POLITECNICO DI TORINO

Facoltà di Ingegneria

Laurea Magistrale in Ingegneria Aerospaziale

Tesi Magistrale

Mappatura digitale del terreno in tempo reale tramite LIDAR per un sistema di monitoraggio incendi

Relatore
Prof. Manuela Battipede

Correlatore
Dott. Francesco De Vivo

Candidato
Luigi Lombardi Vallauri

Dicembre 2019

Ringraziamenti

Grazie a chi mi ha aiutato nella stesura di questa tesi, ma soprattutto grazie a chi mi ha tenuto compagnia e supportato nella strada che vi ha condotto. Grazie i miei genitori che oltre al chiaro supporto economico mi hanno permesso di vivere in modo non stressante la carriera universitaria, essendo entusiasti dei vari progetti che l'hanno rallentata. Grazie ai miei amici dell'università che hanno reso questi anni il periodo più divertente della mia vita e mi hanno fatto scoprire il tipo di persona che vorrei essere. Grazie ad Alessia con la sua dolce presenza e il suo spirito d'iniziativa così adatti ad arricchire la mia vita e ad alleggerire un pesante percorso di studi. Un ben meritato grazie anche a tutti gli amici e parenti che non hanno dubitato per un secondo della mia capacità di raggiungere questo traguardo. Infine un grazie a tutto ciò che per me Torino rappresenta, città che mi ha ospitato in questi anni ampliando i miei orizzonti. In una vita che parte adesso in direzione ignota, so che potrò contare su tutti voi.

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Italiano

Il numero e l'intensità degli incendi stanno aumentando a causa del riscaldamento globale. Soluzioni efficaci per limitarne l'impatto tardano ad emergere. Attualmente il monitoraggio aereo è nella quasi totalità attuato attraverso mezzi grandi con pilota a bordo, con conseguenti lunghi tempi di dispiegamento e alti costi. Solo ultimamente si è cominciato a parlare di droni autonomi per la sorveglianza di incendi. Qui propongo un software che, attraverso un sensore LIDAR, fornisca una mappa digitale del territorio sul quale un drone si trova a volare, senza conoscenza pregressa dell'ambiente. L'utilità della mappa è consentire al sistema di navigazione di riconoscere ostacoli e pendenze sul terreno. La tesi contiene anche una breve guida allo sviluppo e all'implementazione dei codici su una Raspberry Pi3B+. Inoltre, allo scopo di verificare il funzionamento del sistema di navigazione senza mettere a rischio l'incolumità del drone, ho sviluppato un algoritmo per simulare un sensore LIDAR a bordo di un drone virtuale *Ardupilot:Copter SITL* che vola su una mappa DEM. I procedimenti ottenuti con il simulatore possono essere trasposti ad un caso reale.
Questa trattazione è da considerarsi in parallelo alla tesi sviluppata da Sara Senzacqua: *Politecnico di Torino, 2019, "Pianificazione della traiettoria di un APR tramite mappatura digitale del terreno per il monitoraggio incendi"*.

English

The number and intensity of wildfires are rising due to global warming. Effective solutions to limit their impact are not readily available. At the moment most of aerial monitoring is done through large manned vehicles, which require a long time to be deployed and high costs. It is only recently that we hear about wildfire surveillance drones. I'm here proposing software which, using a LIDAR sensor, provides a digital map of the terrain on which the drone is flying, without previously knowing the environment. The map is useful to allow the navigation system to recognise obstacles and terrain slopes. The thesis also contains a short guide to developing and implementing the codes on a Raspberry Pi3B+. Furthermore, in order to verify the behaviour of the navigation system without endangering the drone, I developed an algorithm to simulate a LIDAR sensor on board of a virtual *Ardupilot:Copter SITL* drone flying on a DEM map. The steps obtained with the simulation can be transposed to a real case.
This dissertation is to be considered together with the thesis developed by Sara Senzacqua:

Politecnico di Torino, 2019, "UAV obstacle avoidance based on LIDAR mapping for real time wildfire monitoring.".

Capitolo 1

Introduzione

Gli *aeromobili a pilotaggio remoto*, comunemente chiamati *droni*, sono apparecchi volanti senza pilota a bordo. Essi possono essere controllati da un computer di bordo, da un pilota remoto, o da un ibrido dei due.
Mentre il loro uso si è consolidato negli anni in ambito militare, è recente lo sviluppo per applicazioni civili. Un drone è adatto a svolgere un compito che per un veicolo con pilota a bordo sarebbe pericoloso, lento, costoso, noioso, sporco o, a volte, totalmente impossibile.
La tesi è nata dalla necessità di creare un sistema nuovo in grado di monitorare l'estensione e lo sviluppo in tempo reale degli incendi.
Complice il riscaldamento globale, negli ultimi tempi abbiamo sentito di fuochi devastanti in luoghi inaspettati come la Siberia e i Paesi scandinavi. Anche in Italia il fenomeno è tutt'altro che in diminuzione [1]: considerando una media dei dati dal 2008 al 2019 l'Italia è al terzo posto in Europa per area bruciata e seconda per numero di incendi [1]. Si tratta di un problema le cui prospettive sembrano aggravarsi e per cui le soluzioni efficaci stanno tardando ad arrivare.

