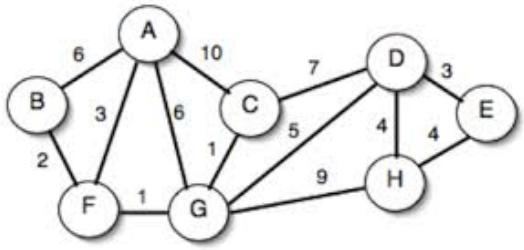


Figura 4.3: Esempio di grafo

dei grafi alla problematica studiata in questa tesi si ha che ciascun nodo rappresenta gli waypoint verso cui e da cui il drone può volare, mentre i collegamenti la strada per raggiungerli; maggiore è il numero di nodi con cui viene discretizzata la mappa, maggiore sarà la raffinatezza della traiettoria.

Un algoritmo di pathfinding ricerca una traiettoria all'interno di un grafo, partendo da un nodo rappresentante la condizione di start e esplorando i nodi adiacenti fino al raggiungimento del goal prestabilito; la traiettoria viene definita con l'intento di ottenere il percorso migliore sulla base di alcuni criteri, ad esempio il più breve, il più economico o il più veloce. Le principali problematiche nel pathfinding sono due:

- Trovare il percorso che collega due nodi in un grafo;
- Trovare il percorso ottimale tra quelli possibili.

Il primo problema può essere risolto da algoritmi anche semplici che esplorano tutte le possibilità all'interno di un grafo partendo dal vertice start. Il secondo problema risulta il più complicato dei due da realizzare e ci sono molteplici soluzioni con cui è stato risolto, le due soluzioni più generali utilizzate sono l'algoritmo di Bellman-Ford e l'algorimo di Dijkstra. Il primo calcola il percorso in un grafo pesato, processa tutti i collegamenti tra i nodi, anche quando questi hanno peso negativo. La ricerca su tutti i nodi di un grafo non è necessaria e risulta essere computazionalmente inefficiente, da questa consapevolezza nasce il secondo algoritmo che in maniera euristica scarta strategicamente i percorsi riducendo i tempi di calcolo, ma necessitando di pesi non negativi dei collegamenti. Visto che in questa tesi l'ambito di utilizzo è una mappa reale, i collegamenti possibili nel *free space* sono tutti positivi e dunque risulta possibile protendere verso la categoria di algoritmi derivati da quello di Dijkstra.

L'algoritmo A* [1] è una variante di quello di Dijkstra. A* assegna a ciascun nodo un valore pari al peso dei collegamenti necessari ad arrivare a quel nodo più la distanza approssimativa tra quel nodo e il goal; tale distanza rappresenta la variabile euristica che l'algoritmo A* per migliorare il comportamento dell'algoritmo di Dijkstra. Questo algoritmo esamina dunque un grafo conosciuto a priori, è quindi funzionale per applicazioni dove la mappa è disponibile e nota quali ad esempio i videogame, ma è stato scartato per risolvere la problematica di questa tesi in quanto l'APR naviga in terra ignota.

Gli algoritmi basati sulla ricerca incrementale e euristica combinano questi due elementi per velocizzare la ricerca di soluzioni in domini che sono parzialmente conosciuti o che cambiano dinamicamente. La ricerca incrementale permette di riutilizzare informazioni ottenute precedentemente per velocizzare la ricerca corrente qualora insorga la necessità di una ripianificazione, non rendendo necessaria la ricompilazione da zero; la ricerca euristica, come dimostrato con il confronto tra A* e l'algoritmo di Dijkstra, risulta essere una scelta efficiente. Questi algoritmi risultano essere perfetti per risoluzioni di problemi dove la traiettoria deve essere ricomputata per via di cambiamenti all'interno del grafo, oppure dove il goal cambia nel tempo, motivi per il quale la categoria risulta essere d'interesse per questa tesi. Nel tempo sono state sviluppati algoritmi che possono essere suddivisi in tre classi:

- La prima classe di algoritmi riavvia l'algoritmo A* nel momento in cui la ricerca corrente varia da quella precedente, ad esempio quando cambiano i pesi dei collegamenti dei nodi. Un esempio è l'algoritmo Fringe-Saving A*.
- La seconda classe di algoritmi aggiorna i valori della funzione euristica durante ogni ricerca per far sì che questi siano sempre aggiornati. Un esempio è l'algoritmo Generalized Adaptive A*.
- La terza classe di algoritmi aggiorna i valori della funzione rappresentante la distanza dal nodo start, mantenendola sempre aggiornata. Esempi di algoritmi appartenenti a questa categoria sono Lifelong Planning A*, D* e D* Lite.

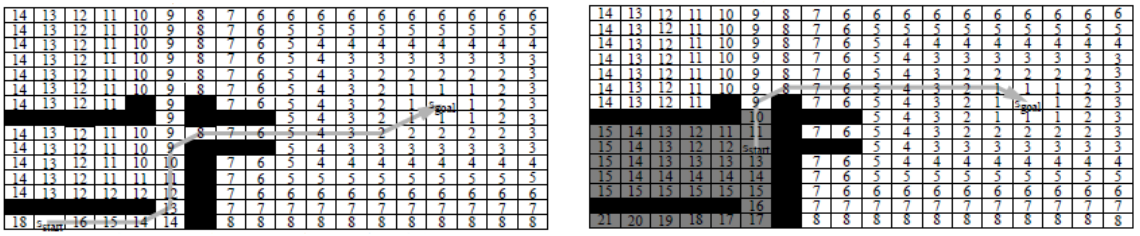
Gli algoritmi di ricerca incrementale e euristica appartenenti a queste tre classi sono differenti dagli altri algoritmi che prevedono la ripianificazione in quanto la qualità della pianificazione non peggiora con il numero di cambiamenti.

Per ridurre il campo di ricerca si è fatto riferimento alla letteratura a riguardo degli algoritmi appena citati. L'algoritmo Fringe-Saving A* [2] è la versione incrementale dell'algoritmo A* e trova ripetutamente, all'interno di un grafo noto, il percorso più breve che colleghi lo start con il goal anche quando i collegamenti tra i nodi crescono o diminuiscono; vista la necessità di un grafo noto non risulta essere appropriato per l'argomento trattato in questa tesi. L'algoritmo Generalized Adaptive A* [3], similarmente a quanto detto per il Fringe-Saving A*, ricalcola il percorso più breve qualora varino i pesi dei collegamenti ed è utilizzato per la ricerca di un percorso verso un target in movimento; in questa tesi il drone deve navigare verso waypoint evitando ostacoli, ma tali waypoint non sono in movimento essendo punti fisici sulla mappa, ne deriva che questo algoritmo non è adatto al caso di studio. Approfondendo la terza classe di algoritmi l'attenzione è stata rivolta verso l'algoritmo D* Lite [4], tale algoritmo è il risultato dell'applicazione del Lifelong Planning A* alla navigazione di un robot in terra ignota; questo utilizzo si incontra perfettamente con la problematica da risolvere nella tesi, motivo per il quale è stato implementato il D* Lite per la pianificazione della traiettoria del drone.

4.2 Implementazione dell'algoritmo di pathfinding

Il D* Lite è un algoritmo costruito a partire dal Lifelong Planning A* e permette al robot di determinare un percorso per collegare il nodo start con il nodo goal in un grafo incognito,

il robot osserva i nodi a lui circostanti e valuta se questi siano attraversabili o meno. L'algoritmo computa il percorso più breve dalla posizione corrente al goal considerando come presupposto che i nodi incogniti siano attraversabili; il robot segue questo percorso finché non raggiunge il target oppure finché non viene riscontrato un nodo non attraversabile, nel qual caso il percorso viene ricomputato. Un esempio del comportamento di questo algoritmo si può osservare in Figura 4.4.



