Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Tesi di Laurea Specialistica in Ingegneria Informatica e Automatica

Progettazione di un sistema reattivo per la navigazione autonoma di un drone in ambienti sconosciuti

Relatore:
Prof. Eliseo Clementini

Laureando:
Luca Di Stefano

Correlatore:
Dott. Enrico Stagnini

Anno Accademico 2015-2016

Indice

Elenco delle figure

Sommario

Il volo autonomo di un drone (UAV) può essere implementato da una gerarchia di sistemi di controllo. La ricerca punta principalmente allo sviluppo di sistemi per droni multirotore, che garantiscono costi contenuti e un'elevata manovrabilità.
Questa tesi affronta, in particolare, il problema della navigazione locale di un drone multirotore: l'obiettivo è gestire la navigazione punto-punto, cioè portare il drone da una posizione di partenza a una di arrivo, adeguando la traiettoria agli eventuali ostacoli rilevati durante la navigazione.
Presentiamo un sistema mapless e reattvo, che applica tecniche di elaborazione delle immagini e valutazioni qualitative su una mappa di profondità acquisita dal drone. La soluzione proposta può essere implementata sulla stazione di terra, in modo da ridurre i requisiti computazionali del drone.
Dimostriamo infine le capacità del sistema in un ambiente simulato.

Abstract

Unmanned Aerial Vehicles (UAVs) can achieve autonomous flight capabilities by relying on multiple control systems. Multirotor drones are the main research target, due to their low cost and high maneuverability.
This thesis focuses on local path planning, which is defined as the ability to ensure safe point-to-point navigation in an unknown environment.
We present a mapless, reactive system for a multirotor UAV.
The algorithm applies image processing techniques and qualitative evaluations on a depth map acquired by the UAV in order to assess the safety of the current flight plan and to establish an alternative path when unforeseen obstacles are detected.
The system is designed to run on a ground control station, thus minimizing the UAV requirements.
We finally evaluate our system in simulated environments, with satisfying results. Implementation and testing on a real UAV is underway.

Capitolo 1

Introduzione

I sistemi UAV (Unmanned Aerial Vehicles), meglio noti come droni, stanno guadagnando una popolarità sempre crescente in ambito civile: [6] fornisce una panoramica degli avanzamenti tecnologici che hanno favorito questa diffusione e documenta un mercato composto da oltre 1000 modelli, destinato a crescere ulteriormente nel corso del prossimo decennio in termini di varietà dell'offerta e di fatturato complessivo.

I settori interessati all'impiego di droni (teleguidati o automatizzati) possono essere suddivisi in due grandi categorie applicative:

- Fotografia e raccolta dati aerea: un drone permette l'acquisizione di immagini o altri dati di telemetria a basso costo e con rischi limitati. Questa caratteristica li rende adatti a supportare i soccorsi nelle operazioni di emergenza o a monitorare aree altrimenti inaccessibili: [2] presenta un'applicazione nel campo della vulcanologia.

  Un'altra applicazione appartenente a questa branca è quella della sorveglianza anti-intrusione e del supporto alle operazioni di polizia.

- Trasporto di merci e carichi postali: esperimenti in tal senso sono stati effettuati da aziende come Amazon [1] e DHL [11]. I vantaggi interesserebbero sia i trasporti in aree rurali, dove i collegamenti stradali sono più accidentati, sia quelli nelle aree densamente urbanizzate in cui la consegna via terra è ostacolata dal traffico.

L'interesse della comunità accademica riguarda soprattutto i seguenti campi:

- Stima e controllo dello stato del drone, cioè dei valori di posizione, orientamento e velocità. La capacità di portare il drone in uno stato desiderato è infatti una condizione fondamentale allo sviluppo di sistemi di pilotaggio automatico e navigazione.

- Sistemi anticollisione, volti a mantenere il drone ad una certa distanza di sicurezza rispetto agli ostacoli percepiti dai sensori di bordo.

- Sistemi di navigazione e guida autonoma, il cui obiettivo è comandare il drone in modo che compia una "missione", solitamente definita da un operatore umano sotto forma di una sequenza di *waypoint* che il drone deve attraversare. Il termine "guida autonoma" viene talora usato per quei sistemi che includono meccanismi decisionali più avanzati, capaci di pianificare in maniera autonoma e dinamica le missioni basandosi su un input minimo da parte dell'utente.

La produzione scientifica si concentra in larga maggioranza sui droni di dimensioni ridotte, cioè tali da non superare i cinque metri di dimensione, classificati sotto il termine MAV (*Micro Aerial Vehicles*).

La tipologia degli UAV multirotore, formati da una fusoliera centrale e una corona di rotori (tipicamente da 3 a 8), ha riscosso particolare successo nel mondo della ricerca (Figura 1.1). Le motivazioni citate in [6, pp. 13, 19] sono molteplici e spaziano dai bassi costi di costruzione all'elevata manovrabilità e alla possibilità di trasportare un carichi paganti di dimensioni adeguate a vari tipi di missione. La molteplicità di formati disponibili ha costituito un ulteriore vantaggio competitivo per questa categoria di UAV, che si presta sia a ricerche sulla navigazione al chiuso [4] che ad applicazioni in ambienti aperti, che richiedono tempi di autonomia e carichi paganti molto maggiori.

L'obiettivo di questa tesi è implementare un sistema di navigazione autonoma che sia efficace in zone urbane o comunque a elevata densità di ostacoli e che faccia affidamento su un numero ridotto di sensori. Le considerazioni di cui sopra hanno portato alla scelta del multirotore come modello di riferimento per il nostro lavoro.

I vantaggi della soluzione proposta sarebbero molteplici: innanzitutto la barriera economica all'utilizzo degli UAV verrebbe ulteriormente ridotta grazie al basso costo dei sensori.

Figura 1.1: Esempi di MAV multirotore con quattro, sei e otto pale rotanti.

Un altro risultato desiderabile è la riduzione della massa al decollo del drone, che costituisce un fattore rilevante per la regolamentazione del volo e che va quindi (perciò) minimizzato al fine di favorire la diffusione dei droni negli ambiti civili.

La tesi è strutturata come segue.

Nel **capitolo 2** il problema della navigazione autonoma viene definito nel dettaglio; si fornisce una panoramica dello stato dell'arte nel campo e si delineano gli obiettivi che sono stati perseguiti nel corso del lavoro di tesi.

Nel **capitolo 3** vengono introdotte alcune nozioni fondamentali per il lavoro svolto.

Il **capitolo 4** descrive la fase di progettazione dell'algoritmo di navigazione locale.

Con il **capitolo 5** si passa alla fase implementativa e di simulazione, con l'esposizione di alcuni risultati sperimentali.

Il **capitolo 6** riassume le conclusioni del lavoro e i possibili sviluppi futuri.

Capitolo 2

Stato dell'arte e obiettivi

2.1 Definizione del problema

Un sistema di navigazione autonoma è un insieme di apparati hardware e software che consentono a un drone di eseguire una missione senza la guida di un pilota umano.

La missione viene definita come una sequenza di punti nello spazio che il drone dovrà raggiungere, comunemente detti *waypoints*.

Il suddetto sistema può essere *on-board* o distribuito, cioè parzialmente dislocato sulla stazione di controllo a terra (GCS): la necessità di ricorrere ad un'architettura distribuita è soprattutto dovuta alle limitate capacità computazionali dell'hardware di bordo.

2.2 Stato dell'arte

La rassegna [13] espone nel dettaglio le diverse soluzioni disponibili in letteratura, catalogandole in base ai rispettivi fondamenti teorici.

Si può immediatamente constatare che la problematica della "navigazione e guida autonoma" è in realtà composta da diversi sottoproblemi il cui grado di astrazione è via via crescente:

1. Evitare l'impatto del drone con un ostacolo;
2. Aggirare un ostacolo di piccole dimensioni;

3. Raggiungere un waypoint;

4. Stabilire se un waypoint è irraggiungibile...

Una soluzione ingegneristica tipicamente adottata in casi simili è quella codificata nel pattern architetturale *Layers*, che consiste nel suddividere il sistema in componenti stratificate in base al loro livello d'astrazione [5]. Per alcuni esempi dell'applicazione di un'architettura stratificata al problema della navigazione autonoma si vedano [15] e [22].

Questa architettura, oltre ai vantaggi forniti dall'elevata modularità, consente agli strati più "alti" di operare nell'ipotesi che tutti i problemi "meno astratti" vengano risolti negli strati inferiori.

La collaborazione può avvenire anche nel senso opposto: quando un sottosistema stabilisce che le proprie risorse non sono sufficienti a risolvere un dato problema, può passarlo allo strato superiore.

Un approccio analogo può essere riscontrato nello stack TCP/IP, dove ad esempio lo strato di rete può ignorare il fatto che i pacchetti IP dovranno attraversare una sequenza di collegamenti punto-punto: si tratta infatti di una responsabilità delegata al sottostante livello *data link*.

La Figura 2.1 schematizza l'architettura appena descritta e introduce i termini comunemente usati per classificare i diversi sottoproblemi della navigazione e guida autonoma. La rassegna dello stato dell'arte verrà suddivisa nelle medesime categorie.

2.2.1 Controllo di missione

Si parla di *mission planning* per indicare quelle operazioni decisionali che interessano la missione nel suo insieme: questi sistemi dovrebbero cioè essere in grado di avviare, modificare o annullare la missione in base a direttive stabilite dall'utente.

Per il problema della pianificazione di missione sono state proposte soluzioni basate su sistemi multi-agente (ad esempio SOAR [20]), macchine a stati finiti gerarchiche (HFSM) o più recentemente mediante Behavior Trees (BT) [17].

Un altro approccio è basato su tecniche di Model-Driven Engineering (MDE) che partono da una descrizione ad alto livello della missione (tipologia, area da sorvolare) per generare un piano di volo per uno o più droni [7].

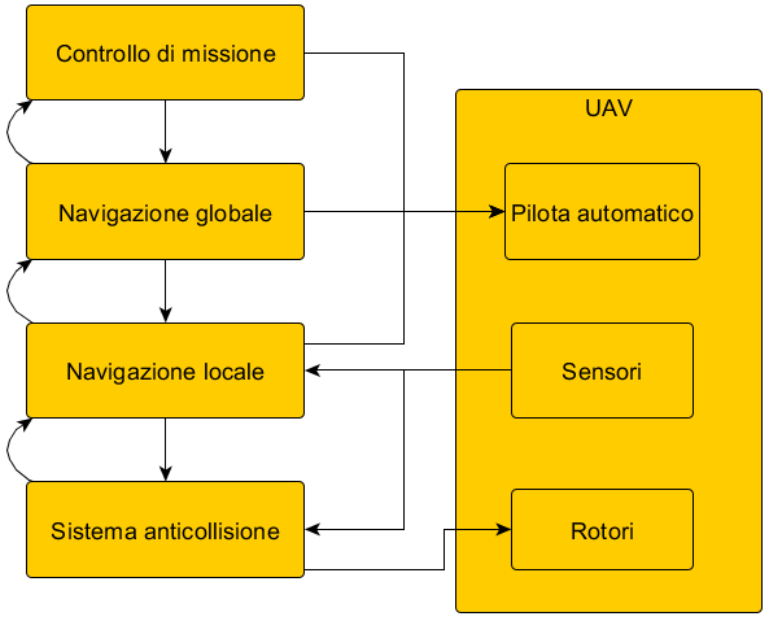


Figura 2.1: Architettura stratificata del sistema di controllo. Notiamo che il sistema anticollisione è l'unico che comanda esplicitamente i rotori dell'UAV; gli altri strati si limitano a comunicare con il pilota automatico di bordo. Immagine adattata da [15].

2.2.2 Sistemi di navigazione globale

Un *global path planner* è un sistema capace di elaborare un piano di volo ottimale in base ad un modello dell'area della missione e un insieme di waypoint prestabiliti.

Il piano di volo risultante può tuttavia non essere attuabile se le conoscenze dell'area di volo sono inesatte o incomplete: questa limitazione è rilevante soprattutto nelle aree urbane, per le quali raramente sono disponibili modelli tridimensionali sufficientemente dettagliati.

L'ottimalità può essere intesa in termini di tempo, lunghezza del percorso, dispendio di energia o di una combinazione di questi e altri fattori. Ad esempio si può desiderare che il drone rimanga presso un particolare waypoint per un certo intervallo di tempo, oppure stabilire una maggiore tolleranza rispetto alle coordinate da raggiungere (per cui è sufficiente che il drone entri in un intorno del waypoint). Questi fattori ovviamente influenzano il calcolo del percorso ottimo.

Se la sequenza dei waypoint è prestabilita, il sistema deve trovare le traiettorie

ottimali tra ciascun punto e il successivo.

Riportiamo le principali metodologie note in letteratura ([13], [9]).

Road Maps Si induce un grafo sullo spazio navigabile e si traduce il problema in una ricerca di cammini minimi. Alcune tipologie di grafo:

- *Grafo di visibilità:* i nodi corrispondono ai vertici degli ostacoli; due nodi sono connessi da un arco solo se il segmento che li congiunge è interamente contenuto nello spazio navigabile.
- *Decomposizione in celle:* lo spazio navigabile viene suddiviso in volumi (celle); le celle adiacenti sono connesse da archi. La suddivisione può essere uniforme o "ottimizzata" rispetto alla conformazione dello spazio.