Un grande problema dell'attuale sistema di vigilanza del fuoco è che gli addetti si trovano davanti ad un fronte di fiamme e hanno grande difficoltà a sapere su quali aree concentrare le proprie forze: non sanno l'attuale estensione dell'incendio e quindi la prossimità del fuoco con abitazioni o aree sensibili, non sanno con certezza la direzione del vento e il conseguente spostamento dell'area interessata. Queste incognite possono essere rivelate usando elicotteri o aeroplani, forze che richiedono però molto tempo per essere dispiegate e il quale costo è sempre elevato.

Alcuni dipartimenti per la salvaguardia forestale statunitensi stanno cominciando a usare droni a pilotaggio remoto per monitorare gli incendi, con risultati più che soddisfacenti [2], mentre l'Universidad Carlos III de Madrid sta collaborando con Telefónica R+D+i per sviluppare droni autonomi per lo stesso scopo [3].

L'obbiettivo della tesi è fare in modo che un drone, navigando sopra un qualsiasi luogo geografico di topologia ignota, possa, in tempo reale, sviluppare una mappa tridimensionale del terreno sottostante. Il sistema di navigazione utilizzerà le informazioni per non scontrarsi con ostacoli presenti a terra.

Dispiegando un drone che non ha bisogno di controllo umano, si potrebbe garantire una sorveglianza dell'incendio prima di portare la quadra in prossimità di esso, con una serie di vantaggi: guadagnare tempo, migliorare in anticipo la strategia di approccio, evitare di mettere a rischio un pilota remoto nelle vicinanze dell'incendio.

Abbiamo deciso di usare un sensore LIDAR per le rilevazioni del terreno a causa della maggiore precisione rispetto ad un sensore RADAR, oltre al recente boom di applicazione della tecnologia LIDAR nel settore della guida autonoma [4].
Abbiamo scelto un sensore Leddar Vu8 con campo visivo 48° x 0.3° da montare sul drone puntato verso il terreno.

Per ricavare i dati del terreno, oltre ai dati di distanza forniti dal LIDAR, è necessario conoscere le coordinate geografiche nelle quali il drone sta volando e il suo assetto. L'algoritmo che acquisisce i dati LIDAR produrrà delle coordinate del terreno grazie alle quali il sistema di navigazione potrà conoscere le zone percorribili e quelle rischiose per l'incolumità del drone.

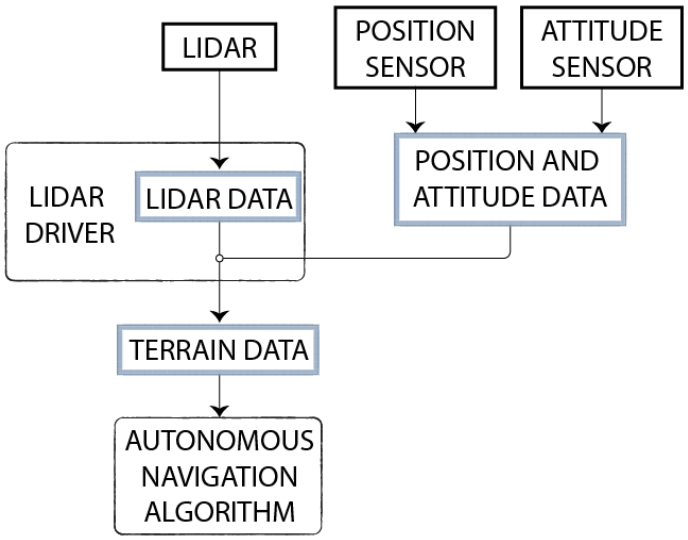


Figura 1.1: Il flusso di dati dai sensori all'algoritmo di navigazione autonoma.

Per il procedimento di acquisizione dei dati di posizione e per la navigazione autonoma sul terreno si faccia riferimento alla tesi parallela di Sara Senzacqua [5].
Oltre al funzionamento dell'hardware reale, abbiamo deciso di implementare un simulatore di sensore LIDAR che accetti in ingresso gli stessi parametri del LIDAR reale e restituisca un output nello stesso formato. Il simulatore è fondamentale per poter validare gli algoritmi sviluppati prima di mettere a rischio il drone.