(a) Conoscenza della mappa prima del primo movimento del robot

(b) Percorso non valido

Figura 4.4: Conoscenza della mappa dopo il primo movimento del robot

4.2.1 Lifelong Planning A*

Lifelong Planning A* (LPA*) è la versione incrementale dell'algoritmo A* e viene applicato su grafi noti in cui i pesi dei collegamenti possono crescere o decrescere nel tempo, il suo pseudocodice è riportato in Algoritmo 1. S rappresenta l'insieme dei vertici all'interno del grafo, mentre $Succ(s) \subseteq S$ rappresenta l'insieme dei successori del nodo $s \in S$, ovvero i nodi raggiungibili dal nodo s. Similarmente, $Pred(s) \subseteq S$ rappresenta l'insieme dei predecessori dei vertici $s \in S$, ovvero i nodi dai quali si può raggiungere s. La funzione costo $c(s,\ s')$ rappresenta il costo necessario per muoversi da s a $s' \in Succ(s)$. Conoscendo il grafo e gli attuali pesi dei collegamenti, LPA* computa il percorso più breve che collega il nodo start $s_{start} \in S$ al nodo goal $s_{goal} \in S$. Con la funzione $g(s)$ viene indicata la distanza del percorso più breve che collega s_{start} ad un generico nodo s, mentre con $h(s,\ s_{goal})$ viene indicata la funzione euristica che approssima la distanza del goal dal generico vertice s. La funzione euristica, per ogni $s \in S$ e $s' \in Succ(s)$ con $s \neq s_{goal}$, deve essere non negativa ($h(s_{goal},\ s_{goal}) = 0$) e consistente [5], ovvero rispettare la disuguaglianza angolare ($h(s,\ s_{goal}) \leq c(s,\ s') + h(s',\ s_{goal})$).

Similarmente con quanto avviene in A*, l'algoritmo LPA* mantiene per ogni nodo una stima della sua distanza dallo start, ovvero la funzione $g(s)$; inoltre LPA* utilizza un secondo parametro per descrivere la distanza dal nodo start, ovvero la funzione $rhs(s)$, che risulta più precisa della funzione g. Questo secondo parametro è descritto da:

$$rhs(s) = \begin{cases} 0 & s = s_{start} \\ min_{s' \in Pred(s)}(g(s') + c(s',\ s)) & altrimenti \end{cases}$$

Un vertice s viene detto localmente consistente se il suo $g(s)$ è uguale al suo $rhs(s)$, altrimenti viene detto localmente inconsistente. Se tutti i nodi sono localmente consistenti allora la funzione g in ogni nodo vale quanto la sua distanza dallo start; in questa casistica si può

Algoritmo 1: Lifelong Planning A*

```
   funzione CalcolaKey(s)
1  return [min(g(s), rhs(s)) + h(s, s_goal); min(g(s), rhs(s))];
   funzione Inizializzazione()
2  U = ∅;
3  for ∀ s ∈ S:
4      rhs(s) = g(s) = ∞;
5  rhs(s_start) = 0;
6  U.Inserisci(s_start, CalcolaKey(s_start));
   funzione AggiornaVertice(u)
7  if u ≠ s_start:
8      rhs(u) = min_{s' ∈ Pred(u)}(g(s') + c(s', u));
9  if u ∈ U:
10     U.Rimuovi(u);
11 if g(u) ≠ rhs(u):
12     U.Inserisci(u, CalcolaKey(u));
   funzione ComputaPercorsoPiùBreve()
13 while U.PrimaKey() < CalcolaKey(s_goal) ∨ rhs(s_goal) ≠ g(s_goal):
14     u = U.Estrai();
15     if g(u) > rhs(u):
16         g(u) = rhs(u);
17         for ∀ s ∈ Succ(u):
18             AggiornaVertice(s);
19     else:
20         g(u) = ∞;
21         for ∀ s ∈ Succ(u) ∪ {u}:
22             AggiornaVertice(s);
   funzione Main()
23 Inizializzazione();
24 forever:
25     ComputaPercorsoPiùBreve();
26     Aspetta che cambino i pesi dei collegamenti;
27     for ∀ collegamento orientato tra (u, v) con peso variato:
28         Aggiorna il costo dei collegamenti c(u, v);
29         AggiornaVertice(v);
```

facilmente ricostruire il percorso più breve dallo start a qualsiasi vertice u, andando a ritroso a partire da u verso i suoi predecessori s' seguendo quelli che minimizzano la funzione $rhs(s')$ finché non viene raggiunto lo start. Quando i pesi dei collegamenti variano LPA* non rende i nodi di nuovo localmente consistenti, usa la funzione euristica per focalizzare la ricerca e aggiornare solo i valori di g dei nodi rilevanti nella ricerca del percorso più breve, in questo senso viene mantenuta una coda prioritaria, la quale dunque contiene esattamente i nodi localmente inconsistenti. La priorità di un vertice nella coda prioritaria è definita dalla funzione *key*, un vettore di due componenti: $k(s) = [k_1(s);\ k_2(s)]$, con $k_1 = min(g(s),\ rhs(s) + h(s,\ s_{goal}))$ e $k_2(s) = min(g(s),\ rhs(s))$. Le *key* sono ordinate secondo un ordine lessicografico, in particolare $k(s) \leq k(s')$ se $k_1(s) < k_1(s')$ oppure $k_1(s) = k_1(s')$ e $k_2(s) \leq k_s(s')$. LPA* estrae dalla coda prioritaria sempre il vertice con *key* minore.

Facendo riferimento allo pseudocodice riportato in Algoritmo 1, si definiscono le funzioni che fanno riferimento alla coda prioritaria:

- U.Estrai() restituisce il nodo con *key* minore all'interno della coda prioritaria U e lo

elimina dalla coda;

- U.PrimaKey() restituisce la più piccola *key* all'interno della coda prioritaria U. Se la coda è vuota restituisce $[\infty;\ \infty]$;
- U.Inserisci($s,\ k$) inserisce il nodo s nella coda prioritaria U con *key* k;
- U.Rimuovi(s) rimuove il nodo s dalla coda prioritaria U.

La funzione $Main()$ dell'algoritmo per prima richiama la funzione $Inizializzazione()$, la quale imposta i valori di g e rhs di tutti i vertici ad esclusione del nodo start che risulta essere l'unico localmente inconsistente, condizione necessaria a non avere la coda prioritaria vuota. L'inizializzazione garantisce che il primo ciclo della funzione $ComputaPercorsoPiùBreve()$ porti allo stesso risultato fornito dall'algoritmo A*. In seguito l'algoritmo aspetta eventuali cambiamenti nei pesi dei collegamenti e se qualcosa cambia aggiorna i valori di rhs e *key* dei nodi intaccati da tali variazioni e calcola nuovamente il percorso più breve. Un vertice s localmente inconsistente viene detto sovraconsistente se $g(s) > rhs(s)$; quando viene incontrato un nodo che rispetta questa condizione, il valore di g viene abbassato al valore di rhs, il che lo rende localmente consistente. Un vertice s localmente inconsistente viene invece detto sottoconsistente se $g(s) < rhs(s)$; con un nodo appartenente a questa casistica l'algoritmo si comporta diversamente dal caso precedente, il valore di g viene posto a infinito $g(s) = \infty$ rendendo così il nodo o localmente consistente o sovraconsistente. Nel caso di nodo sovraconsistente vengono influenzati i suoi successori, mentre per il caso di nodo sottoconsistente vengono influenzati il nodo stesso e i suoi successori; stabilito il gruppo vengono dunque aggiornati i valori di rhs e aggiornata di conseguenza la coda prioritaria. L'algoritmo procede come descritto finché il nodo goal non diventa localmente consistente, qualora $g(s_{goal}) = \infty$ significherebbe che non esiste un percorso che possa collegare start e goal. Diversamente, il percorso più breve viene costruito a partire dal goal traslando verso i predecessori s' che minimizzano $g(s') + c(s',\ s)$ finché non viene raggiunto lo start.

4.2.2 D* Lite

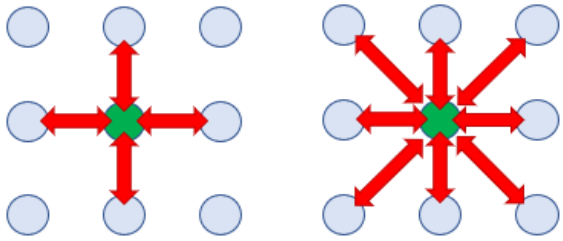
L'algoritmo LPA* appena descritto è stato utilizzato per costruire l'algoritmo D* Lite, il cui pseudocodice è riportato in Algoritmo 2. D* Lite determina ripetutamente il percorso più breve, senza fare stime o previsioni sui cambiamenti nel peso dei collegamenti, tra il nodo corrente a cui si trova il robot e il nodo goal, mentre il robot stesso si sta muovendo; per tale ragione questo algoritmo viene utilizzato per risolvere la navigazione verso una posizione goal in un grafo ignoto. Il terreno può essere modellato o con un grafo *four-connected* oppure con un grafo *eight-connected*, i cui schemi sono riportati in Figura 4.5. Nel caso *four-connected* il robot ha la possibilità di movimento solo verso i quattro punti cardinali a lui circostanti, nel caso *eight-connected* si aggiunge la possibilità di movimento anche lungo le quattro diagonali; il peso dei collegamenti verso i punti cardinali viene posto pari ad 1, mentre il peso dei collegamenti diagonali pari a $\sqrt{2}$, per mantenere una coerenza realistica nell'elaborazione del percorso. In questa tesi è stato utilizzato il caso *four-connected*. Ogni vertice del grafo rappresenta un appezzamento di terra pari alla discretizzazione che viene fatta della mappa di navigazione; qualora venga rilevato un ostacolo il nodo corrispondente viene considerato inattraversabile, senza informazioni a riguardo dell'estensione dell'ostacolo