Campi di potenziale Viene definita una funzione di potenziale sullo spazio navigabile. Agli ostacoli vengono associate forze repulsive, mentre l'obiettivo esercita una forza di attrazione tale che la sua posizione è un punto di minimo globale del potenziale (Figura 2.2). Quindi si calcola il percorso simulando la traiettoria di una particella sottoposta alle forze così definite.

Il vantaggio di questo metodo è la semplicità dal punto di vista sia concettuale che computazionale. Tuttavia, se il potenziale consente l'esistenza di minimi locali, l'algoritmo potrebbe non dare il risultato sperato: la particella potrebbe concludere il proprio percorso in uno di questi punti di minimo e non raggiungere mai l'obiettivo. Di conseguenza qualsiasi ricerca in questo ramo deve aggirare questo problema: si cerca ad esempio di definire funzioni prive di minimi locali o si applicano tecniche di ricerca capaci di riconoscerli ed evitarli.

Metodi di ottimizzazione Il problema viene posto in un'ottica di ottimizzazione numerica in cui waypoint e ostacoli sono vincoli da rispettare. Teoricamente questi metodi offrono soluzioni ottimali, e possono essere estesi con ulteriori vincoli che caratterizzino le dinamiche del drone, ma hanno un elevato costo computazionale, che costituisce la principale sfida di ricerca.

Spesso però l'ordine dei waypoint non è prestabilito ma va a sua volta ottimizzato: ad esempio si vuole che il drone effettui operazioni di telemetria in n località nel minor

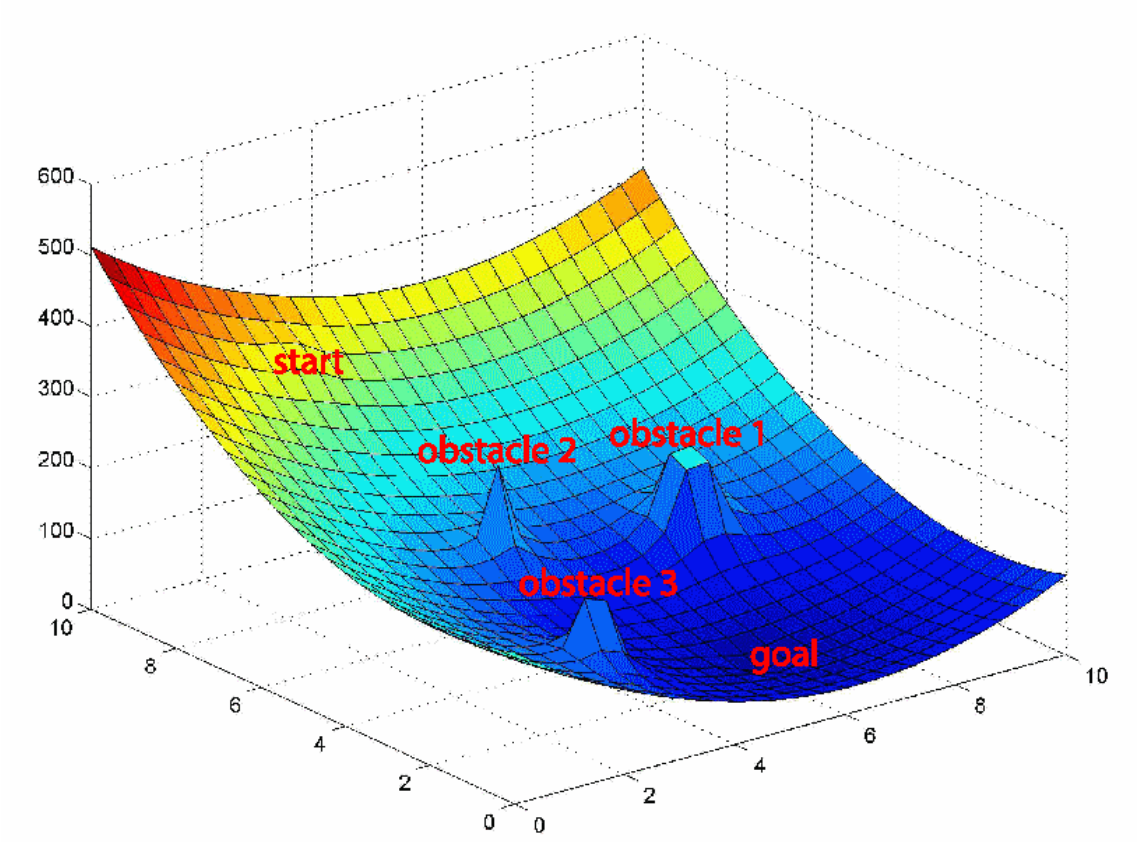


Figura 2.2: Navigazione basata su campi di potenziale. Fonte: [18].

tempo possibile. Si tratta di una variante del problema del commesso viaggiatore (TSP), che è notoriamente NP-completo; tuttavia, istanze del problema con n sufficientemente piccolo possono essere risolte in tempi accettabili grazie a tecniche di programmazione lineare [10]: un esempio particolarmente noto in tal senso è il risolutore Concorde [3].

Visto che prendere in considerazione gli ostacoli potrebbe rendere intrattabile il problema, un approccio sub-ottimale ma accettabile consiste nel risolvere il TSP ignorando la loro presenza, per poi determinare i percorsi ottimi tra waypoint adiacenti per mezzo di uno dei metodi sopra descritti [16].

2.2.3 Navigazione locale

Il *local path planning* consiste nell'interpretare i dati rilevati dai sensori di bordo per supportare la navigazione del drone su distanze relativamente brevi: ad esempio verso

il prossimo waypoint della missione.

La definizione del problema può variare in base alle caratteristiche del drone e dell'ambiente in cui opera.

Una prima possibilità è quella di costruire una mappa locale su cui impiegare i metodi già visti nell'ambito della navigazione globale. Bisogna tuttavia considerare che stavolta il modello dell'ambiente è in continuo aggiornamento, può arricchirsi di ostacoli precedentemente imprevisti ed è soggetto al rumore dei sensori. Si cerca quindi di semplificare il sistema in modo che possa ricalcolare periodicamente il percorso in base alle ultime conoscenze acquisite.

Se il drone non può conoscere la propria posizione per mezzo di sensori GPS o simili, l'operazione di mappatura deve includere una fase di autolocalizzazione: si parla quindi di SLAM (Simultaneous Location and Mapping).

2.2.4 Sistemi anticollisione

La missione di questi sistemi è *time critical*, quindi richiede una reazione tempestiva agli stimoli esterni: per questo motivo essi risiedono tipicamente sul computer di bordo del drone e vengono messi in funzione con altissime frequenze (dalle decine alle centinaia di Hertz).

Per raggiungere queste prestazioni si ricorre a sistemi estremamente veloci, solitamente *mapless*. Il drone, in altre parole, non elabora una vera e propria mappa dell'ambiente, ma agisce istantaneamente in funzione dei dati dei sensori.

Un'altra caratteristica di questi sistemi è che non tengono conto delle decisioni prese in precedenza, limitandosi a definire la migliore strategia rispetto alla percezione corrente. Si parla quindi di sistemi *reattivi*.

Campi di potenziale Il metodo, già descritto nell'ambito della navigazione globale, trova larga applicazione anche in questo contesto. Anche in questo caso vale il discorso sul problema dei minimi locali.

Flusso ottico Confrontando due immagini prese dal drone a un breve intervallo di tempo, è possibile quantificare lo spostamento di regioni di pixel caratteristiche (*features*). Questo spostamento, detto flusso ottico (*optic flow*), è direttamente proporzionale alla velocità del drone e inversamente proporzionale alla distanza

tra il drone e l'ostacolo. Una regione con un flusso ottico elevato corrisponde quindi a un ostacolo molto vicino.

In natura il flusso ottico è ritenuto alla base dei meccanismi navigazionali di numerose specie di insetti volanti, incluso quello di aggiramento degli ostacoli [21].

2.3 Hardware e software per l'utente finale

Gran parte dell'interesse accademico e commerciale per il mondo degli UAV è dovuta alla disponibilità di prodotti che consentono la creazione di soluzioni altamente personalizzate.

In questa sezione riporteremo le tecnologie a nostro avviso più importanti e che saranno prese in considerazione per un'implementazione fisica del sistema.

2.3.1 PixHawk

PixHawk [1] è una board di pilotaggio automatico che può essere impiegata su diverse tipologie di UAV (multirotori, elicotteri, aeroplani, robot di terra...).

Sviluppata nell'ambito del progetto PX4, la scheda (attualmente giunta alla seconda versione) è completamente open-source. Tra le specifiche degne di nota ricordiamo:

- Microcontrollore principale STM32F427 basato su processore ARM Cortex-M4F (168 MHz) e provvisto di 256 KB di RAM e 2 MB di memoria flash;
- Controllore di *failsafe* STM32F103, che assume il controllo del mezzo in caso di avaria della CPU primaria;
- Sensori integrati: giroscopio, magnetometro, accelerometro, barometro;
- Numerose interfacce per l'inserimento di periferiche (UART, I2C, CAN...);
- Slot di memoria microSD.

[1] PX4 Dev Team. `http://pixhawk.com/`

2.3.2 ArduPilot e Mission Planner

Il progetto ArduPilot [2] è composto da tre firmware open-source (Copter, Plane, Rover) che forniscono funzionalità avanzate a droni di varia tipologia: Copter è orientato ai multirotori e agli elicotteri, Plane agli aeroplani, Rover ai robot di terra.

I firmware sono compatibili con molte schede di controllo, tra cui la PixHawk.

Il software per PC Mission Planner (Figura 2.3) permette di sfruttare appieno le capacità dei firmware ArduPilot: ad esempio è possibile far eseguire al drone una missione pianificata in precedenza, oppure inviare comandi in tempo reale. Oltre all'interfaccia grafica, Mission Planner può essere comandato mediante script Python o attraverso richieste HTTP. Questo permette di automatizzare ed estendere il software senza modificarne il suo codice sorgente.

Si prevede quindi che un'implementazione reale di un sistema di navigazione autonoma collochi lo strato di *collision avoidance* all'interno del firmware ArduCopter, mentre i sistemi di controllo di più alto livello si interfacceranno con Mission Planner.

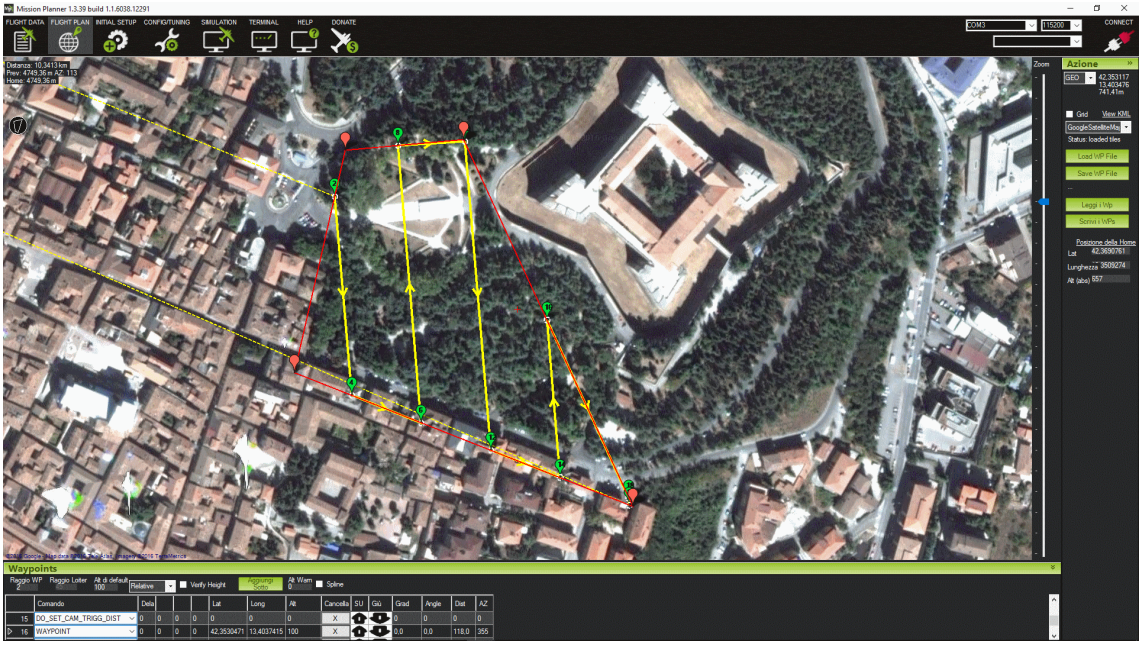


Figura 2.3: Una schermata di Mission Planner.

[2] DIY Drones. `http://ardupilot.org/`

2.4 Obiettivi

L'obiettivo della tesi è proporre un sistema di navigazione locale che guidi il drone verso un waypoint prestabilito, all'interno di un ambiente che può contenere ostacoli sconosciuti.

Il sistema dovrà essere implementato sulla GCS, in modo da permetterne l'impiego su droni dalle limitate risorse computazionali. Un ulteriore requisito imposto da questa scelta sarà quindi un'elevata velocità di esecuzione, per bilanciare il ritardo di comunicazione tra il drone e la stazione di terra.