Con questa tesi ho dunque sviluppato:

- un *driver* per l'acquisizione e l'elaborazione dei dati LIDAR
- un algoritmo per la simulazione del sensore LIDAR facendo volare il drone virtuale su una mappa DEM (Digital Elevation Model)

- una routine Matlab per visualizzare i dati del terreno ottenuti e confrontarli con la mappa di partenza.

Il lavoro svolto nel corso della tesi sarà utile agli sviluppi futuri del progetto di cui fa parte, ma può essere applicato in altri contesti che prevedono l'utilizzo di un LIDAR reale o virtuale. In particolare non è ristretto prettamente all'ambito aereo: è una tecnologia che può trovare ampia applicazione in qualsiasi progetto che coinvolga lo spostamento autonomo in terreno incognito (auto senza guidatore, *rover*, robotica...).

La tesi si estende in quattro capitoli: il primo tratta del sensore LIDAR e di come acquisire i dati; il secondo illustra il procedimento per la creazione di una nuvola di punti rappresentativa del terreno; il terzo spiega il rapporto tra i dati del sistema di navigazione e il funzionamento del driver, oltre a contenere una guida sull'implementazione del codice su un possibile computer di bordo; il quarto si occupa della simulazione del sensore LIDAR a partire dalla mappa DEM.

Capitolo 2

Stato dell'arte

La tecnologia LIDAR è usata ormai comunemente per creare mappe tridimensionali dell'ambiente circostante [6]. Vi sono applicazioni specifiche per mappare foreste e il loro sviluppo sul territorio [7], anche affiancando il sensore LIDAR con metodi diversi come fotocamere infrarosse [8]. Certamente è uno strumento versatile per la creazione di nuvole di punti che descrivano un ambiente [9]. Numerosi studi si sono occupati di acquisire dati di terreno con un LIDAR montato su un piccolo drone [10].
Mentre le applicazioni sul suolo tendono a favorire sensori a 360°, ovvero che acquisiscano i punti in tutte le direzioni intorno al veicolo, nei casi in cui un LIDAR sia montato su un oggetto volante, tendenzialmente il sensore avrà una direzione preferenziale verso il basso. Costituiscono eccezione esemplare i droni per la mappatura interna di edifici difficilmente accessibili [11]
Il problema a cui la tesi fa riferimento è non solo l'acquisizione di dati grezzi dal sensore su un drone, ma anche l'elaborazione di essi in tempo reale per essere comunicati immediatamente al sistema di navigazione autonoma [12, 13].
Per questo ci siamo occupati di far creare una mappa in cui il drone possa navigare, ma che al tempo stesso fornisca informazioni sulla topologia del terreno che possono essere sfruttati in altri modi [14].

Il sensore da noi acquistato è un LeddarTech Leddar Vu8 [15] a configurazione orizzontale. Insieme al sensore viene fornito un kit per lo sviluppo del software (Leddar Enabler SDK), che contiene un'interfaccia di programmazione delle applicazioni (API) *user-friendly* con librerie .NET e C oltre ad un esempio di codice per Windows e Linux.
Il driver dimostrativo si limita a stampare a video i dati acquisiti dal sensore, elaborati secondo le impostazioni di partenza. Per ognuno degli otto raggi LIDAR, stampa la distanza del target e l'intensità della risposta: non si occupa in nessun modo di come il sensore è posizionato nell'ambiente. In letteratura si trovano principalmente implementazioni di LIDAR per la mappatura a partire da dati di sensori dal funzionamento diverso, in particolare il molto usato tipo sensore rotante a 360 gradi [10].

Di più raro sviluppo è la simulazione di sensori per la mappatura con il fine di dimostrare la funzionalità dei sistemi di navigazione senza dover far volare nessun oggetto [16]. Questi si basano principalmente sul *ray tracing* [17] ottimizzato per permettere una simulazione in tempo reale [18]. I metodi trovati in letteratura si occupano principalmente di sensori

per *imaging*, ma ci sono applicazioni anche con LIDAR [19].

Capitolo 3

Il LIDAR

Il sensore che serve in questo progetto deve essere in grado di spazzare il terreno per ottenere una mappa sopra alla quale un drone possa volare. La tipologia rotante a 360° non è adatta allo scopo, quindi è stato scelto un sensore *allo stato solido*, ovvero senza parti in movimento: LeddarTech Leddar Vu8 a configurazione orizzontale con campo visivo di 48° `x` 0.3°.
Per testare il funzionamento del LIDAR acquistato, mi sono occupato di sviluppare un driver che, oltre a raccoglierne i dati, produca una mappa del terreno sul quale esso sta volando.
Per cominciare ho scaricato l'SDK di Leddar [20] che contiene un driver esemplificativo delle funzioni principali scritto in C++.
Una volta compreso il funzionamento del driver, l'ho modificato per incontrare i requisiti dello studio, ho verificato l'accuratezza dei dati estrapolati e ne ho esteso la funzionalità per creare una nuvola di punti e una mappa.