Algoritmo 2: D* Lite

funzione CalcolaKey(s)
return $[min(g(s),\ rhs(s)) + h(s_{start},\ s) + k_m;\ min(g(s);\ rhs(s))]$;
funzione Inizializzazione()
$U = \emptyset$;
$k_m = 0$;
for $\forall\ s\ \in\ S$**:**
    $rhs(s) = g(s) = \infty$;
$rhs(s_{goal}) = 0$;
U.Inserisci($s_{goal},\ CalcolaKey(s_{goal})$);
funzione AggiornaVertice(u)
if $u \neq s_{goal}$**:**
    $rhs(u) = min_{s'\ \in\ Succ(u)}(c(u,\ s') + g(s'))$;
if $u\ \in\ U$**:**
    U.Rimuovi(u);
if $g(u) \neq rhs(u)$**:**
    U.Inserisci($u, CalcolaKey(u)$);
funzione ComputaPercorsoPiùBreve()
while $U.PrimaKey() < CalcolaKey(s_{start})\ \vee\ rhs(s_{start}) \neq g(s_{start})$**:**
    $k_{old} = U.PrimaKey()$;
    $u = U.Estrai()$;
    if $k_{old} < CalcolaKey(u)$**:**
        U.Inserisci($u,\ CalcolaKey(u)$);
    elif $g(u) > rhs(u)$**:**
        $g(u) = rhs(u)$;
        for $\forall\ s\ \in\ Pred(u)$**:**
            AggiornaVertice(s);
    else:
        $g(u) = \infty$;
        for $\forall\ s\ \in\ Pred(u)\ \cup\ \{u\}$**:**
            AggiornaVertice(s);
funzione Main()
$s_{last} = s_{start}$;
Inizializzazione();
ComputaPercorsoPiùBreve();
while $s_{start} \neq s_{goal}$**:**
    $s_{start} = arg\ min_{s'\ \in\ Succ(s_{start})}(c(s_{start},\ s') + g(s'))$;
    Vai verso s_{start};
    Controlla il grafo per cambiamenti nei pesi dei collegamenti; **if** *qualche collegamento è cambiato*:
        $k_m = k_m + h(s_{last},\ s_start)$;
        $s_{last} = s_{start}$;
        for $\forall$ *collegamento orientato tra* $(u,\ v)$ *con peso variato*:
            Aggiorna il costo dei collegamenti $c(u,\ v)$;
            AggiornaVertice(u);
        ComputaPercorsoPiùBreve();

uno spostamento diagonale del drone, avendo questo un volume finito e non puntuale, potrebbe non garantire l'evitamento di quell'ostacolo. Come per l'algoritmo precedente, anche nel D* Lite quando un nodo è considerato inattraversabile vengono posti a infinito i pesi dei collegamenti da e verso quel vertice. LPA* effettua la ricerca del percorso dallo start verso il goal, dunque la funzione g rappresenta la distanza di ciascun nodo dallo start;

D* Lite cerca invece dal goal verso lo start, di conseguenza la funzione g rappresenta le distanze dei vari vertici dal goal.



(a) Grafo *four-connected* (b) Grafo *eight-connected*

Figura 4.5: Esempi di connessione tra i nodi all'interno di un grafo

Per risolvere il problema della navigazione in terra ignota, la funzione $Main()$ deve fare in modo di spostare il robot lungo il percorso computato da $ComputaPercorsoPiùBreve()$. Quando cambiano i pesi dei collegamenti variano anche le priorità dei vari vertici all'interno della coda prioritaria, riordinare ogni volta la coda risulta computazionalmente dispendioso, ma non aggiornarla porterebbe ad avere la priorità di un vertice differente dalla sua *key* e questo non può mai verificarsi. D* Lite, per evitare di riordinare la coda prioritaria, utilizza un metodo derivante dall'algoritmo D* [6], ovvero priorità che presentano un limite inferiore rispetto alle priorità utilizzate dal LPA*. La funzione euristica h deve essere non negativa e soddisfare $h(s,\ s') \leq c(s,\ s')$ e $h(s,\ s'') \leq h(s,\ s')+h(s',\ s'')$ per tutti i vertici $s,\ s',\ s'' \in S$, con $c(s,\ s')$ che rappresenta il costo della distanza minima tra il nodo s e il nodo s'. Quando il robot parte dal nodo s_{start} inizia a percorrere le celle finché, ad un generico nodo s non viene riscontrata una variazione nel costo dei collegamenti all'interno del grafo. Nel momento in cui avviene il cambiamento bisogna dunque aggiornare la funzione euristica e poiché nel D* Lite il calcolo delle distanze viene fatto rispetto al goal, tale funzione andrà decrescendo man mano che il robot si allontana dallo start avvicinandosi al goal. Questa decrescita non può essere inferiore non può essere superiore a $h_\delta = h(s_{start},\ s)$, ovvero la distanza tra lo start e la posizione generica a cui si è verificata la variazione nei collegamenti. Per mantenere il limite inferiore nella coda prioritaria D* Lite dovrebbe dunque sottrarre il valore h_δ a tutti i vertici all'interno della coda, ma dato che la sottrazione sarebbe la stessa per tutti, comunque non cambierebbe l'ordine all'interno della coda. Dal momento successivo alla variazione nel grafo, quando una nuova priorità viene elaborata, le prime componenti risulterebbero più piccole di un termine h_δ rispetto alle altre all'interno della coda, motivo per il quale nell'algoritmo viene sommato alle prime componenti ogni qualvolta si presenti una variazione nei collegamenti (sommare h_δ alle nuove priorità risulta essere equivalente a sottrarlo a tutte le altre). Per tenere conto degli incrementi necessari da fare alla funzione euristica è stato introdotto il parametro *key modifier* k_m che viene aggiornato ad ogni modifica che avviene nel grafo. In questo modo la coda prioritaria non deve essere costantemente riordinata e viene garantito il limite inferiore rispetto alle priorità del LPA*.

4.2.3 Adattamento dell'algoritmo alla missione dell'APR

L'algoritmo presentato nel precedente sotto-paragrafo è inteso per un utilizzo in campo 2D, per questo motivo anche per l'APR è stato inizialmente implementato in 2D e, una volta appurato il suo corretto funzionamento, si è proceduto con l'espansione in campo 3D.

Il primo passaggio ha previsto il valutare la zona entro cui il drone possa navigare, questa è stata inizializzata come un'area quadrata di 150m x 150m; per tale mappa si è fatta una discretizzazione dividendola in quadrati più piccoli di 1,5m x 1,5m. Su questa mappa discretizzata è stato dunque costruito il grafo, ogni area è rappresentata da un nodo posto al suo centro, ne deriva che il grafo sia matriciale e di dimensione 100 x 100. Per simulare il comportamento del sensore LIDAR a bordo è stato necessario l'utilizzo di un modello digitale di elevazione (DEM, dall'inglese *Digital Elevation Model*), da questo modello si è creata una mesh e tramite l'implementazione di un codice di *ray tracing* è stato possibile simulare il comportamento dei raggi del sensore e ottenere una simulazione di misurazioni del terreno. Il sensore ad ogni impulso produce un fascio composto da 8 raggi, questi raggi incontreranno il terreno in 8 punti e dunque ogni misurazione è composta da un ottetto di dati rappresentanti l'altitudine del terreno. Come riferimento di latitudine e longitudine è stato trovato un DEM disponibile online con discretizzazione 5m x 6,57m; tale discretizzazione risulta paragonabile alla precisione del sensore, ad un'altezza di 30m, dato il FOV di 48°x 0,3°, il fascio di 8 raggi spazia su circa 40m, dunque con una distanza di circa 5m tra ciascun dato nell'ottetto. Valutando la frequenza di acquisizione del sensore si è stabilita una velocità per il drone tale per cui le acquisizioni fossero abbastanza ravvicinate tra loro per avere una ricreazione accurata della mappa; tale velocità di navigazione è stata impostata ad 1 m/s. Per testare il comportamento dell'algoritmo di pathfinding si è pensato di utilizzare un ambiente più realistico, motivo per il quale è stato creato un DEM popolato di ostacoli con una discretizzazione pari a quella del grafo. Per la latitudine e la longitudine di riferimento sono state mantenuti i valori della mappa originale, vista la simulazione in SITL è stato considerata un'area pianeggiante di modo che l'altimetro non riscontrasse ostacoli e interrompesse così la simulazione. Il punto di riferimento per calcolare la posizione del drone all'interno del grafo è il punto più a Nord-Ovest dell'area quadrata di navigazione, evidenziato nello schema riportato in Figura 4.6 (per semplicità grafica si è riportato un grafo 10 x 10 invece di uno 100 x 100); tale punto di riferimento ha le seguenti coordinate:

$$Lat_{rif} = 40,82233^{\circ}$$

$$Lon_{rif} = 14,70328^{\circ}$$

Per iniziare la navigazione vengono impostati i punti, a partire dallo start, che deve raggiungere il drone, questi *waypoint* a coppie diventano start e goal, di conseguenza viene computato il percorso più breve a collegarli tramite l'algoritmo di *pathfinding*. Tali coordinate vengono ricondotte a nodi nel grafo tramite le seguenti formule:

$$coo_{x_{start}} = int(round(deg2rad(Lon_{start} - Lon_{rif}) \cdot R_{Terra}/\delta_x))$$

$$coo_{y_{start}} = int(round(deg2rad(Lat_{rif} - Lat_{start}) \cdot R_{Terra}/\delta_y))$$

$$coo_{x_{goal}} = int(round(deg2rad(Lon_{goal} - Lon_{rif}) \cdot R_{Terra}/\delta_x))$$

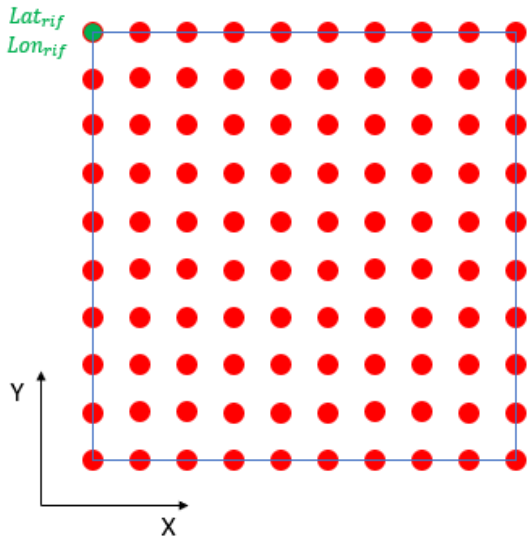


Figura 4.6: Schema d'esempio rappresentante il grafo 2D utilizzato per il drone

$$coo_{y_{goal}} = int(round(deg2rad(Lat_{rif} - Lat_{goal}) \cdot R_{Terra}/\delta_y))$$

Dove la funzione $deg2rad$ permette la conversione da gradi a radianti, $R_{Terra} = 6371000m$ è il raggio della Terra, δ_x e δ_y rappresentano le discretizzazioni lungo gli assi X e Y e come già detto sono pari a 1,5m, la funzione $round$ restituisce l'arrotondamento del risultato e la funzione int restituisce un numero intero.

Dopo queste premesse è stato implementato l'algoritmo D* Lite e si è verificato il corretto funzionamento su un piano 2D in quota. L'algoritmo computa il percorso più breve che collega start e goal, il velivolo deve portarsi verso ciascuna cella componente tale percorso; la cella viene convertita in coordinate spaziali con le seguenti formule:

$$Lat_{wp} = Lat_{rif} - \ rad2deg(float(coo_y \cdot \delta_y)/R_{Terra})$$

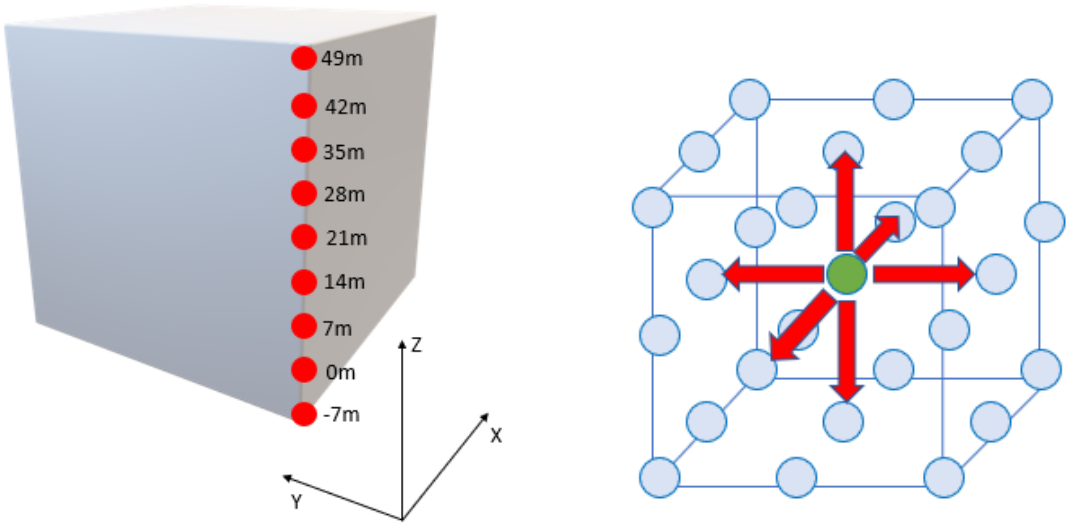
$$Lon_{wp} = rad2deg(float(coo_x \cdot \delta_x)/R_{Terra}) + Lon_{rif}$$

Dove la funzione $rad2deg$ converte i radianti in gradi, la funzione $float$ rende il numero in formato virgola mobile, coo_x e coo_y sono le coordinate della cella da raggiungere all'interno del percorso computato. Una volta ottenute latitudine e longitudine del waypoint risulta immediato inviare il comando di spostamento al drone tramite il protocollo MAVLink; quando il velivolo si trova nel nuovo punto il ciclo ricomincia finché non viene raggiunto il goal stabilito. Dopo aver verificato il funzionamento del codice in ambiente 2D, si è introdotta la lettura dei dati prodotti dal sensore per l'aggiornamento della mappa e dunque la navigazione in campo 3D.

4.2.4 Espansione 3D

Per ampliare il campo di navigazione in ambiente 3D è stato necessario variare la tipologia di grafo e i collegamenti tra i nodi. Il grafo è stato reso tale da occupare tutto il volume di navigazione, l'asse Z è stato discretizzato in 9 vertici distanti tra loro 7m; tale δ_z è stato

selezionato per garantire al sensore di valutare in tempo gli ostacoli e gestire la distanza di sicurezza. I 9 livelli sono stati disposti secondo lo schema riportato in Figura 4.7 dove il livello a 0m rappresenta l'altitudine del terreno corrispondente alla posizione di decollo; ad ogni livello lungo l'asse Z si trova un grafo 2D nel piano XY uguale a quello descritto precedentemente. I livelli possono essere posti differentemente (ad esempio per navigare sopra un terreno in pendenza) o essere in numero differente a seconda delle applicazioni, cambieranno le formule con cui viene convertita l'altitudine in coordinata nel grafo e viceversa. I collegamenti tra i vertici diventano 6 come mostrato in Figura 4.7, all'interno del piano XY il grafo rimane *four-connected* come già approfondito, a questi 4 collegamenti se ne aggiungono altri 2 paralleli all'asse Z necessari a collegare i piani tra loro.