Per questo si prevede che il sistema sarà mapless e reattivo, basato sulla percezione degli ostacoli fornita da un LIDAR e un sensore ottico orientati nel verso di marcia del drone. Il controllo della posizione avverrà invece tramite antenna GPS.

Questa decisione impone un altro vincolo progettuale: non sfrutteremo appieno i gradi di libertà offerti dal multirotore, limitandoci a navigare nello spazio delimitato dal campo visivo del drone stesso.

D'altro canto ci serviremo di altre capacità tipiche di questa categoria di UAV, quali il volo stazionario e la possibilità di effettuare imbardate sul posto.

Il sistema di navigazione cercherà di stabilire una rotta sicura al netto delle conoscenze a disposizione, ma presuppone la presenza di un sistema anticollisione di bordo in grado di reagire tempestivamente a perturbazioni impreviste della traiettoria (causate ad esempio dal vento o da malfunzionamenti nei rotori) o alla comparsa improvvisa di ostacoli. Questo sistema inoltre avrà una percezione più ampia, seppure ad un raggio minore, rispetto a quella offerta dal LIDAR e dal sensore ottico.

Capitolo 3

Nozioni preliminari

Presentiamo in questo capitolo alcuni concetti cui abbiamo fatto ricorso durante la progettazione dell'algoritmo.

3.1 Sistemi di coordinate

3.1.1 Sistema di coordinate ECEF

La scelta dei sistemi di riferimento geografici da utilizzare è di fondamentale importanza. Il sistema geodetico è senza dubbio il più conosciuto, anche tra i meno esperti: ogni punto dello spazio viene individuato da una coppia di coordinate (latitudine λ, longitudine φ) e da un valore di *altitudine* h.

Tuttavia l'algoritmo deve valutare la posizione degli ostacoli e dell'obiettivo rispetto al MAV, per cui le coordinate geodetiche si rivelano inadeguate. Per questo motivo prevediamo di utilizzarle solo a livello di "interfaccia" con altri sistemi, mentre l'algoritmo userà internamente il sistema di riferimento ECEF (Earth-Centered, Earth-Fixed).

In questo sistema l'origine degli assi è posta al centro della Terra; gli assi x e y intersecano l'equatore in corrispondenza, rispettivamente, del meridiano di Greenwich e del 90° meridiano Est; l'asse x, infine, passa per il Polo Nord (Figura 3.1). Le coordinate sono espresse in metri.

La trasformazione da un sistema di coordinate all'altro è immediata.

Siano ad esempio

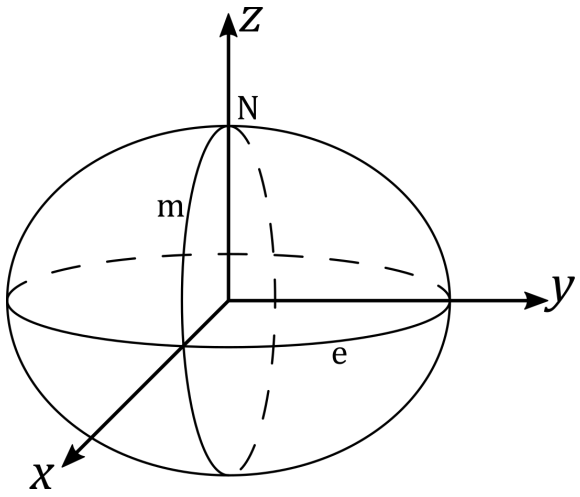


Figura 3.1: Sistema ECEF. *N* denota il polo Nord; *m* il meridiano di Greenwich; *e* l'equatore.

$$P_g = \begin{pmatrix} \lambda \\ \varphi \\ h \end{pmatrix}$$

le coordinate di un punto P nel sistema geodetico. Per ottenere le corrispondenti coordinate ECEF bisogna ricordare che le coordinate geodetiche si basano su un *ellissoide di riferimento* che approssima la forma della Terra.

Lo standard usato dal sistema GPS è noto come WGS 84 e definisce l'ellissoide attraverso tre parametri:

Semiasse maggiore $R_{Ea} = 6378137.0$ m

Semiasse minore $R_{Eb} \simeq 6356752.31$ m

Schiacciamento $f = 1/298.257223563.$

Noti questi parametri, le coordinate ECEF si ottengono come segue:

$$P_e = \begin{pmatrix} x_e, \\ y_e, \\ z_e \end{pmatrix} = \begin{pmatrix} (N_E + h)\cos\varphi\cos\lambda \\ (N_E + h)\cos\varphi\sin\lambda \\ \left(N_E\left(1 - \boldsymbol{e}^2\right) + h\right)\sin\varphi \end{pmatrix}$$

dove

$$\boldsymbol{e} = \frac{\sqrt{R_{E_a}^2 - R_{Eb}^2}}{R_{Ea}}$$
$$N_E = \frac{R_{Ea}}{\sqrt{1 - e^2 \sin^2(\varphi)}}$$

sono rispettivamente la *prima eccentricità* e il *raggio di curvatura del primo verticale* dell'ellissoide.

3.1.2 Assi corpo del MAV

Le coordinate ECEF semplificano notevolmente i calcoli vettoriali e ci permettono anche di calcolare la posizione degli *assi corpo* del drone.

Gli assi corpo sono una terna cartesiana la cui origine è posta nel baricentro del drone stesso. Usiamo la seguente definizione intuitiva:

1. L'asse x' è diretto nel verso di marcia del drone;
2. y' punta verso il suo fianco sinistro;
3. z' è diretto verso l'alto.

Qualsiasi rotazione del MAV può essere espressa da una terna di angoli (α, β, γ) che corrispondono all'entità della rotazione attorno a ciascun asse corpo (Figura 3.2). Definiamo quindi:

Rollio la rotazione attorno all'asse x';

Beccheggio quella attorno all'asse y';

Imbardata quella attorno all'asse z'.

Se il rollio e il beccheggio del MAV sono entrambi nulli, allora il drone è perpendicolare alla superficie terrestre. L'imbardata è invece pari a zero quando il MAV punta verso Nord.

Possiamo ora definire una trasformazione che, date le coordinate ECEF e gli angoli di rollio, beccheggio, imbardata del MAV, restituisca i tre versori degli assi corpo. Per

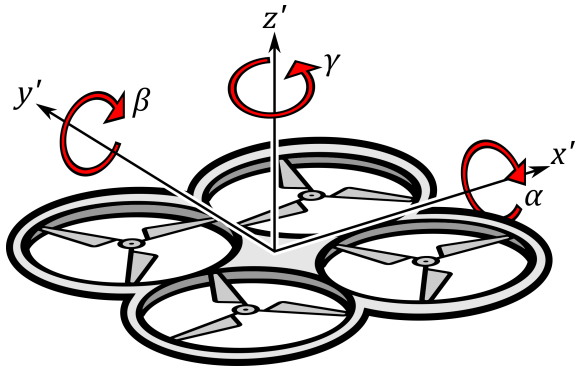


Figura 3.2: Assi corpo del MAV e relativi angoli di rotazione.

ottenere una maggiore accuratezza bisognerebbe considerare la forma ellissoidale della Terra, ma l'errore è del tutto trascurabile dal momento che l'algoritmo opera nell'ordine di grandezza delle decine di metri.

La prima parte tiene conto soltanto delle coordinate ECEF ed è una semplice costruzione geometrica.

Sia D la posizione del drone e N quella del polo Nord nel sistema ECEF. Indichiamo invece con O l'origine degli assi.

Sia n la retta su cui giace il segmento ON. Allora esiste un'unica retta r ortogonale a OD e che interseca il punto D e la retta n.

A questo punto definiamo:

x'' il versore parallelo a r e applicato in D;

z'' il versore parallelo a $\vec{(}OD)$, anch'esso applicato in D;

y'' il prodotto vettoriale $z'' \times x''$ (Figura 3.3).

Se i tre angoli (α, β, γ) sono tutti nulli allora la terna così definita costituisce gli assi corpo del MAV; in caso contrario occorre applicare tre rotazioni alla terna di versori. Al termine otterremo i tre assi desiderati, x', y', z'.

Questa terna ci consente anche di descrivere lo spazio in termini di coordinate sferiche relative al drone. Ogni punto p nello spazio è infatti univocamente individuato dalla terna (ρ, θ, φ), dove:

- ρ è la distanza tra il punto e il MAV;

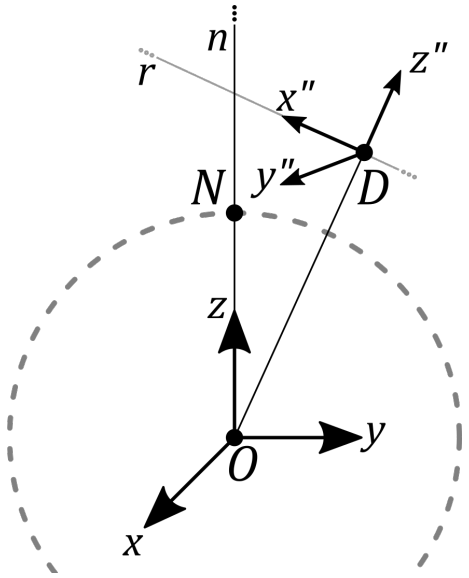


Figura 3.3: Calcolo degli assi corpo del MAV.

- θ (**azimuth**) è l'angolo che la proiezione di p sul piano del drone (individuato dai versori x', y') forma con l'asse x';
- φ (**elevazione**) è l'angolo che p forma con il piano del drone. Lo zenith corrisponde a un'elevazione di 90°.

3.2 Relazioni qualitative di visibilità nello spazio

Descriviamo brevemente la classificazione proposta in [8].

Siano B, C due oggetti nello spazio euclideo, $\mathbb{E}^3$. Convenzionalmente indicheremo con B l'*osservatore* della scena e con C il potenziale *ostacolo*.

Definizione 1 *Dati due punti $p, q \in \mathbb{E}^3$, diremo che p è* visibile *da q rispetto a un solido C se il segmento pq non interseca C.*

Notiamo che la relazione di visibilità è simmetrica.

Definizione 2 *Dati due oggetti B, C, possiamo scomporre lo spazio euclideo in tre volumi:*

Light Zone (LZ) *Contiene i punti visibili da tutti i punti di B rispetto a C;*

$$p \in LZ(B,C) \Leftrightarrow \forall q \in B, pq \cap C = \emptyset.$$

Shadow Zone (SZ) *Contiene i punti che sono completamente nascosti a causa di C;*

$$p \in SZ(B,C) \Leftrightarrow \forall q \in B, pq \cap C \neq \emptyset.$$

Twilight Zone (TZ) *Punti visibili solo da una parte dei punti di B.*

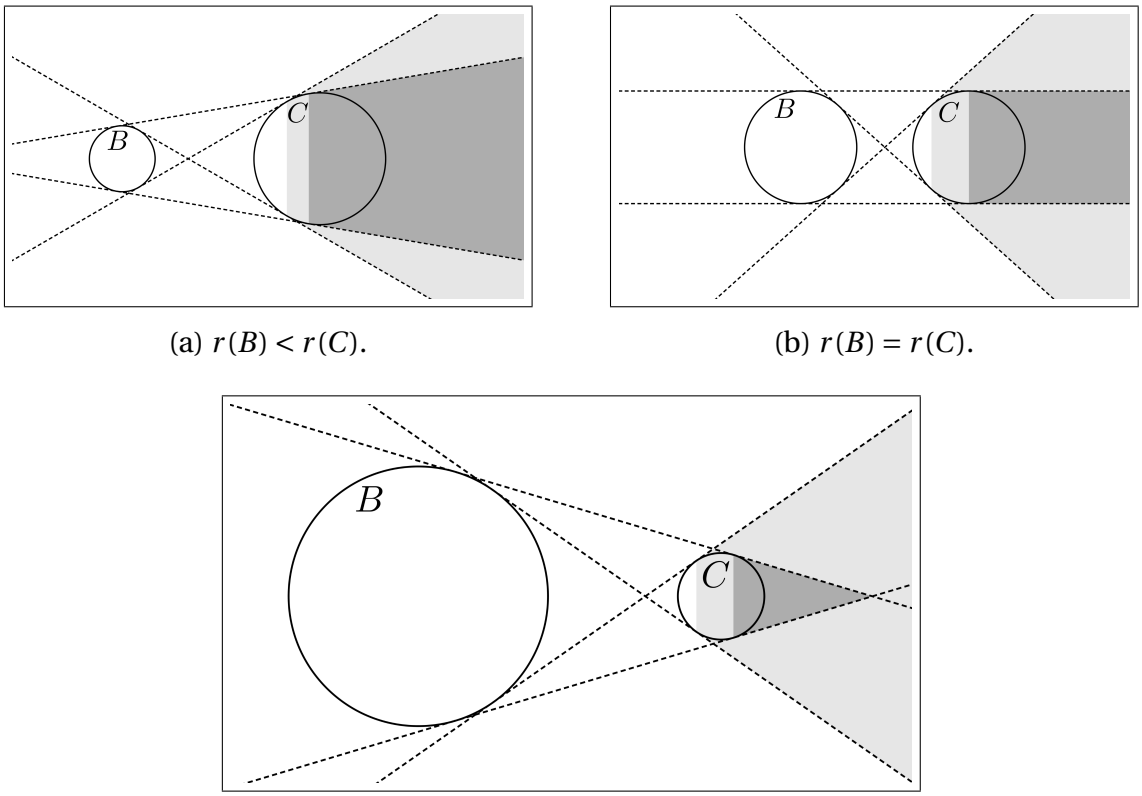
I nomi derivano dal fatto che possiamo interpretare B come una fonte di luce, e C come un oggetto che getta un'ombra SZ nello spazio opposto a B. La "zona di penombra" TZ viene colpita solo parzialmente dai raggi di luce.