3.1 Metodo

3.1.1 Alterazione dell'algoritmo di acquisizione dati del LIDAR

L'esempio presente nell'SDK si occupa della connessione di un computer a diversi tipi di LIDAR LeddarTech attraverso diverse interfacce. Per il LeddarTech Vu8 le interfacce disponibili sono:

- USB/Serial tramite protocollo Modbus
- SPI
- CANBus (protocollo SPI)
- CANBus (protocollo CAN)

Ho alleggerito il codice di tutto ciò che non riguardasse la connessione del nostro LIDAR tramite USB.
Una volta connesso alla periferica voluta, il software permette di estrapolare i dati (**echoes**) acquisiti dal LIDAR per ogni rilevazione; il formato di output predefinito è:

Tabella 3.1: Rilevamento su parete piana parallela al sensore.

N^o canale	Distanza rilevata	Ampiezza di segnale
7	2.77	31.7
6	2.6	50.6
5	2.55	57.5
4	2.53	62.2
3	2.56	65.4
2	2.58	60.7
1	2.62	52.8
0	2.76	36.9

Per motivi attribuibili all'alta velocità di acquisizione, è frequente che uno degli otto echoes sia mancante o, talvolta, sia raddoppiato. Per questo ho memorizzato i dati di un'acquisizione in una matrice più ordinata in cui gli echoes mancanti hanno valore pari a quelli acquisiti in posizione analoga durante la rilevazione precedente.
Di questa matrice il software stampa solamente la riga in cui compaiono gli echoes delle distanze (canali da 0 a 7).

Tabella 3.2: Rilevamento su parete piana parallela al sensore – sole distanze.

2.76	2.62	2.58	2.56	2.53	2.55	2.6	2.77

Come prima verifica della verosimiglianza dei valori forniti dal LIDAR, ho messo in atto un esperimento.
Impostata la frequenza di campionamento delle ottuple a circa 40 Hz, ho posizionato il sensore in verticale in modo che il piano dei raggi fosse perpendicolare al suolo; ho avviato l'acquisizione dei dati e ho cominciato a ruotare il sensore intorno a un asse verticale con velocità costante. Dopo qualche secondo, ho fermato la registrazione e ho copiato le distanze output del software su Excel.
Ho trattato ogni elemento della matrice come un pixel, assegnando una scala di colore alle caselle, in dipendenza del valore che contenevano: rosso per i valori più alti (punti più distanti) e verde per quelli più bassi (punti più vicini).
Ho quindi scattato una fotografia dallo stesso punto dal quale ho effettuato la misura con il LIDAR, e ho confrontato le due immagini ottenute. Si nota una forte correlazione tra i dati di distanza estratti dal LIDAR e le profondità visibili nella fotografia, segno del corretto funzionamento del dispositivo.

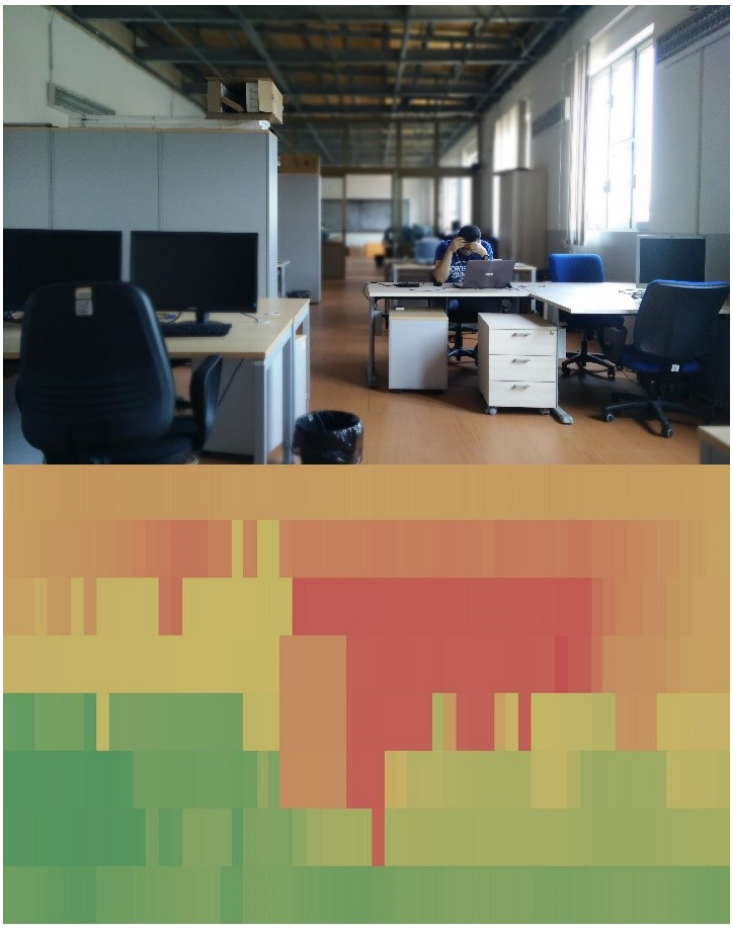


Figura 3.1: Confronto tra fotografia e dati estratti dal LIDAR.