(a) Grafo 3D (b) Collegamenti 3D

Figura 4.7: Schemi del grafo 3D e dei collegamenti tra i nodi al suo interno

Durante la navigazione si è utilizzata la funzione *callback* dell'autopilota ArduCopter, tramite la quale vengono stampati su file di testo i valori di assetto (angoli di rollio, beccheggio e imbardata) e i valori di posizione (latitudine, longitudine e altitudine) ogni qualvolta questi varino durante la navigazione; questi file vengono letti dal sensore LIDAR affinché esso sappia dove si trova il drone e con che assetto naviga per direzionare i raggi in maniera coerente. Il sensore è montato sul drone con un angolo di 45° verso la direzione di avanzamento, per predire quanto più possibile la mappa e avere tempo di intervenire sulla rotta prima che il velivolo incontri ostacoli. Gli ottetti misurati dal sensore vengono dunque stampati su un file che a sua volta viene aperto dal codice di navigazione dopo ogni spostamento dell'APR all'interno del grafo. Ciascun valore in un ottetto è composto da tre valori, latitudine e longitudine a cui il raggio ha incontrato il terreno e altitudine del terreno a quella posizione. Quando avviene la lettura del file, le misurazioni vengono convertite per ottenere le coordinate all'interno del grafo:

$$coo_{x_{ost}} = int(round(deg2rad(Lon_{ost} - Lon_{rif}) \cdot R_{Terra}/\delta_x))$$

$$coo_{y_{ost}} = int(round(deg2rad(Lat_{rif} - Lat_{ost}) \cdot R_{Terra}/\delta_y))$$

$$\begin{cases} coo_{z_{ost}} = int(round((Alt_{ost} - Alt_{start})/\delta_z) + 2) & Se\ (Alt_{ost} - Alt_{start}) > -3,5 \\ coo_{z_{ost}} = int(2 + round((Alt_{ost} - Alt_{start})/\delta_z)) & Se\ (Alt_{ost} - Alt_{start}) \leq -3,5 \end{cases}$$

Dove Alt_{ost} è la misurazione effettuata dal sensore e Alt_{start} è l'altitudine del terreno alla posizione di decollo. La coordinata z misurata dal sensore traccia il livello del terreno e i potenziali ostacoli incontrabili dal drone, di conseguenza il codice pone inattraversabile quella cella e tutte quelle sottostanti. In questo modo viene popolato il grafo di ostacoli e viene ricostruita la mappa di navigazione, il drone ha così modo di evitare gli impedimenti e ricomputare il percorso più breve per raggiungere il goal impostato. L'algoritmo D* Lite, opportunamente modificato per gestire un grafo 3D, restituisce dunque una cella nel grafo da raggiungere dove le coordinate lungo gli assi X e Y vengono convertite in longitudine e latitudine con le formule già approfondite, la coordinata lungo l'asse Z viene convertita in altitudine con:

$$Alt_{wp} = float((coo_z - 2) \cdot \delta_z) + Alt_{start}$$

In questo modo viene individuato un punto nello spazio verso cui il drone deve navigare e tale comando viene inviato con un messaggio coerente con il protocollo MAVLink.

Il ciclo si sviluppa come approfondito finché il velivolo non raggiunge l'ultimo goal impostato all'inizio della missione, terminata la necessità di computare un percorso l'algoritmo cessa di essere richiamato, il drone passa in modalità atterraggio e la simulazione viene interrotta. I vari codici implementati in questa tesi possono essere trovati in appendice in questa tesi.

Capitolo 5

Test e validazione

Per validare il funzionamento dell'algoritmo implementato si sono resi necessari dei test che lo mettessero alla prova, in questo modo si è potuto sperimentare per ricercare eventualmente dei limiti nella sua applicazione e per saperne prevedere il comportamento al variare dell'ambiente di navigazione. Come approfondito nel capitolo precedente, l'area è

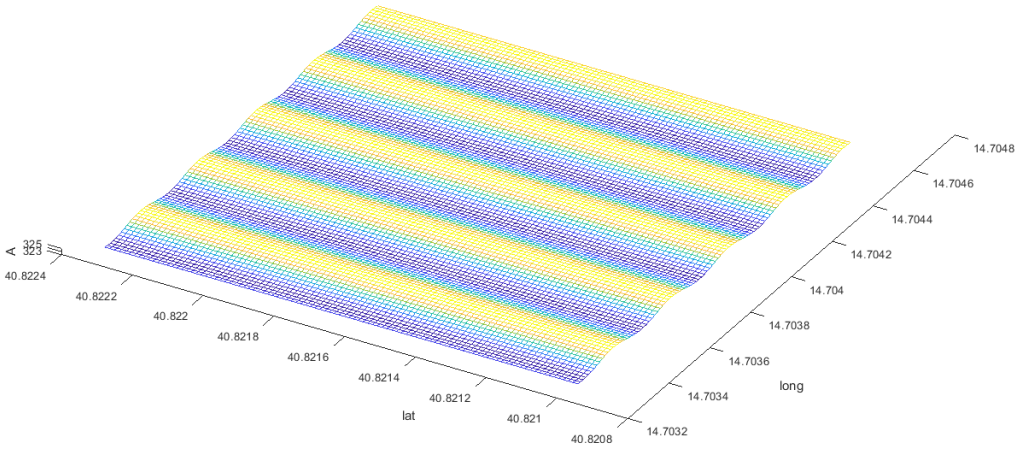


Figura 5.1: Superficie del modello digitale di elevazione

rappresentata con una mappa suddivisa in 100 x 100 celle della dimensione di 1,5m x 1,5m ciascuna. Per rappresentare il terreno si è usata la funzione:

$$Cella(i,\ j) = sin(Lon(j)/5) + Alt_{start}$$

per simulare un terreno non esattamente pianeggiante; tale superficie è riportata in Figura 5.1. Il modello di superficie è stato popolato di ostacoli per effettuare differenti simulazioni che verranno approfondita nei seguenti paragrafi.

I test sono stati effettuati in ambiente Linux utilizzando 3 terminali contemporaneamente, uno per la simulazione dell'autopilota in ambiente SITL, uno a rappresentazione del

drone per lanciare il codice implementato in questa tesi, e l'ultimo per simulare il sensore, utilizzando il codice scritto e approfondito dal collega Luigi Lombardi Vallauri nella sua tesi di laurea.

5.1 Ambiente popolato da ostacoli

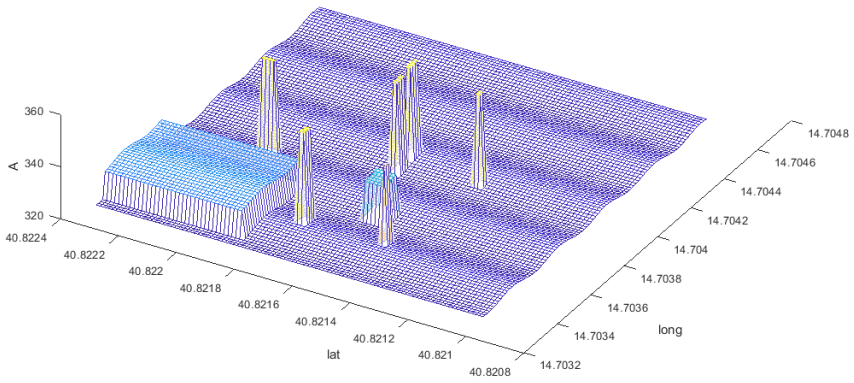


Figura 5.2: DEM impostato per la prima simulazione

Come prima simulazione si è creato il DEM rappresentato in Figura 5.2; con questo modello si è voluto rappresentare un ambiente frastagliato sia con ostacoli poco elevati per verificare che non venissero riconosciuti come ostacolo, sia con ostacoli che non occupassero un'area grande in quanto tali impedimenti risultano essere i più insidiosi durante una navigazione in terra ignota. Le celle modificate per realizzare gli ostacoli sono riportate in Tabella 5.1.

Tabella 5.1: Disposizione celle ostacolo nella mappa durante la prima simulazione

Latitudine [celle]	**Longitudine [celle]**	**ΔAltitudine [m]**
35	$16 \div 18$	30
$40 \div 45$	$25 \div 30$	15
$55 \div 56$	$15 \div 18$	35
$77 \div 80$	$40 \div 41$	35
$49 \div 50$	$46 \div 49$ $52 \div 55$	35
$30 \div 31$	$50 \div 51$	35
$65 \div 100$	$5 \div 30$	10

Questa simulazione è consistita in una navigazione ad una quota più elevata per mappare il terreno sottostante e tale rotta è stata ripercorsa ad una quota inferiore per verificare

che l'algoritmo ricomputasse il percorso per evitare gli ostacoli individuati. L'insieme di waypoint impostati a comporre la rotta sono riportati in Tabella 5.2, dove l'altitudine è riportata in metri dal terreno.