Il calcolo delle tre zone è piuttosto semplice se B, C sono due sfere: infatti la forma di $SZ(B,C)$ dipende unicamente dai loro raggi e può essere un tronco di cono infinito, un cilindro infinito o un cono di altezza finita $TZ(B,C)$ è in ogni caso un tronco di cono infinito a cui viene rimossa la $SZ(B,C)$ e una calotta sferica di C che fa parte di $LZ(B,C)$ (Figura 3.4).

Notiamo che, se B si riduce a un punto, allora non esiste una $TZ(B,C)$.

Possiamo infine classificare la visibilità di un oggetto A da B rispetto a C calcolando la seguente matrice di valori booleani:

$$\Big(A \cap LZ(B,C) \quad A \cap TZ(B,C) \quad A \cap SZ(B,C)\Big).$$



(a) $r(B) < r(C)$.

(b) $r(B) = r(C)$.

(c) $r(B) > r(C)$.

Figura 3.4: Le tre possibili situazioni di visibilità di un osservatore sferico B rispetto a un ostacolo C, a sua volta sferico. La zona dal tratteggio più scuro è $SZ(B,C)$; l'altra $TZ(B,C)$.

Capitolo 4

Progettazione dell'algoritmo

4.1 Introduzione e requisiti

In questo capitolo presentiamo l'algoritmo di navigazione locale che costituisce il fulcro di questo lavoro di tesi.

Anche se finora abbiamo parlato della presenza di un LIDAR a bordo del drone, l'algoritmo riceve in input una mappa di profondità, cioè un'immagine raster in cui i valori dei pixel corrispondono a valori di distanza (Sezione 4.2).

Quindi l'unico requisito che imponiamo è che il drone sia capace di fornire una mappa di profondità, o comunque una raccolta di dati che permetta alla GCS di ricostruire la mappa in tempi brevi: in questo modo l'algoritmo è facilmente adattabile a diversi tipi di equipaggiamento sensoristico.

Se viene rilevato un ostacolo, l'algoritmo trova un possibile punto di aggiramento nello spazio libero compreso tra il drone e l'obiettivo.

4.1.1 Lavori correlati

[12] propone una strategia analoga a quella proposta in questo lavoro, ma con un approccio basato su una mappa tridimensionale: i dati dei sensori vengono immessi in un database di *voxels* (elementi di volume) su cui si basano le considerazioni sulla raggiungibilità dei waypoint e le pianificazioni di percorsi alternativi.

Per una soluzione mapless si veda invece [14]. L'idea è efficace quando il MAV si imbatte in ostacoli molto voluminosi (edifici, macchie di vegetazione), ma riteniamo sia sub-ottimale in molti altri casi: viene infatti classificato come un algoritmo di "wall-following" dagli autori stessi. Si è quindi deciso di mantenerla come strategia secondaria, che verrà messa in pratica qualora l'algoritmo principale dovesse trovarsi in situazioni che non può risolvere (Sezione 4.4.3).

4.2 Mappa di profondità

Supponiamo che i dati di profondità siano forniti all'algoritmo sotto forma di immagine raster, con una certa risoluzione (r_x, r_y) e pixel a valori reali compresi tra 0 e 1, detta *mappa di profondità*.

I valori sono normalizzati rispetto ai limiti di sensibilità del LIDAR: laddove la distanza è minore di quella minima d_{min} il pixel viene posto a zero, altrimenti il valore viene normalizzato rispetto alla massima distanza accettata d_{max}.

Il LIDAR acquisisce i dati spazzando un angolo orizzontale α e uno verticale β: perciò il campo visivo del drone va da $-\frac{\alpha}{2}$ a $+\frac{\alpha}{2}$ in orizzontale e da $-\frac{\beta}{2}$ a $+\frac{\beta}{2}$ in verticale.

Seguendo la convenzione della grafica computerizzata, indicheremo con $(0,0)$ il pixel nell'angolo in alto a sinistra; l'asse delle x è quello orizzontale, mentre quello delle y è verticale e orientato verso il basso.

Quindi, per ciascun punto $p = (\rho, \vartheta, \varphi)$ rilevato dal LIDAR (espresso nel sistema di coordinate sferiche centrato sul drone presentato in Sezione 3.1), un pixel p' nella mappa di profondità viene impostato come segue:

$$
\begin{aligned}
x_{p'} &= \left\lfloor \frac{1}{2} + \frac{\vartheta}{\alpha} \right\rfloor r_x \\
y_{p'} &= \left\lfloor \frac{1}{2} + \frac{\varphi}{\alpha} \right\rfloor r_y \\
v_{p'} &= \frac{\rho}{d_{max}}
\end{aligned}
$$

Viceversa, dato un pixel nella mappa, si può risalire al punto corrispondente (al netto dell'errore introdotto dal troncamento) invertendo le formule appena espresse.

4.2.1 Dilatazione della mappa

Vedremo in seguito che l'algoritmo di navigazione sfrutta la mappa di profondità per decidere la traiettoria del MAV, che viene rappresentato come un punto nello spazio. Ciò significa che gli spazi troppo vicini agli ostacoli vanno evitati, in quanto il volume del drone potrebbe causare collisioni impreviste.

Per farlo adatteremo una tecnica tipica dei sistemi di navigazione a due dimensioni e che descriviamo di seguito.

Si parte da una rappresentazione "top-down" dello spazio: gli ostacoli sono poligoni o curve sul piano e il drone stesso è inizialmente individuato da una forma geometrica, tipicamente una circonferenza

A questo punto si effettua un'operazione morfologica di *dilatazione* sugli ostacoli. Ricordiamo brevemente la definizione dell'operatore:

$$A \oplus B = \bigcup_{b \in B} A_b$$

dove A_b è la traslazione di A in b (Figura 4.1). Quando si effettua l'operazione su un'immagine, il secondo oggetto viene detto *elemento strutturante*.

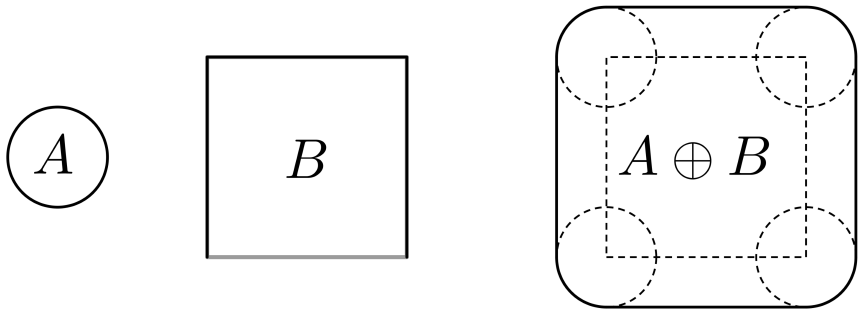


Figura 4.1: Effetto dell'operatore morfologico di dilatazione.

Se si usa come elemento strutturante la forma del drone (o una sua approssimazione, tipicamente un disco), si ottiene un "accrescimento" degli ostacoli e il drone stesso può essere considerato puntiforme. Se è possibile unire due punti con una curva che non interseca gli ostacoli dilatati, allora il drone può viaggiare tra quei due punti senza provocare collisioni (Figura 4.2).

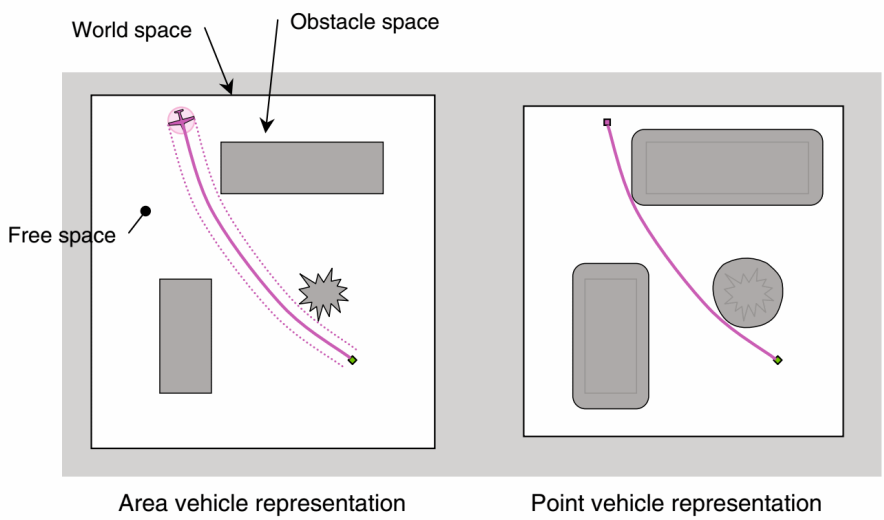


Figura 4.2: Navigazione in 2D con il drone ridotto a un punto. Tratta da [9].

Si vuole effettuare un'operazione analoga sulla mappa di profondità, così da escludere dallo spazio navigabile le zone a ridosso degli ostacoli. Non possiamo però applicare un unico elemento strutturante: infatti una dilatazione di piccola entità sarebbe insufficiente ad evitare l'impatto con gli ostacoli più vicini, ma d'altro canto se si aumenta il raggio dell'elemento strutturante si escludono vaste porzioni di spazio attorno agli oggetti più lontani, con risultati insoddisfacenti. Il problema, in altre parole, è dato dal fatto che la mappa di profondità è una rappresentazione "in prima persona" dell'ambiente.

La soluzione proposta (Algoritmo 1) consiste nello scomporre l'immagine in n "livelli" di differente profondità ed effettuare una dilatazione minore per i livelli lontani.

L'immagine I funge da accumulatore: i livelli lontani vengono sovrascritti, se necessario, dalle informazioni sugli ostacoli più vicini.

La Figura 4.3 mostra un esempio di applicazione dell'algoritmo su un'immagine di 256×256 pixel, con $n = 9$ e un elemento strutturante prima quadrato e poi circolare: la dimensione dell'elemento strutturante varia da un minimo di 22×22 a un massimo di 98×98 pixel.

Input: Una mappa di profondità M

$I \leftarrow$ immagine di risoluzione pari a M

$i \leftarrow 0 \ \forall i \in I$

$d_{max} \leftarrow$ massima distanza rilevata dal LIDAR

$r \leftarrow$ raggio di dilatazione massimo

$n \leftarrow$ numero di livelli.

for $s \leftarrow n, n-1, \ldots, 2, 1$ **do**

  $d \leftarrow \dfrac{s}{n}$

  $d_{-1} \leftarrow \dfrac{s-1}{n}$

  $L \leftarrow$ immagine di risoluzione pari a M

  $$L(x,y) = \begin{cases} M(x,y) & \text{se } M(x,y) \in (d_{-1}, d]; \\ 0 & \text{altrimenti.} \end{cases}$$

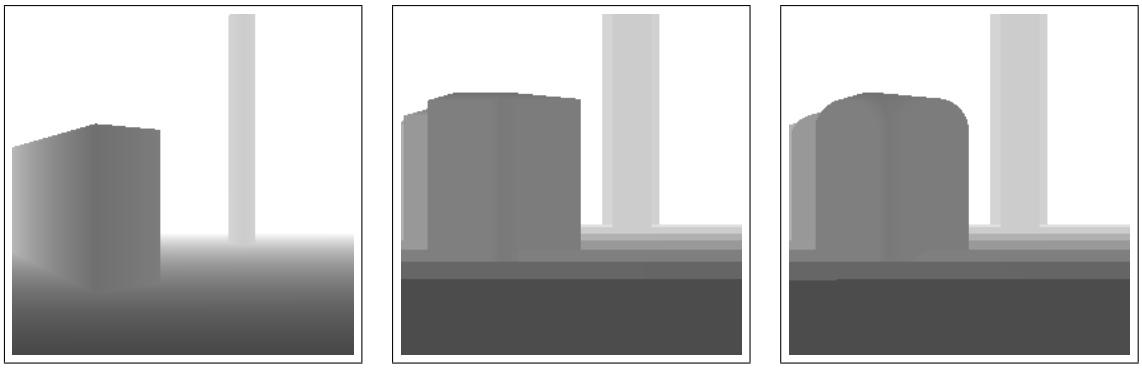
  $L \leftarrow$ `Dilata(`$L, r \cdot d$`)`

  $$I(x,y) = \begin{cases} L(x,y) & \text{se } L(x,y) \neq 0; \\ I(x,y) & \text{altrimenti.} \end{cases}$$

end

return I

Algoritmo 1: `DilataMappa(M)`.



(a) Mappa di profondità in input.

(b) Output (elemento strutturante quadrato).

(c) Output (elemento strutturante circolare).

Figura 4.3: Esempi di dilatazione di una mappa di profondità.

4.3 Controllo della visibilità

Questa funzione (Algoritmo 2) verifica la relazione di visibilità tra un waypoint e il drone, il quale viene visto come un punto centrato nel LIDAR. L'algoritmo calcola un volume di sicurezza attorno al waypoint per verificarne l'effettiva raggiungibilità: se l'intorno calcolato contiene degli ostacoli, il waypoint viene considerato irraggiungibile e spetterà all'algoritmo di navigazione gestire questa evenienza.