3.1.2 Verifica e misura dell'accuratezza delle misurazioni ottenute dal LIDAR

Dato che l'obbiettivo è di creare un profilo di altezza del suolo, è necessario eliminare l'errore di parallasse commesso dall'angolo di ogni singolo echo di distanza.

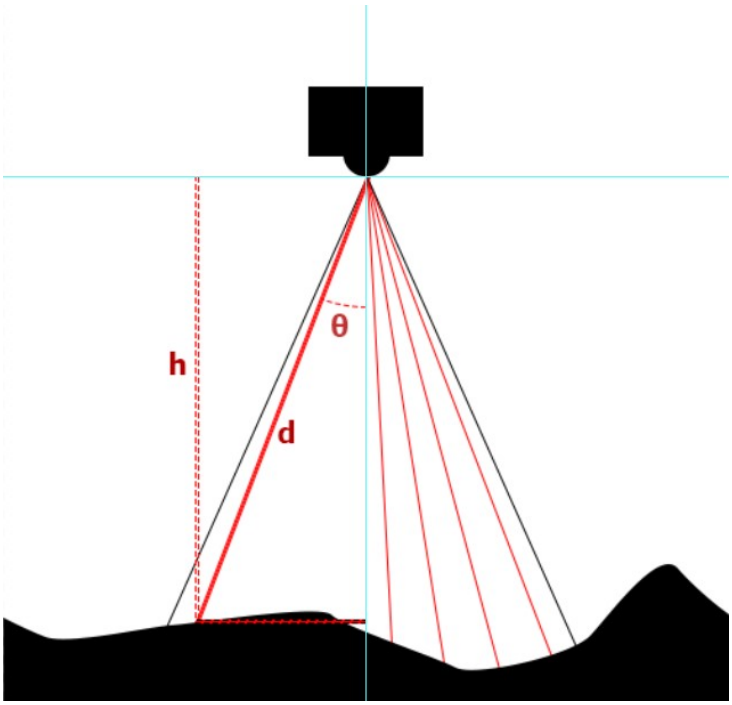


Figura 3.2: Correzione parallasse per la mappatura del terreno.

Avendo la distanza d e conoscendo l'angolo θ per ogni echo, si può facilmente calcolare l'altezza h con

$$h = d \cdot \cos\theta$$

Con la correzione, le altezze rilevate puntando il LIDAR su una parete piana parallela al sensore dovrebbero essere tutte uguali.

2.76	2.62	2.58	2.56	2.53	2.55	2.6	2.77
2.58	2.53	2.55	2.56	2.53	2.51	2.51	2.58

Figura 3.3: Distanze d dal sensore (sopra) e altezze h dopo la correzione (sotto).

Riporto i valori in un grafico per renderli di immediata comprensione.

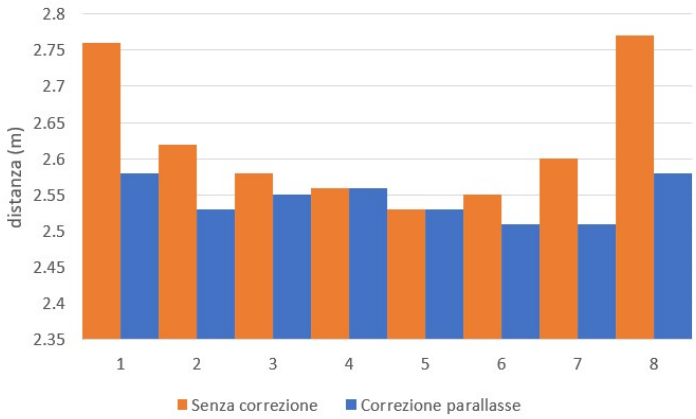


Figura 3.4: Confronto correzione parallasse.

I valori dopo la correzione non sono esattamente uguali a causa degli errori di misura del LIDAR.