Tabella 5.2: Waypoint che compongono la rotta della prima simulazione

Latitudine [°]	**Longitudine** [°]	**Altitudine** [m]
40, 8210	14, 7035	42
40, 8220	14, 7035	42
40, 8220	14, 7039	42
40, 8213	14, 7039	42
40, 8213	14, 7039	28
40, 8213	14, 7035	28
40, 8220	14, 7035	28
40, 8220	14, 7039	28
40, 8213	14, 7039	28

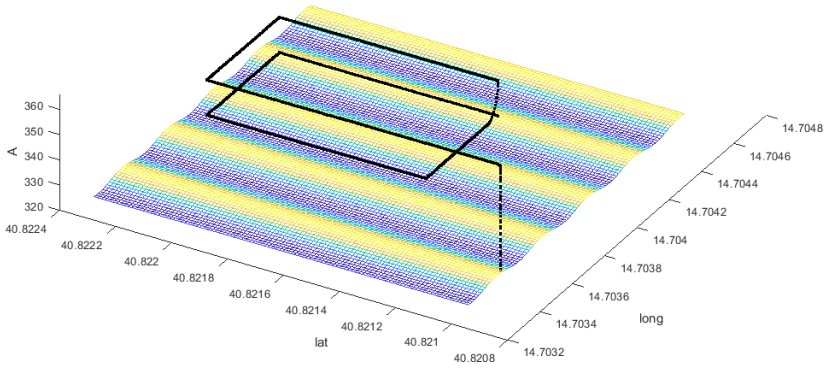


Figura 5.3: Rotta nominale della prima simulazione in ambiente privo di ostacoli

In Figura 5.3 si può osservare la rotta nominale effettuata dal drone in un ambiente privo di ostacoli, tale rotta collega con delle linee rette gli waypoint impostati.

In Figura 5.4 si può osservare il risultato ottenuto dopo la prima simulazione, i punti rossi rappresentano le misurazioni effettuate dal sensore simulato. In questo test si può notare che la traiettoria percorsa alla quota più alta risulta coincidente con la traiettoria nominale, gli ostacoli non vengono riconosciuti come pericolo e il drone prosegue la navigazione indisturbato. A quota più bassa si nota che l'algoritmo esegue il suo compito efficientemente, tutti gli ostacoli sono stati evitati con successo. La particolarità principale di questo test è stato il posizionamento di due ostacoli affiancati con uno spiraglio libero

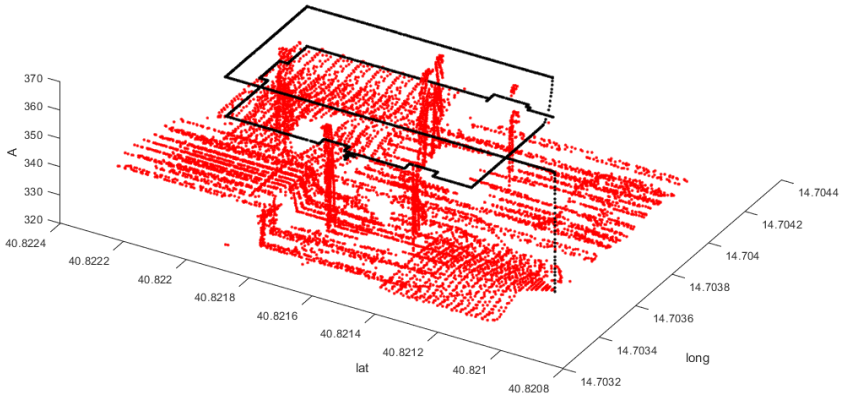


Figura 5.4: Rotta effettiva della prima simulazione

al centro, l'obiettivo era verificare se il drone sarebbe riuscito a valutare di poter passare all'interno di questo spiraglio, ipotesi che si è positivamente realizzata.

Le misurazioni effettuate dal sensore hanno permesso di visualizzare la mesh ricreata a seguito della navigazione, visibile in Figura 5.5. Come si vede nell'immagine la mesh non è

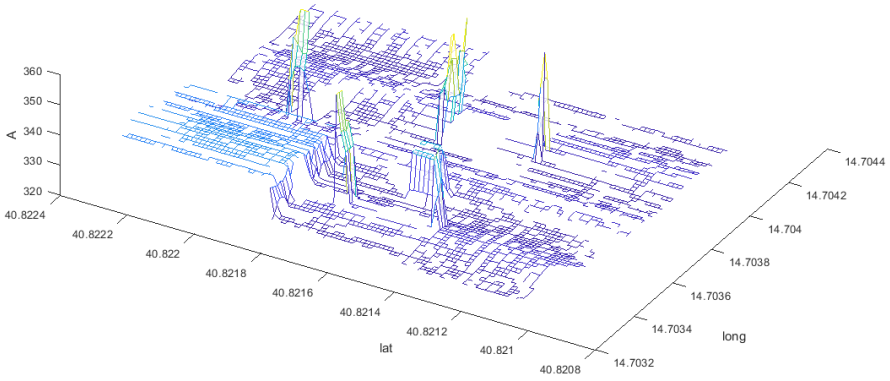


Figura 5.5: Ricostruzione del terreno a seguito della prima simulazione

esattamente identica al DEM impostato, ma ne ricalca similmente le fattezze; in particolare rappresenta sufficientemente gli ostacoli per quanto è necessario al drone ad evitarli.

5.2 Rotta a zig-zag

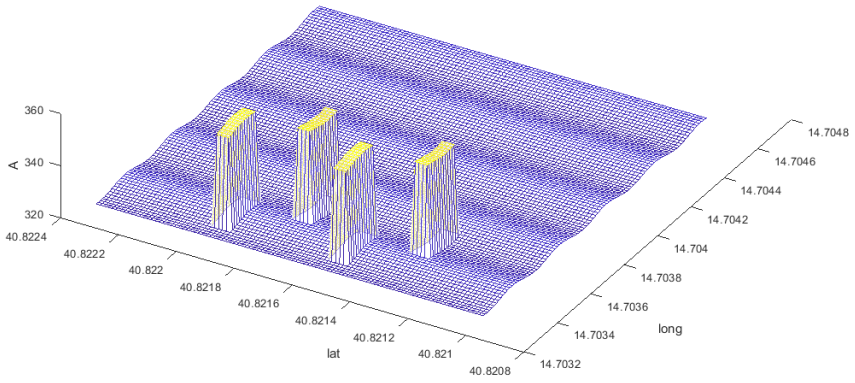


Figura 5.6: DEM impostato per la seconda simulazione

Come seconda simulazione si è creato il DEM rappresentato in Figura 5.6; con questo modello si è voluto rappresentare un insieme di ostacoli ravvicinati per forzare una navigazione a zig-zag da parte del drone. L'intento di questo test è stato quello di valutare se l'APR sarebbe riuscito a navigare tra i palazzi invece di aggirarli dall'esterno, rotta meno efficiente in quanto più lunga. Le celle modificate per realizzare gli ostacoli sono riportate in Tabella 5.3.

Tabella 5.3: Disposizione celle ostacolo nella mappa durante la seconda simulazione

Latitudine [celle]	**Longitudine [celle]**	**ΔAltitudine [m]**
$24 \div 27$	$16 \div 28$	35
$39 \div 42$	$6 \div 18$	35
$54 \div 57$	$16 \div 28$	35
$69 \div 72$	$6 \div 18$	35

Anche per questa simulazione si è impostata una navigazione ad una quota più elevata per mappare il terreno sottostante e un'altra navigazione ad una quota inferiore per testare il comportamento dell'algoritmo. L'insieme di waypoint impostati a comporre la rotta sono riportati in Tabella 5.4, dove l'altitudine è riportata in metri dal terreno.

In Figura 5.7 si può osservare la rotta nominale effettuata dal drone in un ambiente privo di ostacoli, tale rotta collega con delle linee rette gli waypoint impostati.

In Figura 5.8 si può osservare il risultato ottenuto dopo la seconda simulazione, i punti rossi anche in questo caso rappresentano le misurazioni effettuate dal sensore simulato. Similarmente al caso precedente, anche in questo test la traiettoria percorsa alla quota più alta risulta coincidente con la traiettoria nominale, a quota più bassa si nota che il

Tabella 5.4: Waypoint che compongono la rotta della seconda simulazione

Latitudine [°]	**Longitudine** [°]	**Altitudine** [m]
40, 8210	14, 7035	42
40, 8220	14, 7035	42
40, 8220	14, 7035	28
40, 8210	14, 7035	28

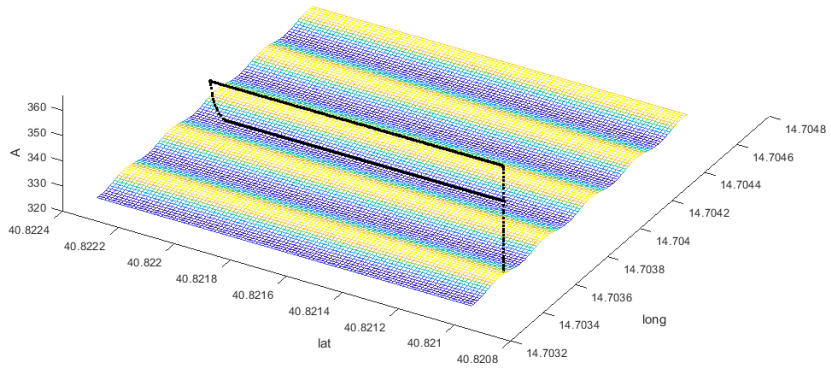


Figura 5.7: Rotta nominale della seconda simulazione in ambiente privo di ostacoli