Al momento l'algoritmo non fa differenza tra situazioni di visibilità parziale e nulla: il waypoint viene classificato "Visibile" solo se interamente contenuto della zona di luce del drone.

Descriviamo brevemente i compiti svolti da alcune funzioni:

`ControllaAltitudine(W)` Regola l'altitudine del drone se rileva un'eccessiva disparità con quella del waypoint W;

`ControllaRotazione(W)` Orienta il drone in modo che punti verso W.

`ProiettaIntorno(W, r)` Restituisce il raggio (in pixel) della proiezione su M della sfera di raggio r centrata in W.

Input: Un waypoint W.

1 $d_{max} \leftarrow$ massima distanza rilevata dal LIDAR
2 $r \leftarrow$ distanza di sicurezza da rispettare
3 `ControllaAltitudine(`W`)`
4 `ControllaRotazione(`W`)`
5 $M \leftarrow$ mappa di profondità
6 $M \leftarrow$ `DilataMappa(`M`)`
7
8 $P_W \leftarrow$ `Proiezione(`W`)` `// proiezione di` W `sul piano di` M
9 $r_W \leftarrow$`CalcolaIntorno(`W, r`)`
10 $B(P_W) \leftarrow \{P \in M : d(P, W') \leq r_W\}$ `// pixel nell'intorno di` W'
11 $m \leftarrow$ valore minimo in $B(P_W)$
12 $m \leftarrow m \cdot d_{max}$
13 $\rho_W \leftarrow$ distanza di W dal drone
14 **if** $\rho_W - m > r$ *or* $m = d_{max}$ **then**
15 | **return** "Visibile"
16 **else if** $|\rho_W - m| < r$ **then**
17 | **return** "Irraggiungibile"
18 **else**
19 | **return** "Invisibile"

Algoritmo 2: `ControllaVisibilità(W)`.

4.4 TrovaWP

Questo algoritmo svolge la funzione fondamentale di trovare un waypoint intermedio W_{new} qualora l'obiettivo W non sia visibile, usando la mappa di profondità dilatata (Sezione 4.2.1).

L'algoritmo si baserà sulle seguenti considerazioni:

1. È conveniente rimanere nelle vicinanze dell'ostacolo: più ci si allontana, più sarà lungo il percorso complessivo.

2. L'aggiramento attraverso un cambio di altitudine comporta rischi aggiuntivi e quindi andrebbe evitato, a meno che non risulti estremamente vantaggioso.

4.4.1 Analisi della mappa di profondità

Innanzitutto si vuole isolare l'ostacolo dal resto della mappa di profondità. Per ottenere questo risultato si effettua un'operazione di *tresholding* sulla mappa stessa.

Siano $d_W, d_B \in [0,1]$, rispettivamente, la distanza drone-obiettivo e quella drone-ostacolo normalizzate rispetto al range del LIDAR. Scegliamo come valore di soglia

$$t = \min\left\{\frac{d_W + d_B}{2}, 0.9\right\}.$$

Il limite 0.9 viene arbitrariamente imposto per evitare che t possa assumere valori superiori a 1, cosa che può facilmente avvenire quando l'obiettivo e l'ostacolo sono lontani.

Tutti i pixel con valore superiore a t vengono posti a 1, gli altri a 0.

Come abbiamo già detto, siamo interessati ai pixel adiacenti agli ostacoli: applichiamo perciò una trasformata distanza (*distance transform*) all'immagine binaria, e consideriamo solo i pixel con valore 1.

Tra questi pixel, consideriamo quello che minimizza la seguente funzione di costo:

$$c(P) = \|W - P\| + k\left\|y_W - y_P\right\|$$

dove k è un parametro che ci permette di sfavorire a piacimento i candidati che comporterebbero un cambio di altitudine da parte del MAV. Indichiamo il pixel ottimo con $\bar{P}$.

4.4.2 Distanza del nuovo waypoint

Il pixel scelto, che indicheremo con $\bar{P}$, individua nello spazio una semiretta uscente dal MAV, in quanto le sue coordinate $x_{\bar{P}}, y_{\bar{P}}$ corrispondono a un azimuth e un'elevazione nel sistema descritto in Sezione 3.1.2. Bisogna ora calcolare un valore di distanza per determinare la posizione del waypoint intermedio su questa semiretta.

Potremmo usare, semplicemente, la distanza drone-ostacolo d_B così come viene rilevata dal sistema di controllo della visibilità. Ricordiamo che questo valore è il minimo tra quelli rilevati in un intorno di P_W, cioè il pixel su cui viene proiettato l'obiettivo. Tuttavia questa scelta si rivela spesso inefficace, poiché non tiene conto del bordo dell'ostacolo stesso.

Un esempio è riportato in Figura 4.4: il drone, situato in D, deve raggiungere l'obiettivo W ma rileva una barriera B. Entrambi i disegni sono visti dall'alto. Se usiamo la distanza d_B si otterrà il waypoint D', che come possiamo vedere (Figura 4.4 (a)) è a sua volta nella zona d'ombra $SZ(W,B)$.

Notiamo che la distanza rispetto al bordo dell'ostacolo porta a risultati migliori (Figura 4.4 (b)), e può essere recuperata dalla mappa di profondità.

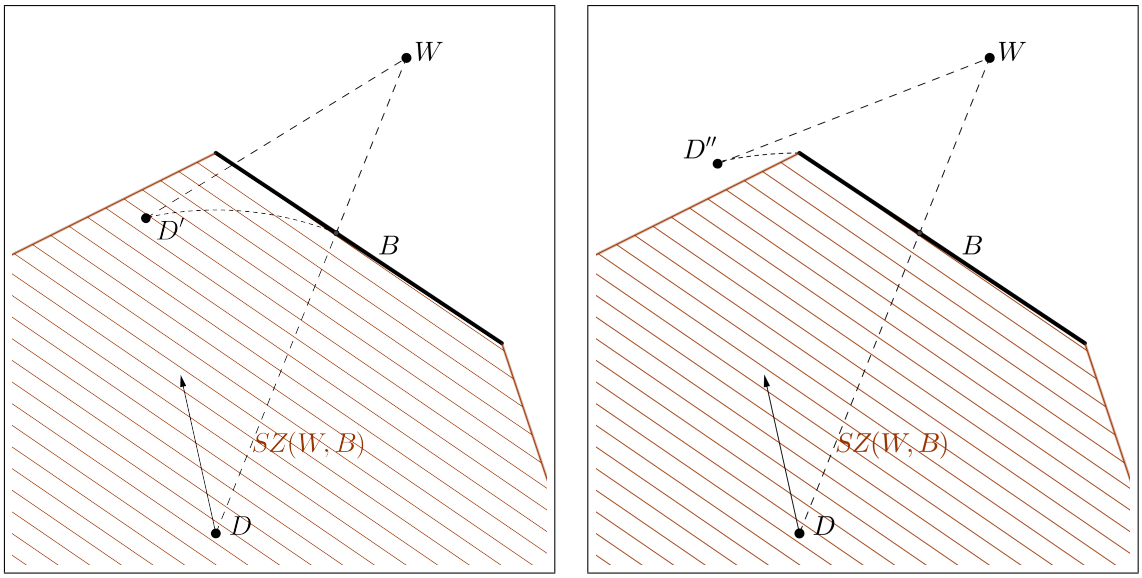
In particolare si effettua una scansione del segmento orientato che va da $\bar{P}$ a P_W e viene restituito il primo valore minore della soglia t.

Il valore così ottenuto viene infine aumentato per tener conto del raggio del MAV e corretto in base ai valori della mappa di profondità nell'intorno di P: in questo modo si può garantire che il nuovo waypoint è effettivamente raggiungibile, almeno al netto delle informazioni a disposizione.

4.4.3 *Wall following*

È ovvio che l'algoritmo appena descritto fallisce quando non ci sono pixel raggiungibili: ciò può essere dovuto a un ostacolo di grandi dimensioni o all'impostazione di margini di sicurezza molto elevati, che causano un "riempimento" della mappa di profondità nella fase di dilatazione.

In questo caso possiamo ricorrere a una variante dell'algoritmo di *wall following* proposto in [14], che qui riportiamo brevemente. Un esempio del comportamento di questo algoritmo è rappresentato in Figura 4.5.



(a) La distanza drone-ostacolo non è sufficiente per l'aggiramento.

(b) Uso della distanza rispetto al bordo dell'ostacolo.

Figura 4.4: Determinare la distanza del nuovo waypoint.

1. Il MAV effettua una scansione di ±60° rispetto all'obiettivo: l'aggiramento avverrà nel verso in cui viene percepita una maggior quantità di pixel "liberi".

2. Il MAV ruota nella direzione prescelta finché non rileva una buona percentuale di pixel liberi. L'articolo propone un angolo di visione navigabile di 60°.

3. Il MAV definisce un waypoint intermedio all'interno dello spazio navigabile appena rilevato, a una distanza fissata rispetto alla posizione attuale del drone (12 metri nell'articolo citato).

4. I tre passi precedenti vengono ripetuti finché non si torna a una situazione che può essere risolta dall'algoritmo `TrovaWP`.

4.4.4 Riepilogo

L'algoritmo deve ricevere in input la mappa di profondità M e la sua versione dilatata M'; le coordinate dell'obiettivo W e la sua proiezione su M, che indichiamo con W'; e

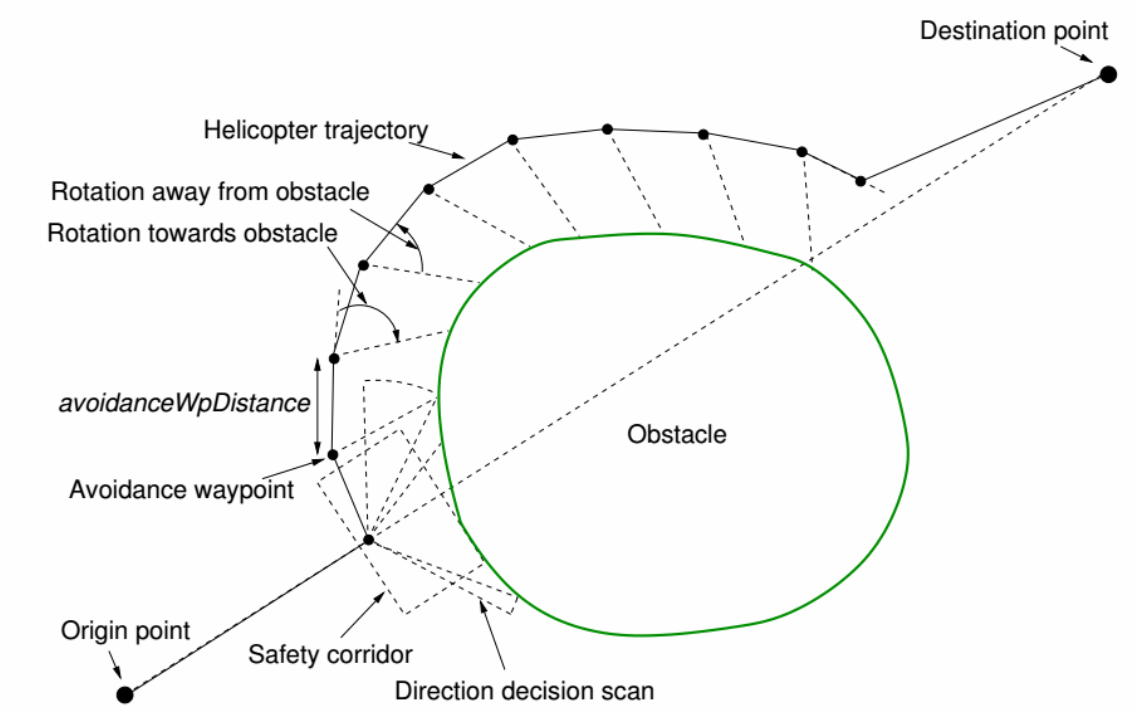


Figura 4.5: Comportamento dell'algoritmo di *wall following*. Fonte: [14],

la distanza d_B dell'ostacolo.

1. Binarizza M' con una soglia sul valore t (calcolato a partire da d_W e d_B). Risultato: un'immagine binaria M_t.

2. Calcola la trasformata distanza di M_t: $DT(M_t)$.

3. Considera i pixel di $DT(M_t)$ con valore 1 e scegli quello che minimizza la funzione di costo $c(P)$. Risultato: un pixel $\bar{P}$.

   Se $DT(M_t)$ è ovunque nulla, esci dall'algoritmo e fai ricorso al *wall following*.

4. Trova la distanza $d_{\partial B}$ tra il drone e il bordo dell'ostacolo.

5. Calcola il nuovo waypoint $W_{new} = \left(d_{\partial B}, \vartheta_{\bar{P}}, \varphi\bar{P}\right)$.

6. Valuta M per determinare se W_{new} è raggiungibile. Eventualmente correggi la distanza in base agli altri ostacoli rilevati.

7. Restituisci W_{new}.

4.5 Algoritmo di navigazione

Vediamo infine l'algoritmo completo per la navigazione locale (Algoritmo 3).

La funzione `VaiVerso(W)` fa spostare il drone in linea retta finché non si trova in W o finché non viene rilevato un'ostacolo che ostruisce la visibilità di W.