Per provare quanto sono accurate le rilevazioni del LIDAR ho predisposto un test preciso e ripetibile. Ho posto il LIDAR a una distanza nota da una parete piana, in modo che l'emettitore ed il sensore fossero paralleli alla parete. Ho svolto l'esperimento con la stessa disposizione a distanza di 1, 2, 4 e 5 metri dalla parete.
Ho cominciato il setup dell'esperimento regolando i parametri di configurazione del LIDAR in modo che fornissero i risultati più accurati possibile. Una guida alla configurazione può essere trovata sul sito di LeddarTech [21]. In particolare, ho notato che il *noise removal* e l'*overshoot management* tendono a peggiorare i risultati, mentre l'*object demerging*[1] non pare cambiarli. Inoltre, lo *smoothing* serve a rendere i risultati più costanti, ma ho preferito fare una media manualmente onde evitare risultati già sottoposti a elaborazione. Ho impostato l'*oversampling* a 8, valore tipico negli esempi forniti sul sito LeddarTech, e l'*accumulation* a 64, in modo da avere una frequenza di campionamento di quasi 5Hz, ma anche una riduzione del rumore sufficiente.
Gli errori assoluti (e_a) e relativi (e_r) di seguito sono calcolati tenendo conto della distanza h, derivata dall'*echo* del driver, e della distanza nominale dalla parete (h_n):

$$e_a = |h - h_n|$$

$$e_r = \frac{|h - h_n|}{h_n}$$

3.2 Dati della validazione

Tabella 3.3: Errori assoluti da 1 metro (cm)

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	10.12	7.97	10.24		8.00	7.90	8.16	10.00
	2	10.02	7.94	10.09		7.94	7.76	8.06	9.98
	3	10.26	8.00	10.47		8.00	7.81	8.16	10.00
	4	10.11	7.95	10.23		8.00	7.77	8.05	10.00

Tabella 3.4: Errori relativi da 1 metro

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	10.1%	8.0%	10.2%		8.0%	7.9%	8.2%	10.0%
	2	10.0%	7.9%	10.1%		7.9%	7.8%	8.1%	10.0%
	3	10.3%	8.0%	10.5%		8.0%	7.8%	8.2%	10.0%
	4	10.1%	8.0%	10.2%		8.0%	7.8%	8.0%	10.0%

[1] Eases the discrimination of multiple objects in the same segment. [...]. The measurement of demerged objects tends to be less precise than for usual detections.

Tabella 3.5: Errori assoluti da 2 metri (cm)

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	0.95	-1.85	4.10	4.97	1.05	0.54	-1.85	1.31
	2	1.98	-1.08	4.32	5.00	1.09	0.62	-1.79	1.15
	3	3.80	-0.15	4.17	4.95	1.17	0.51	-1.56	1.44
	4	3.89	-0.46	4.28	4.94	1.13	0.52	-1.74	1.39

Tabella 3.6: Errori relativi da 2 metri

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	0.5%	-0.9%	2.0%	2.4%	0.5%	0.3%	-0.9%	0.6%
	2	1.0%	-0.5%	2.1%	2.4%	0.5%	0.3%	-0.9%	0.6%
	3	1.9%	-0.1%	2.0%	2.4%	0.6%	0.2%	-0.8%	0.7%
	4	1.9%	-0.2%	2.1%	2.4%	0.6%	0.3%	-0.8%	0.7%

Tabella 3.7: Errori assoluti da 4 metri (cm)

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	4.95	-2.05	1.00	4.00	6.00	5.00	1.29	2.00
	2	4.78	-2.89	1.00	4.00	5.72	5.00	1.15	1.89
	3	4.17	-3.00	1.00	4.00	5.95	5.00	1.00	1.02
	4	4.11	-3.00	1.00	4.00	6.00	5.00	1.00	1.00

Tabella 3.8: Errori relativi da 4 metri

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	1.2%	-0.5%	0.2%	1.0%	1.5%	1.3%	0.3%	0.5%
	2	1.2%	-0.7%	0.2%	1.0%	1.4%	1.3%	0.3%	0.5%
	3	1.0%	-0.8%	0.2%	1.0%	1.5%	1.3%	0.2%	0.3%
	4	1.0%	-0.8%	0.3%	1.0%	1.5%	1.3%	0.3%	0.3%

Tabella 3.9: Errori assoluti da 5 metri (cm)

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	4.31	1.00	4.30	7.98	9.14	8.00	3.86	2.00
	2	4.00	1.00	4.00	7.00	9.00	8.00	3.00	2.00
	3	4.00	1.00	4.00	7.04	9.00	8.00	3.00	2.00
	4	4.00	1.00	4.00	7.19	9.00	8.00	3.06	2.00

Tabella 3.10: Errori relativi da 5 metri

		Echoes							
		1	2	3	4	5	6	7	8
# test	1	0.9%	0.2%	0.9%	1.6%	1.8%	1.6%	0.8%	0.4%
	2	0.8%	0.2%	0.8%	1.4%	1.8%	1.6%	0.6%	0.4%
	3	0.8%	0.2%	0.8%	1.4%	1.8%	1.6%	0.6%	0.4%
	4	0.8%	0.2%	0.8%	1.4%	1.8%	1.6%	0.6%	0.4%

Si possono riassumere i risultati nei seguenti grafici. Nel primo, che rappresenta gli errori assoluti nei quattro test, ho riportato la deviazione standard delle misure per ogni distanza nominale.