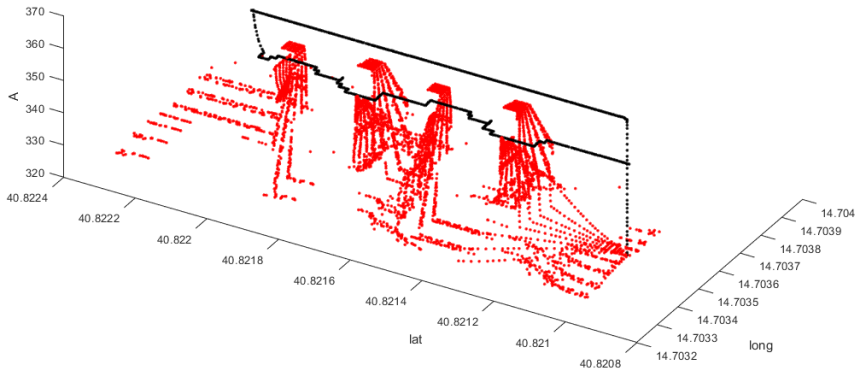


Figura 5.8: Rotta effettiva della seconda simulazione

drone porta a termine la missione come auspicato per questo test. L'ipotesi di rotta a zig-zag tra gli ostacoli si è realizzata, anche se con delle piccole incertezze tra un ostacolo e l'altro (probabilmente dovute all'ottenimento da parte del sensore di nuove misurazioni che hanno ampliato la conoscenza dell'estensione in longitudine degli ostacoli); questo test ha confermato la validità dell'algoritmo nel determinare la rotta più breve anche in presenza di ostacoli ravvicinati.

Le misurazioni effettuate dal sensore hanno permesso di visualizzare la mesh ricreata a seguito della navigazione, visibile in Figura 5.9. Come si vede nell'immagine la mesh non è

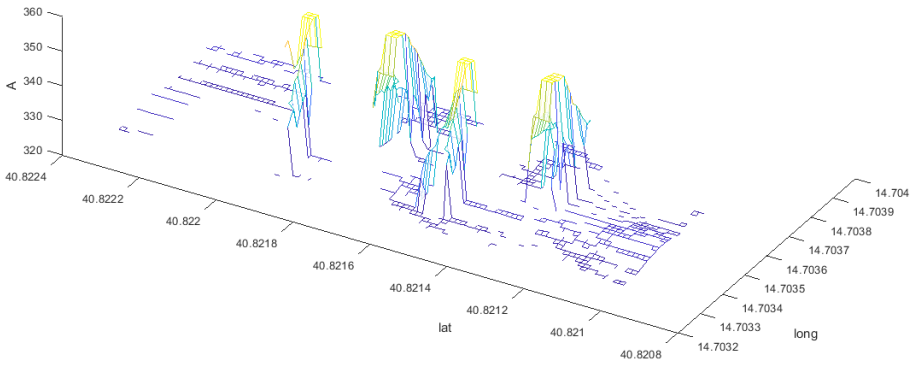


Figura 5.9: Ricostruzione del terreno a seguito della seconda simulazione

esattamente identica al DEM impostato, ma dà l'idea della sua composizione e permette al drone di evitare gli ostacoli.

5.3 Muricciolo

Come terza simulazione si è creato il DEM rappresentato in Figura 5.10; con questo modello si è voluto rappresentare un muricciolo per verificare il corretto funzionamento dell'algoritmo in un campo 3D. Le celle modificate per realizzare gli ostacoli sono riportate in Tabella 5.5.

Tabella 5.5: Disposizione celle ostacolo nella mappa durante la terza simulazione

Latitudine [celle]	**Longitudine [celle]**	**ΔAltitudine [m]**
$20 \div 22$	$1 \div 100$	30

In questa simulazione era previsto che il velivolo navigasse ad una quota più elevata per registrare le informazioni a riguardo del terreno sottostante e ad una quota più bassa, in particolare inferiore all'altezza del muricciolo, per verificare che il drone capisse di doverlo

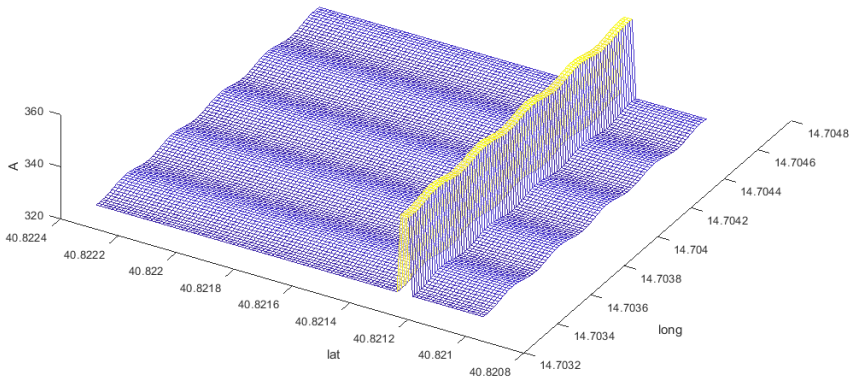


Figura 5.10: DEM impostato per la terza simulazione

scavalcare muovendosi lungo l'asse Z invece di provare ad aggirarlo nel piano di volo senza successo, dato che il muro si estende per tutta la lunghezza della mappa. L'insieme di waypoint impostati a comporre la rotta sono riportati in Tabella 5.6, dove l'altitudine è riportata in metri dal terreno.

Tabella 5.6: Waypoint che compongono la rotta della terza simulazione

Latitudine [°]	**Longitudine** [°]	**Altitudine** [m]
40, 8210	14, 7035	42
40, 8214	14, 7035	42
40, 8214	14, 7035	28
40, 8210	14, 7035	28

In Figura 5.11 si può osservare la rotta nominale effettuata dal drone in un ambiente privo di ostacoli, tale rotta collega con delle linee rette gli waypoint impostati.

In Figura 5.12 si può osservare il risultato ottenuto con la terza simulazione, dove i punti rossi rappresentano sempre le misurazioni effettuate dal sensore simulato. Anche in questo caso la traiettoria alla quota maggiore risulta coincidente con la traiettoria nominale prevista; alla quota inferiore si nota che il drone in avvicinamento verso il muricciolo comprende di doversi muovere lungo l'asse Z e riesce a scavalcarlo per poi tornare alla quota originaria e raggiungere il goal impostato. Durante questo test il rischio consisteva nell'osservare il drone muoversi parallelamente al muro nella speranza di incontrare uno spiraglio per aggirarlo all'interno del piano 2D di navigazione. L'idea alla base di quest'ultima simulazione era appunto quella di verificare la riuscita della pianificazione di percorso in campo 3D da parte dell'algoritmo, tale obiettivo si può considerare raggiunto.

Le misurazioni effettuate dal sensore hanno permesso di visualizzare la mesh ricreata a seguito della navigazione, visibile in Figura 5.13. Come si vede nell'immagine la mesh non

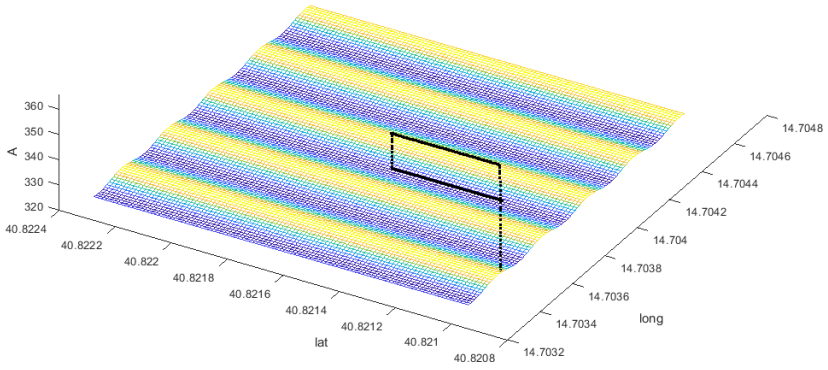


Figura 5.11: Rotta nominale della terza simulazione in ambiente privo di ostacoli

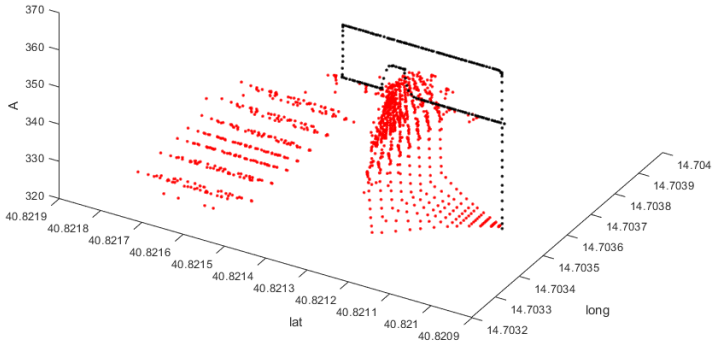


Figura 5.12: Rotta effettiva della terza simulazione

ricorda il muro effettivamente presente nel DEM, anzi visto che il sensore rileva misurazioni ad altezze differenti la mesh ricostruita non risulta chiarificatrice della realtà; tale ricostruzione però mostra ciò che incontra il drone durante il suo percorso, unica informazione necessaria per realizzare l'*obstacle avoidance*.