Notiamo che un obiettivo temporaneo viene automaticamente scartato se ritenuto invisibile dal drone. Quindi la pila non contiene mai più di due punti: l'obiettivo W e l'ultimo waypoint temporaneo calcolato.

Questa strategia non è ovviamente l'unica: si potrebbe ad esempio agire in maniera puramente ricorsiva, ponendo eventualmente un limite al numero di waypoint intermedi per evitare di riempire la pila all'infinito.

4.5.1 Convergenza

Abbiamo visto che l'algoritmo di scelta del waypoint intermedio W_{new} viene eseguito mentre il drone è orientato verso l'obiettivo W, e che il nuovo waypoint viene scelto tra i punti nel campo visivo del MAV. Se l'angolo di visione complessivo α è minore di 180° e se la distanza DW_{new} è sempre minore o uguale a DW, allora possiamo dimostrare che un qualsiasi intorno dell'obiettivo stesso verrà raggiunto in un tempo finito.

Per dimostrarlo vediamo cosa succede nel caso peggiore: scegliamo sempre il W_{new} che massimizza la distanza $DW_{new} + W_{new}W$.

Questo significa che W_{new} si trova sempre alla periferia del campo visivo, cioè ha un azimuth di $\pm\frac{\alpha}{2}$ rispetto al drone, e che la distanza DW_{new} è uguale a DW.

Abbiamo due sottocasi principali:

1. L'azimuth ha sempre lo stesso segno;
2. L'azimuth ha segni alterni.

La Figura 4.6 dimostra graficamente che in entrambi i casi si ottiene una successione di waypoint $W_i|_{i\in\mathbb{N}}$ che tende a W: ciò equivale a dire che, per qualsiasi intorno $B_\epsilon(W)$, esiste un W_i che giace in $B_\epsilon(W)$.

Input: Un waypoint T (target).

```
1  P ← Pila()
2  Push(P, T)
3  while S ≠ ∅ do
4      W ← Pop(S)
5      ControllaAltitudine(W)
6      ControllaRotazione(W)
7      switch ControllaVisibilità(W) do
8          case "Visibile"
9              VaiVerso(W)
10         case "Invisibile"
11             if W = T then
12                 Push (P, T)
13                 W_new ← TrovaWaypoint(T)
14                 if W_new = NULL then
15                     Exit("Impossibile aggirare l'ostacolo.")
16                 else
17                     Push(P, W_new)
18         case "Irraggiungibile"
19             if W = T then
20                 Exit("Impossibile raggiungere l'obiettivo.")
21             else
22                 Push(P, T)
23     endsw
24 end
```

Algoritmo 3: `NavigazioneLocale(T)`.

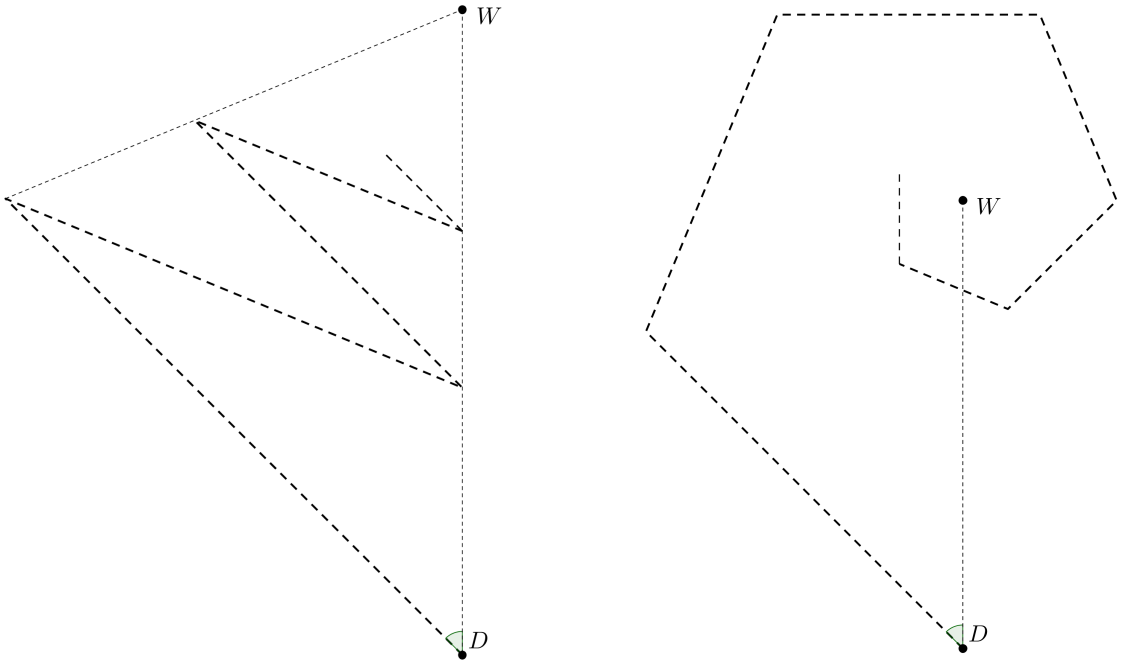


Figura 4.6: I casi peggiori dell'algoritmo di navigazione. In questo esempio $\alpha = 90°$.

Capitolo 5

Implementazione in ambiente simulato

5.1 Introduzione

L'algoritmo presentato nella sezione precedente è stato inizialmente testato tramite simulazioni al computer. Questa decisione deriva dalla necessità di avere un rapido riscontro sulla qualità dell'implementazione, soprattutto nella fase prototipale, nonché dalla volontà di minimizzare i rischi prima di iniziare la fase di prova su un MAV.

5.2 Piattaforma e strumenti di sviluppo

La scelta è ricaduta sulla piattaforma di simulazione robotica V-REP [19]. Il principale vantaggio di questo software è l'elevato grado di flessibilità: è possibile controllare un qualsiasi oggetto della scena grazie a *script* interni, oppure interfacciarsi dall'esterno mediante una API (Application Programming Interface).

Per avvalersi di quest'ultima capacità occorre invocare all'inizio della simulazione la funzione `simExtRemoteApiStart()`. La funzione prende come parametro il numero della porta su cui il server resterà in ascolto fino al termine della simulazione.

Gli script interni sfruttano il linguaggio Lua, mentre i client per la API remota possono essere scritti in molti linguaggi tra cui C/C++, Java, Python e Matlab.

Infine, è possibile interfacciare V-REP con il framework Robot Operating System mediante un plug-in, consentendo la comunicazione attraverso i meccanismi publisher/subscriber tipici di ROS.

L'algoritmo di navigazione è stato programmato in Python 3.5 [1] per massimizzare rapidità di sviluppo e leggibilità, caratteristiche fondamentali durante la fase di ricerca e prototipazione.

Il principale difetto del linguaggio, vale a dire la bassa efficienza nelle applicazioni numeriche, è facilmente superabile facendo ricorso al pacchetto NumPy [2], che offre pieno supporto a strutture dati e algoritmi multidimensionali con funzionalità comparabili a quelle presenti in Matlab.

Per l'elaborazione efficiente delle immagini si ricorre alla libreria OpenCV [3]. Oltre alle buone prestazioni, OpenCV garantisce il supporto verso molti linguaggi di programmazione, il che semplificherà notevolmente un'eventuale migrazione del codice.

5.3 Dettagli dell'implementazione

5.3.1 Controllori

Il controllo della rotazione e dell'altitudine avviene tramite due controllori PID. L'interfaccia del controllore è elementare:

- `control()` prende come parametro un vettore di errore e restituisce un vettore di controllo. Quest'ultimo dovrà essere usato per modificare opportunamente lo stato del drone.
- `reset()` azzera la memoria del controllore.

L'implementazione approssima l'integrale dell'errore con un accumulatore; per approssimare la derivata si usa invece la differenza tra l'errore attuale e quello precedentemente passato alla funzione `control()`.

I parametri di guadagno sono stati calibrati empiricamente in modo da limitare la sovraelongazione e le oscillazioni, mantenendo comunque tempi di salita e assestamento accettabili (nell'ordine dei 5). Per evitare il cosiddetto *integral windup* (cioè un'eccessiva correzione dovuta al termine integrale), viene posto un limite al valore che l'accumulatore interno può raggiungere.

[1] Python Software Foundation. `https://www.python.org/`
[2] http://www.numpy.org/
[3] Intel. `http://opencv.org/`

5.3.2 Uso dell'API

L'API di V-REP per Python è pensata secondo un paradigma procedurale. Il file `vrep.py` definisce numerose funzioni, che hanno solitamente bisogno di questi parametri:

- Un `clientID` che identifica la comunicazione;
- Un riferimento (handle) all'oggetto su cui effettuare l'operazione;
- Se necessari, uno o più parametri necessari all'operazione.
- L'ultimo parametro rappresenta sempre la modalità della richiesta. Infatti V-REP supporta chiamate non bloccanti, ma abbiamo preferito non servirci di questa funzionalità e usare unicamente operazioni bloccanti: lo script aspetterà sempre di ricevere una risposta prima di proseguire.

Per ottenere un `clientID` è necessario invocare la funzione `vrep.simxStart()`, alla quale bisogna passare l'host e la porta su cui V-REP è in ascolto.

Se si conosce il nome di un oggetto all'interno della scena, è possibile recuperarne l'handle tramite la funzione `vrep.simxGetObjectHandle(clientID, objectName, operationMode)`.

Questa API costringe lo sviluppatore a realizzare codice verboso e ripetitivo: un approccio orientato agli oggetti risulterebbe molto più efficace (specie se si considera che molti concetti propri del paradigma OOP hanno origine in Simula 67, un linguaggio di simulazione).

Per soddisfare questo obiettivo abbiamo realizzato le seguenti classi:

`VRepConnection` Contiene la logica per gestire la connessione al server e quella di recupero degli oggetti simulati.

`VRepObject` Incapsula tutte le funzioni che leggono o modificano lo stato di un oggetto, attraverso metodi *getter* e *setter*.

I waypoint sono stati implementati sotto forma di oggetti V-REP sferici e che non risentono delle leggi fisiche (gravità, collisioni).

`VRepDepthBuffer` Eredita da `VRepObject` e rappresenta un sensore di profondità di V-REP: implementa un ulteriore metodo `get_depth_buffer()` che restituisce la mappa di profondità percepita dal sensore.

`Drone` Contiene la logica relativa al drone: ad esempio implementa gli algoritmi `VaiVerso()`, `ControllaAltitudine()`, `ControllaRotazione()`, `ControllaVisibilità()`. Abbiamo inoltre definito un metodo `stabilize()` che controlla la velocità lineare e angolare del drone, sospendendo l'esecuzione dell'algoritmo finché non viene rilevata una situazione di volo stazionario.

5.3.3 Sistema di riferimento

Siccome ci troviamo in un ambiente simulato, usiamo un sistema di riferimento cartesiano con origine nel centro della scena. Questo non ci impedirà di usare l'algoritmo in contesti reali: basterà usare le coordinate ECEF ed effettuare le opportune conversioni da e verso il sistema geodetico (cfr. Sezione 3.1.1).

5.4 Esempi di simulazione

Il primo scenario è molto semplice e contiene solo due barriere (Figura 5.1). Ci aspettiamo che il drone raggiunga l'obiettivo volando nello spazio compreso tra un ostacolo e l'altro.

Una volta orientato verso l'obiettivo, il drone riesce a stabilire un waypoint intermedio: le immagini in Figura 5.2 mostrano il processo decisionale. La linea nera in Figura 5.2 (d) indica i pixel presi in considerazione durante la scelta del nuovo waypoint; il risultato è il punto W_{new}.

Come previsto, il MAV porta a termine la missione come avevamo previsto. Il tempo di ripianificazione è nell'ordine del decimo di secondo.

Il secondo scenario (Figura 5.4) simula invece un ambiente naturale, con un terreno ondulato e cosparso di alberi. Stavolta il nostro algoritmo genera quattro waypoint intermedi prima di raggiungere l'obiettivo. Questa configurazione ci permette di mettere in mostra un'altra proprietà del sistema: esso è in grado di seguire automaticamente l'inclinazione del terreno (Figura 5.5).