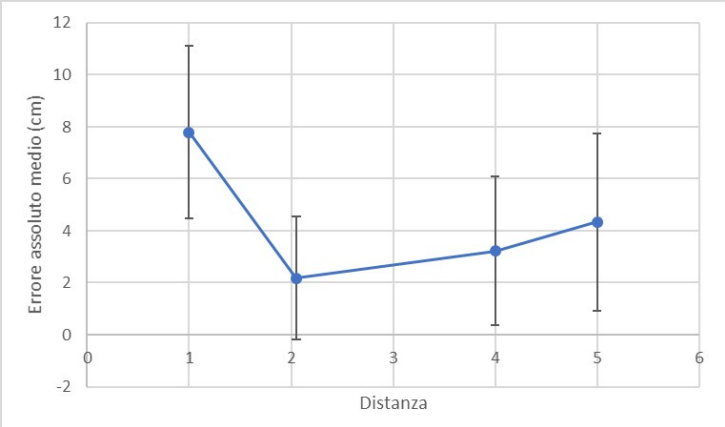


Figura 3.5: Errori assoluti medi alle varie distanze.

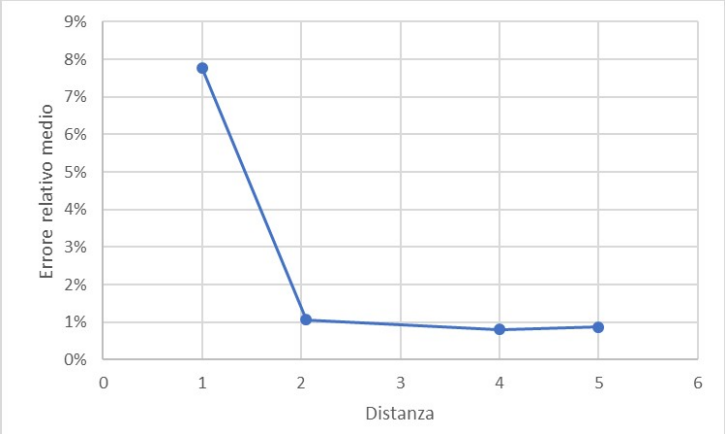


Figura 3.6: Errori relativi medi alle varie distanze.

3.3 Discussione

L'esperimento alla distanza di 1 metro ha portato dei risultati visibilmente insoddisfacenti. L'eccessiva vicinanza con la parete ha reso il quarto echo inutilizzabile a causa della

saturazione del segnale luminoso, e i dati dei segnali funzionanti sono tutti abbondantemente sovrastimati. A breve distanza è anche grande la dipendenza dal colore e dalla riflettività della parete. Chiaramente il LIDAR non è stato progettato per essere accurato ad una distanza così ridotta.
Alle distanze dai 2 metri in su si vede un notevole miglioramento in accuratezza. Nel complesso vediamo l'errore totale che tende a crescere con la distanza; al contrario l'errore relativo scende. Questo può essere attribuito al fatto che il LIDAR calcola la distanza tramite una differenza di tempo, e che quindi le inaccuratezze crescano al crescere della distanza, anche se non velocemente quanto la distanza stessa.
Contrariamente a quanti mi sarei aspettato, l'errore relativo da 5 metri è superiore a quello da 4 metri, ma lo è di talmente poco da essere assimilabile ad una fluttuazione statistica.
Ho notato che i dati rimangono coerenti anche dopo che la rilevazione viene perturbata: con il LIDAR azionato di fronte alla parete chiara i dati ricevuti sono gli stessi di quelli acquisiti dopo aver fatto passare un oggetto scuro a pochi centimetri dal sensore. Questo non è ovvio, in quanto le intensità dei laser cambiano in modo automatico quando le distanze e i tipi di superficie lo richiedono.

I dati ottenuti in questa verifica non rientrano tutti nella accuratezza di 5 cm pubblicizzata nelle specifiche dello strumento [22]. Non sono rese molto chiare le condizioni dei loro test, quindi non è possibile fare un confronto scientifico.
I risultati non sono completi per una validazione definitiva del metodo. Per ottenere risultati più esaustivi consiglio di effettuare test su distanze più grandi, utilizzando un drone o spazi aperti con lunghe pareti piane, condizioni che non potevano essere soddisfatte nell'ambito di questa tesi.