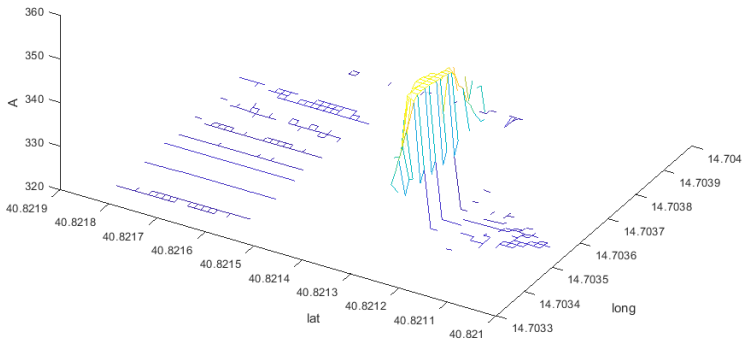


Figura 5.13: Ricostruzione del terreno a seguito della terza simulazione

Capitolo 6

Conclusioni

L'obiettivo di questo lavoro di tesi era dimostrare la fattibilità della navigazione autonoma di un APR in terra ignota con il solo ausilio di un sensore LIDAR a basso costo. Tale obiettivo si considera raggiunto a valle delle validazioni e dei test effettuati e presentati nel capitolo precedente.

Ad oggi, per questioni di sicurezza, i droni vengono abilitati alla navigazione autonoma solo con mappe installate a bordo o eventualmente in presenza di molteplici sensori, tra i quali sensori LIDAR con capacità di ripresa a 360°, atti a ricreare tutto l'ambiente circostante con estrema precisione. L'aver dimostrato la capacità di pianificazione di percorso anche con un sensore meno ambizioso risulta essere un passo avanti verso l'automazione degli APR a vantaggio di migliorie per le applicazioni nelle quali possono essere d'aiuto. La missione prevista per il drone oggetto di questa tesi era il monitoraggio di incendi, non potendo prevedere dove un incendio possa divampare la non necessità della mappa installata a bordo risulta essere un notevole grado di libertà; l'autonomia nell'evitare gli ostacoli permette agli operatori di concentrarsi sulla missione primaria, ovvero sfruttare i dati inviati dal velivolo per porre fine all'incendio nel minor tempo possibile, invece di dover controllare la traiettoria del velivolo che altrimenti perderebbe parte della sua efficacia nella partecipazione alla missione.

Si sottolineano alcune problematiche che possono non essere gestite correttamente dall'algoritmo presentato in questa tesi:

- Ostacoli in movimento esattamente di fronte al velivolo, quali ad esempio volatili, non possono essere previsti e evitati in tempo reale, per questa problematica si consiglia l'ausilio di un sonar posto almeno nel verso della direzione di avanzamento (per questione di sicurezza sarebbero opportuni sensori verso i quattro punti cardinali);

- Ostacoli sospesi verrebbero percepiti come limiti di ostacoli sottostanti in realtà inesistenti, ad esempio un ponte verrebbe ricostruito come una collina; il velivolo riuscirebbe ad evitare questa tipologia di ostacoli scavalcandoli, ma non potrebbe valutare la possibilità di volarci al di sotto.

Per tali problematiche si rimanda l'approfondimento in studi futuri.

Ringraziamenti

Ringrazio la professoressa Battipede e l'ingegner De Vivo che mi hanno dato la possibilità di lavorare ad un progetto che mi ha fortemente interessata e stimolata.

Al termine del mio percorso universitario è doveroso un ringraziamento alla mia famiglia, senza la quale non potrei essere ora qui a scrivere queste righe.
Ringrazio mia mamma per avermi saputa ispirare con la tenacia, l'impegno e la passione che mette sempre nel suo lavoro, sarà un successo per me diventare anche solo un quarto della donna che è lei.
Ringrazio mio papà che con ogni "fai la brava" ha saputo ricordarmi il percorso giusto da seguire e come renderlo orgoglioso di me.
Ringrazio mia sorella, amica per la vita, per la gioia con cui ha accolto ogni mio ritorno a casa, mi ha dato la forza di affrontare ogni settimana.

Ringrazio Alessio per sapermi capire, stare accanto e amare in ogni circostanza e senza alcun dubbio, mi mancherebbe anche se non l'avessi mai conosciuto.

Ringrazio Camilla per avermi sempre saputo spronare ad essere la versione migliore di me stessa, quella che vede lei. *Semper fidelis.*

Bibliografia

[1] Hart P. E., Nilsson N., Raphael B. (1968), A Formal Basis for the Heuristic Determination of Minimum Cost Paths.

[2] Sun X., Koenig S. (2007), The Fringe-Saving A* Search Algorithm - A Feasibility Study.

[3] Sun X. Koenig S., Yeoh W. (2008), Generalized Adaptive A*.

[4] Koenig S., Likhachev M. (2002), D* Lite.

[5] Pearl J. (1985), Heuristics: Intelligent Search Strategies for Computer Problem Solving.

[6] Stentz A. (1995), The focussed D* algorithm for real-time replanning.

[7] Koenig S., Likhachev M. (2005), Fast Replanning for Navigation in Unknown Terrain.

[8] Mauro Lo Brutto (2015), I SAPR (Sistemi Aeromobili a Pilotaggio Remoto) per il rilievo e il monitoraggio del territorio: stato dell'arte e applicazioni fotogrammetriche.

[9] "Drone" in Grande Dizionario di Italiano, Garzanti Linguistica.

[10] ICAO, Unmanned Aircraft Systems (UAS)

[11] B. P. Tice (1991), Unmanned Aerial Vehicles. The Force Multiplier of the 1990s.

[12] Kozera C. A. (2018), Military Use of Unmanned Aerial Vehicles - A Historical Study.

[13] Buckley J. J. (1998), Air Power in the Age of Total War.

[14] Hallion R. P. (2003), Taking Flight: Inventing the Aerial Age, from Antiquity Through the First World War.

[15] La nascita dei droni, `http://www.gtfondazione.org/industria-4-0/la-nascita-dei-droni/`

[16] ArduPilot, `http://ardupilot.org/ardupilot/index.html`

[17] Extended Kalman Filter Navigation Overview and Tuning, `http://ardupilot.org/dev/docs/extended-kalman-filter.html`

[18] Code Overview (Copter), `http://ardupilot.org/dev/docs/apmcopter-code-overview.html`

[19] Copter Attitude Control, http://ardupilot.org/dev/docs/apmcopter-programming-attitude-control-2.html

[20] Copter Position Control and Navigation, http://ardupilot.org/dev/docs/code-overview-copter-poscontrol-and-navigation.html

[21] MAVLink Basics, http://ardupilot.org/dev/docs/mavlink-basics.html

[22] Copter Commands in Guided Mode, http://ardupilot.org/dev/docs/copter-commands-in-guided-mode.html

[23] MAVLink Developer Guide, https://mavlink.io/en/

[24] SITL Simulator (Software in the Loop), http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

[25] Setting up SITL on Linux, http://ardupilot.org/dev/docs/setting-up-sitl-on-linux.html

[26] DroneKit Tutorial, http://ardupilot.org/dev/docs/droneapi-tutorial.html

[27] Latombe J. (2012), Robot Motion Planning.

[28] Wilson R. J. (1996), Introduction to Graph Theory.

[29] Dijkstra E. W. (1959), A Note on Two Problems in Connexion with Graphs.

[30] Koenig S., Likhachev M., Liu Y., Furcy D. (2004), Incremental Heuristic Search in Artificial Intelligence.

[31] Stentz A. (1994), Optimal and Efficient Path Planning for Partially-Known Environments.