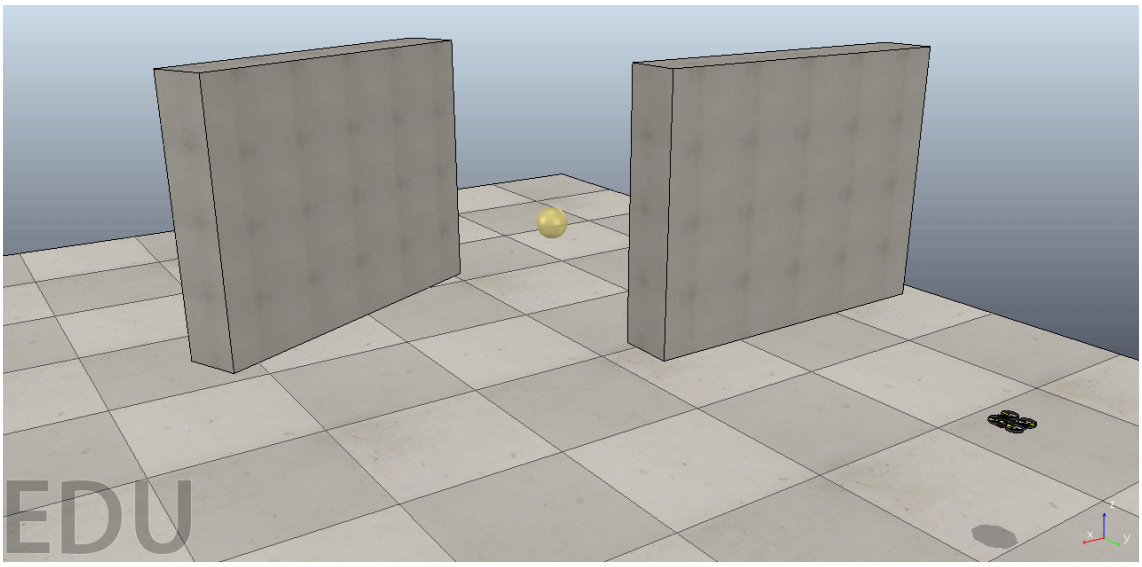
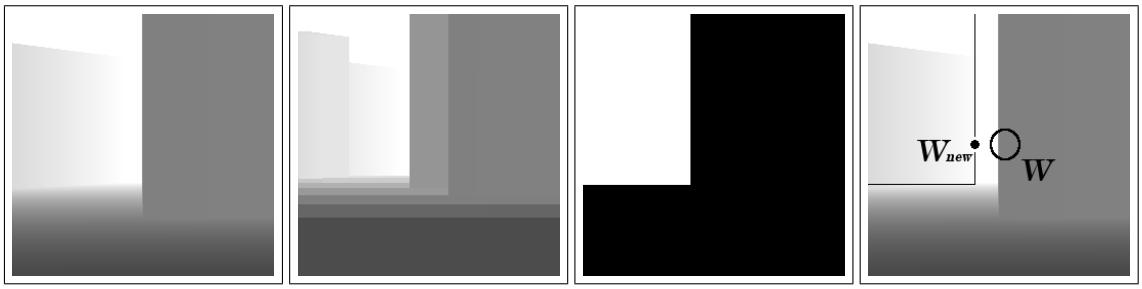


Figura 5.1: Panoramica del primo scenario di simulazione. L'obiettivo è rappresentato dalla sfera gialla.



(a) Mappa di profondità. (b) Dilatazione. (c) Binarizzazione. (d) Scelta del pixel corrispondente al nuovo waypoint.

Figura 5.2: Esecuzione dell'algoritmo `TrovaWP` nel primo scenario simulato.

5.5 Valutazione sperimentale

Il modello di quadrirotore fornito da V-REP è stato equipaggiato con un sensore che simula uno scanner laser 3D. La configurazione del sensore stabilisce un range massimo di 10 metri e una copertura angolare di 90° sia in orizzontale che in verticale.

I risultati descritti nella Sezione 5.4 sono stati ottenuti con un sensore configurato

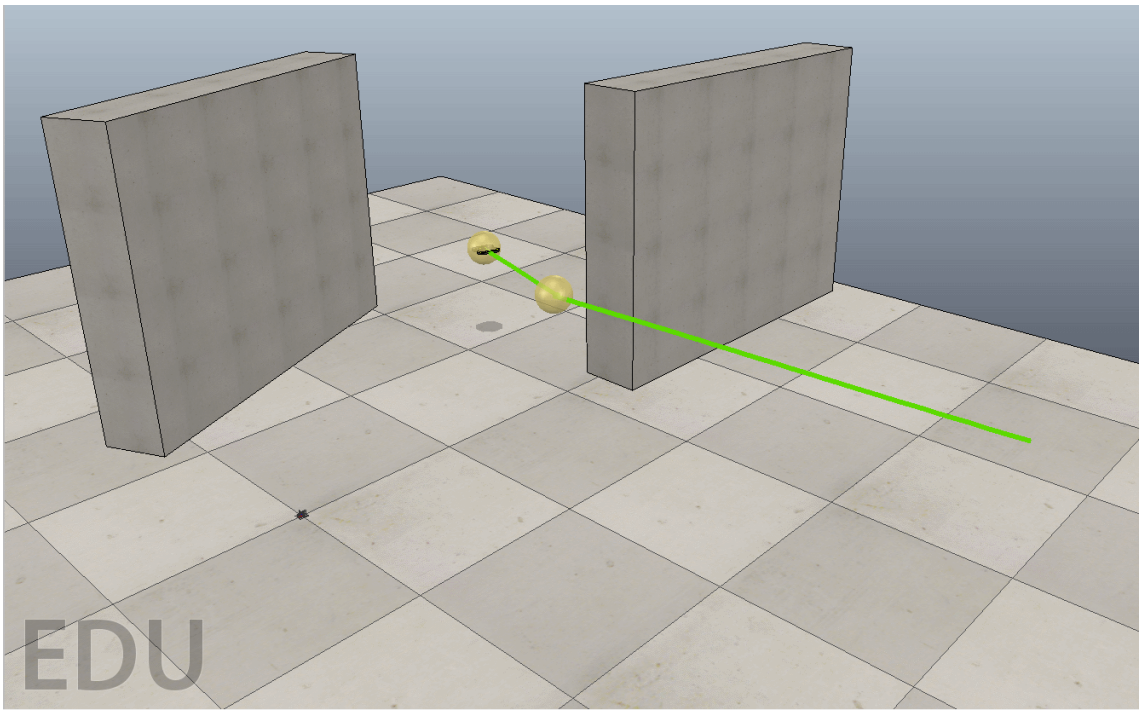


Figura 5.3: Percorso scelto dal drone per raggiungere l'obiettivo.

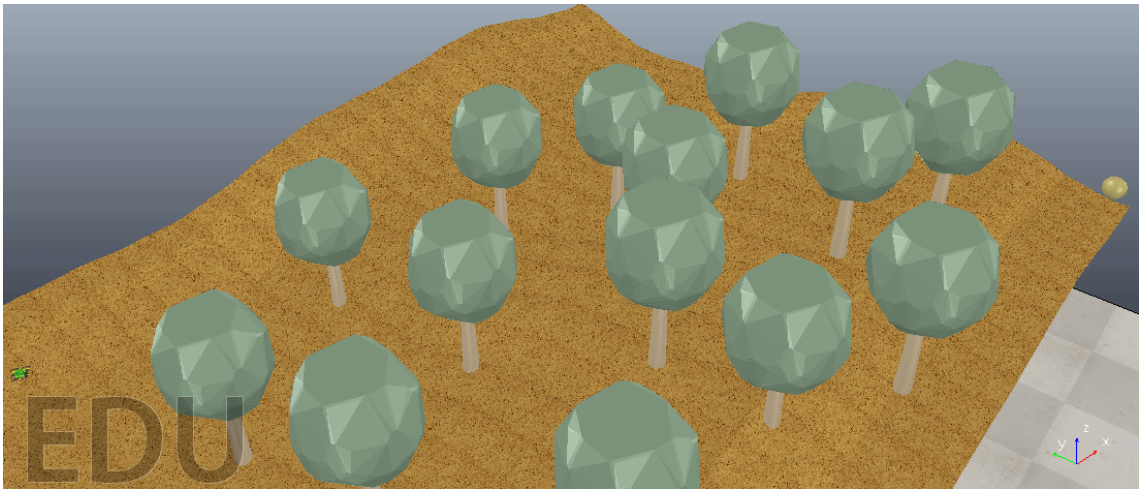


Figura 5.4: Panoramica della seconda simulazione.

per restituire mappe di profondità da 256x256 pixel.

Sono stati effettuati numerosi test con risoluzioni via via minori, per verificare la robustezza dell'algoritmo rispetto a sensori meno dettagliati.

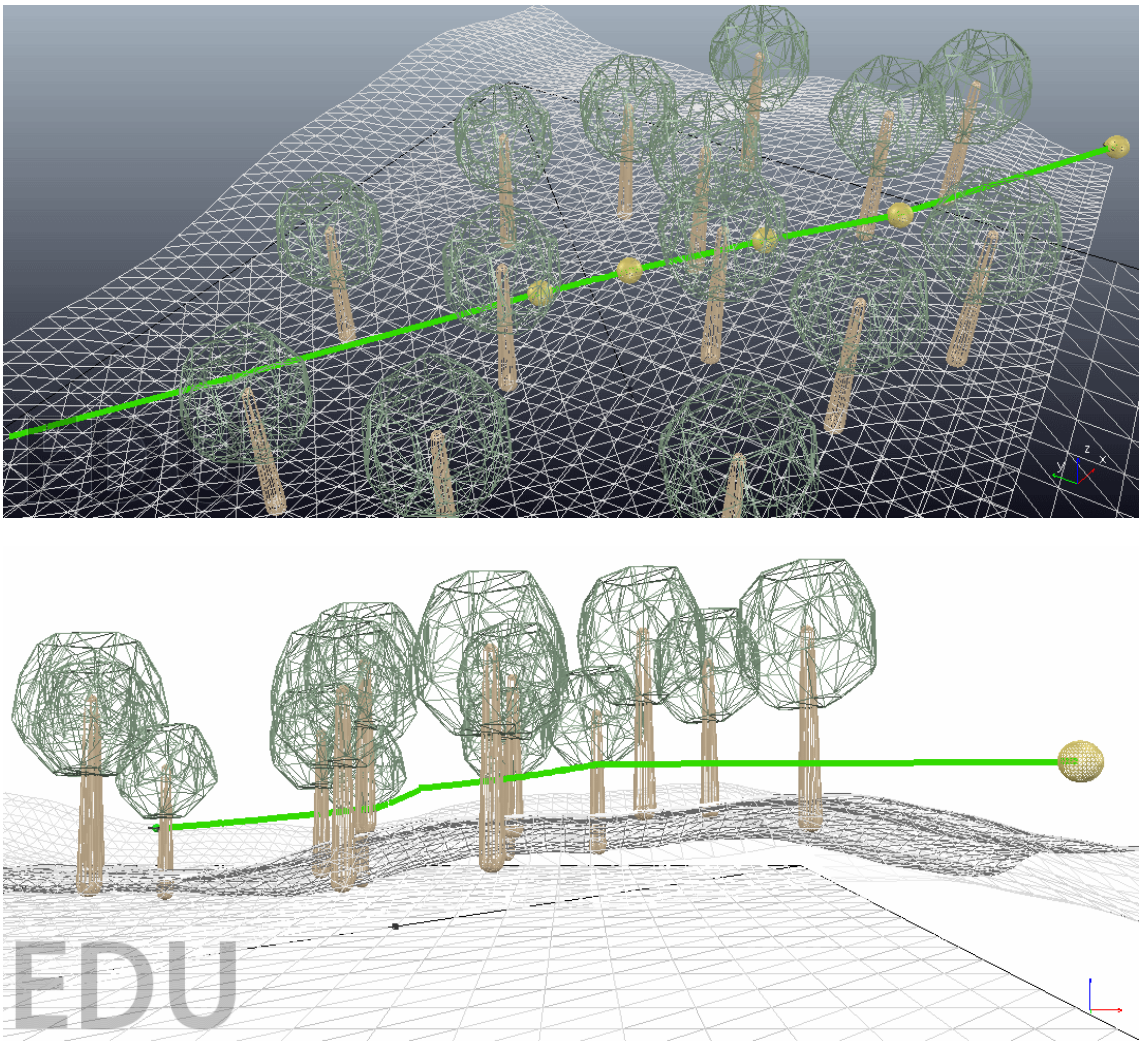


Figura 5.5: Percorso scelto dal drone. Si può notare la capacità dell'algoritmo di adattarsi alle irregolarità del terreno.

Il primo scenario è stato risolto con modalità identiche a quelle già descritte, anche dopo un'estrema diminuzione della risoluzione del LIDAR (16x16). Questo risultato era prevedibile, visto che la geometria della scena è piuttosto regolare e quindi non risente particolarmente del cambio di risoluzione.

L'algoritmo è riuscito a risolvere anche il secondo problema di navigazione, ma in questo caso abbiamo notato una maggiore variabilità dei risultati che merita un approfondimento. La Tabella 5.1 riporta alcune metriche usate per stimare la qualità

#	Risoluzione	Waypoint		Distanza (m)		Tempo
		Raggiunti	Scartati	Assoluta (m)	Normalizzata	
1	256	2	0	24.16	1.04	**00:58.6**
2	256	2	1	23.83	1.03	01:12.0
3	256	3	0	**23.43**	**1.01**	01:22.5
4	128	2	1	23.62	1.02	01:13.0
5	128	4	1	24.08	1.03	01:32.0
6	128	3	1	23.54	1.01	01:12.1
7	64	4	5	**24.99**	**1.07**	**02:09.6**
8	64	3	2	24.19	1.04	01:20.7
9	64	3	1	23.68	1.02	01:20.3
10	32	4	1	23.73	1.02	01:25.8
11	32	3	2	24.05	1.03	01:46.2
12	32	3	2	23.79	1.02	01:18.3

Tabella 5.1: Risultati sperimentali per il secondo scenario.

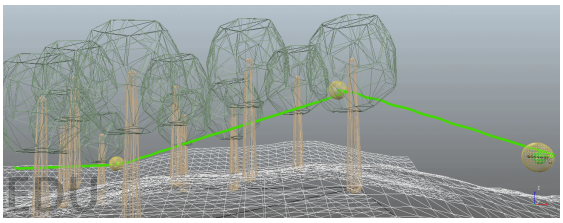
delle soluzioni: il tempo impiegato, la distanza totale percorsa dal drone, il numero di waypoint attraversati e di quelli inizialmente scelti ma successivamente scartati.

La lunghezza del percorso è stata inoltre normalizzata rispetto alla distanza in linea d'aria tra il punto di partenza e l'obiettivo, pari a 23.27 metri.

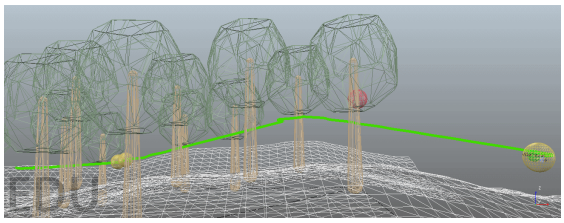
5.5.1 Analisi dei risultati

Notiamo che l'algoritmo riesce a completare il percorso senza distaccarsi eccessivamente dal percorso ottimo: il rapporto con la distanza in linea d'aria non supera mai la soglia di 1.10.

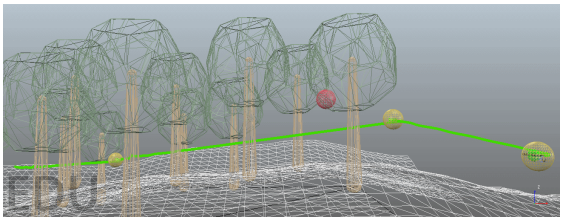
Il principale limite dell'algoritmo è la creazione di waypoint che poi si scoprono non raggiungibili. Questo fenomeno porta a un aumento del tempo di percorrenza complessivo e si verifica principalmente quando l'ostacolo è molto lontano dal drone: in questi casi l'algoritmo può generare un waypoint nelle vicinanze di un ostacolo non



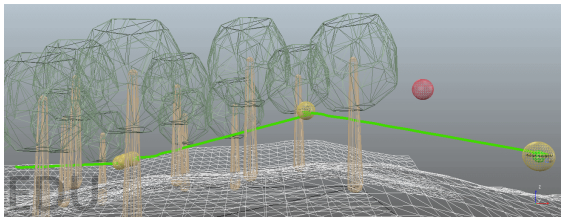
(a) Simulazione 1.



(b) Simulazione 4.



(c) Simulazione 9.



(d) Simulazione 10.

Figura 5.6: Le soluzioni di alcuni dei test effettuati.

ancora percepito. L'algoritmo di dilatazione della mappa di profondità può a sua volta portare all'eliminazione di un waypoint per eccesso di prudenza.

Infine, bisogna considerare che la velocità del drone è stata limitata a 0.6 m/s per garantire la frenata in tempi e spazi ridottissimi. Questa scelta ha ovviamente influito sui tempi di percorrenza totali.

Capitolo 6

Conclusioni e sviluppi futuri

Abbiamo presentato il problema della navigazione e guida autonoma di un multirotore come un insieme di sottoproblemi che possono essere organizzati secondo un'architettura stratificata.

Il lavoro di tesi si è concentrato sul problema della navigazione locale, esponendo le diverse tecniche note in letteratura e proponendo un nuovo approccio mapless e reattivo, capace di pilotare con successo il drone verso il waypoint di destinazione basandosi su una quantità limitata di informazioni relative all'ambiente.

L'algoritmo è stato implementato e testato mediante diverse simulazioni, che hanno mostrato una buona qualità delle soluzioni e una robustezza accettabile rispetto alla diminuzione della risoluzione dei sensori di bordo. Sarà opportuno eseguire ulteriori test per valutare il comportamento dell'algoritmo al variare di altre caratteristiche del drone, quali ad esempio l'angolo di visione del LIDAR o il suo *range* massimo.

Un altro aspetto che finora è stato trascurato è quello della precisione del posizionamento GPS. Il sistema dovrebbe essere in grado di valutare la qualità dei dati di posizione e regolare di conseguenza i margini di sicurezza, o nel peggiore dei casi sospendere la missione e mantenere il drone in volo stazionario finché la qualità del *fix* non migliora.

L'implementazione su un MAV reale richiederà i seguenti passaggi:

- Realizzare un plug-in per Mission Planner che raccolga i dati LIDAR e li renda disponibili all'esterno;

- Implementare una nuova classe `Drone` con la stessa interfaccia dell'attuale ma che si interfacci con Mission Planner anziché V-REP;
- Realizzare una classe `Waypoint` che incapsuli le informazioni da passare a Mission Planner e i metodi di conversione tra i sistemi di coordinate.

Infine, un sistema più avanzato potrebbe immagazzinare informazioni sugli ostacoli incontrati nel corso della missione: queste informazioni potranno essere utilizzate da un *global path planner* per pianificare meglio le successive missioni nella stessa località.

Glossario dei termini e delle abbreviazioni

API *Application Programming Interface*. Insieme di procedure rese disponibili da un software e che permettono a programmi esterni di comunicare facilmente con il software stesso.

Fix GPS La posizione calcolata da un sensore in seguito all'osservazione dei satelliti GPS. L'accuratezza del fix può variare a seconda di diversi parametri (numero e posizione dei satelliti rilevati, qualità dell'antenna, condizioni atmosferiche).

GCS *Ground Control Station*. Insieme di strumenti hardware e software con cui è possibile monitorare e guidare il drone da terra. Tipicamente una stazione di terra sarà formata da un radiocomando e da dispositivi per il monitoraggio dello stato dell'UAV.

MAV *Micro Aerial Vehicle*. Termine coniato dal DARPA e inizialmente riservato agli UAV con dimensioni estremamente ridotte (< 15 cm.); tuttavia in letteratura può indicare qualunque drone di dimensioni inferiori ai 5 m., cioè quelli più usati nella ricerca accademica e nelle applicazioni civili (in contrapposizione a quelli più grandi, realizzati a fini prevalentemente militari).

OOP *Object-Oriented Programming*.

Stato (dell'UAV) Insieme dei seguenti valori:

- Coordinate GPS
- Valori di rollio, beccheggio, imbardata
- Velocità

Trasformata Distanza operazione su un'immagine binaria che associa ad ogni punto di valore "1" la minima distanza dallo sfondo (cioè dai pixel di valore "0"). Tipicamente si ricorre alla metrica L_1, anche detta *distanza di Manhattan*, in modo che il risultato sia a valori interi.

UAV *Unmanned Aerial Vehicle*, vale a dire un velivolo privo di un equipaggio umano a bordo. Il termine è a tutti gli effetti un sinonimo di *drone*, che però ha assunto una connotazione piuttosto negativa presso l'opinione pubblica.

Waypoint Un punto intermedio di un piano di volo. Include sempre una posizione GPS ed un'altitudine desiderata; a seconda del tipo di missione e delle caratteristiche del drone (ad esempio: raccolta dati con un sensore fisso) può anche includere un orientamento desiderato. I sistemi di navigazione più sofisticati possono dotare i w. di valori semantici e di una precisione desiderata: in tal caso il drone deve raggiungere un intorno del w. la cui dimensione dipenderà dai suddetti fattori.

Bibliografia

[1] Amazon. Amazon Prime Air, 2014.
http://www.amazon.com/b?node=8037720011.

[2] Stefania Amici, Matteo Turci, Salvatore Giammanco, Letizia Spampinato, and Fabrizio Giulietti. UAV Thermal Infrared Remote Sensing of an Italian Mud Volcano. *Advances in Remote Sensing*, 2:358–364, 2013. ISSN 2169-267X. doi:10.4236/ars.2013.24038.
http://file.scirp.org/pdf/ARS_2013122610151089.pdf.

[3] David Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. Concorde TSP Solver, 2003.
http://www.math.uwaterloo.ca/tsp/concorde/index.html.

[4] Abraham Bachrach, Ruijie He, and Nicholas Roy. Autonomous Flight in Unknown Indoor Environments. *International Journal of Micro Air Vehicles*, 1(4):217–228, 2010. ISSN 1756-8293. doi:10.1260/175682909790291492.

[5] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. 1996. ISBN 0471958697. doi:ISBN: 978-0-471-95869-7.

[6] Guowei Cai, Jorge Dias, and Lakmal Seneviratne. A Survey of Small-Scale Unmanned Aerial Vehicles: Recent Advances and Future Development Trends. *Unmanned Systems*, 02(02):175–199, 2014. ISSN 2301-3850. doi:10.1142/S2301385014300017.
http://www.worldscientific.com/doi/abs/10.1142/S2301385014300017.

[7] Davide Di Ruscio, Ivano Malavolta, and Patrizio Pelliccione. Engineering a platform for mission planning of autonomous and resilient quadrotors. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8166 LNCS:33–47, 2013. ISSN 03029743. doi:10.1007/978-3-642-40894-6_3.

[8] Paolo Fogliaroni and Eliseo Clementini. Modeling Visibility in 3D Space : a Qualitative Frame of Reference. *9th International 3DGeoInfo 2014 - Lecture Note in Geoinformation and Cartography*, pages 1–19, 2014. doi:10.1007/978-3-319-12181-9_15.
https://www.researchgate.net/publication/266681207_Modeling_Visibility_in_3D_Space_a_Qualitative_Frame_of_Reference.

[9] C. Goerzen, Z. Kong, and B. Mettler. *A survey of motion planning algorithms from the perspective of autonomous UAV guidance*, volume 57. 2010. ISBN 0921-0296. doi:10.1007/s10846-009-9383-1.

[10] Michael Hahsler and Kurt Hornik. TSP – Infrastructure for the Traveling Salesperson Problem. *Journal Of Statistical Software*, 1(1):1–21, 2007. ISSN 15487660. doi:10.1002/wics.10.
http://www.jstatsoft.org/v23/i02.

[11] Matthias Heutger and Markus Kuckelhaus. Unmanned Aerial Vehicles in Logistics. Research, DHL Customer Solutions & Innovation, 2014.
https://www.dhl.de/content/dam/dhlde/images/ueber_uns/content/dhl_trend_report_uav.pdf.

[12] Stefan Hrabar. Reactive obstacle avoidance for rotorcraft UAVs. *IEEE International Conference on Intelligent Robots and Systems*, (August):4967–4974, 2011. ISSN 2153-0858. doi:10.1109/IROS.2011.6048312.

[13] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, mar 2012. ISSN 15564959. doi:10.1002/rob.20414.
http://doi.wiley.com/10.1002/rob.20414.

[14] Torsten Merz and Farid Kendoul. Beyond visual range obstacle avoidance and infrastructure inspection by an autonomous helicopter. *IEEE International Conference on Intelligent Robots and Systems*, (August 2016):4953–4960, 2011. ISSN 2153-0858. doi:10.1109/IROS.2011.6048249.

[15] Matthias Nieuwenhuisen and Sven Behnke. Hierarchical Planning with 3D Local Multiresolution Obstacle Avoidance for Micro Aerial Vehicles. In *Joint 45th International Symposium on Robotics (ISR) and 8th German Conference on Robotics (ROBOTIK)*. University of Bonn, Germany, 2014.

[16] Matthias Nieuwenhuisen, David Droeschel, Marius Beul, and Sven Behnke. Obstacle detection and navigation planning for autonomous micro aerial vehicles. *2014 Int. Conf. Unmanned Aircr. Syst.*, (May):1040–1047, 2014. doi:10.1109/ICUAS.2014.6842355.
`http://www.ais.uni-bonn.de/papers/ICUAS_2014_Nieuwenhuisen.pdf$\delimiter"026E3B2$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6842355`.

[17] Petter Ögren. Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. *AIAA Guidance, Navigation, and Control Conference*, (August):1–8, 2012. ISSN 1070-9932. doi:10.2514/6.2012-4458.
`http://arc.aiaa.org/doi/abs/10.2514/6.2012-4458`.

[18] Ulises Orozco-Rosas, Oscar Montiel, and Roberto Sepúlveda. Pseudo-bacterial potential field based path planner for autonomous mobile robot navigation. *International Journal of Advanced Robotic Systems*, 12, 2015. ISSN 17298814. doi:10.5772/60715.

[19] Eric Rohmer, Surya P N Singh, and Marc Freese. V-REP : a Versatile and Scalable Robot Simulation Framework. In *Proceedings of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.
`http://coppeliarobotics.com/v-repIros2013.pdf`.

[20] A Stenger, B Fernando, and M Heni. Autonomous Mission Planning for Uavs : a Cognitive Approach. In *Deutscher Luft- und Raumfahrtkongress*, pages 1–10, 2012.

http://publikationen.dglr.de/?tx_dglrpublications_pi1[document_id]=281398http://www.dglr.de/publikationen/2013/281398.pdf.

[21] Lance F Tammero and Michael H Dickinson. The influence of visual landscape on the free flight behavior of the fruit fly Drosophila melanogaster. *The Journal of experimental biology*, 205(Pt 3):327–343, 2002. ISSN 0022-0949.

[22] George Vachtsevanos, Liang Tang, Graham Drozeski, and Luis Gutierrez. From mission planning to flight control of unmanned aerial vehicles: Strategies and implementation tools. *Annual Reviews in Control*, 29(1):101–115, 2005. ISSN 13675788. doi:10.1016/j.arcontrol.2004.11.002.

Licenze

Figura 3.2 derivata da opere della società Freepik, rese disponibili sul sito Flaticon (http://www.flaticon.com/).