Capitolo 4

Nuvola di punti del terreno tramite i dati LIDAR, orientamento e posizione

Fino ad ora i dati del LIDAR erano semplicemente una serie di vettori numerici, e permettevano una misurazione bidimensionale senza informazioni sulla posizione nella quale le misurazioni venissero fatte.
Per costruire una mappatura del terreno sopra il quale il drone vola, sono necessari di tre elementi fondamentali:

- La posizione del sensore LIDAR nel sistema ambiente (ad esempio con GPS montato sullo stesso drone)
- L'orientamento del sistema di riferimento del sensore LIDAR rispetto a quello ambiente (per esempio ottenibile con accelerometri)
- I dati di distanza forniti dall'output del sensore LIDAR, associati ai diversi echoes dello stesso.

La combinazione di questi dati è un problema trigonometrico. L'ho approcciato riferendomi ad un sistema di riferimento solidale al drone (X, Y e Z) e con origine nel LIDAR, e poi ruotando nello spazio le coordinate ottenute in dipendenza dell'assetto del drone, per condurmi al sistema di riferimento dell'ambiente (x, y e z) con origine nel sensore LIDAR.
Chiamo le coordinate del drone nel sistema ambiente Lat_d, Lon_d e Alt_d, mentre l'indice i sarà riferito all'$i$esimo echo, e ne rappresenta il punto di rilevamento sul terreno (distante dal drone x_i, y_i e z_i).
Chiamerò gli angoli di rollio, beccheggio e imbardata con i loro nomi inglesi *roll*, *pitch* e *yaw* per brevità lessicale. Per eseguire la mappatura si suppone che il LIDAR sia montato in modo da avere il piano del fascio di rilevamento perpendicolare al piano *avanti/dietro-alto/basso* del drone.
Inoltre introduco un angolo ϕ di puntamento in avanti del LIDAR rispetto al drone.

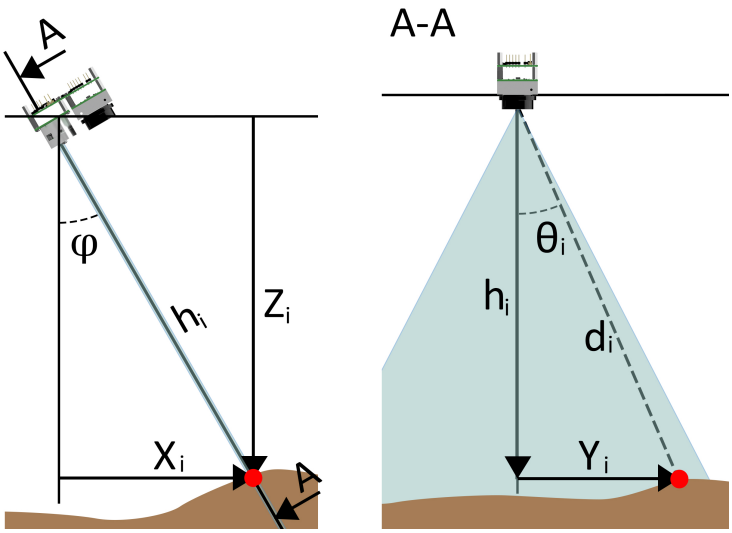


Figura 4.1: Il punto rilevato nel sistema di riferimento solidale al drone.

Per conoscere le coordinate del punto sul terreno i nel sistema di riferimento solidale al drone, basta avere l'echo di distanza del punto (d_i), l'angolo rispetto alla verticale dell'echo in questione (θ_i) e l'angolo di puntamento (ϕ). Si può dunque calcolare

$$X_i = d_i \cdot \cos(\theta_i) \cdot \sin(\phi)$$

$$Y_i = -d_i \cdot \sin(\theta_i)$$

$$Z_i = d_i \cdot \cos(\theta_i) \cdot \cos(\phi)$$

ovvero un vettore che parte dal drone e finisce nel punto i.
Voglio poi trasformare tale vettore nel sistema di riferimento globale, ruotato degli angoli di assetto rispetto a quello solidale al drone. Ricorro alle matrici di rotazione tridimensionale [23]

$$R_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(roll) & -\sin(roll) \\ 0 & \sin(roll) & \cos(roll) \end{bmatrix}$$

$$R_{pitch} = \begin{bmatrix} \cos(pitch) & 0 & \sin(pitch) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & \cos(pitch) \end{bmatrix}$$

$$R_{yaw} = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

i cui termini scriverò con notazione ridotta

$$R_{roll} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cr & -sr \\ 0 & sr & cr \end{bmatrix}, \; R_{pitch} = \begin{bmatrix} cp & 0 & sp \\ 0 & 1 & 0 \\ -sp & 0 & cp \end{bmatrix}, \; R_{yaw} = \begin{bmatrix} cy & -sy & 0 \\ sy & cy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dato che devo ruotare il vettore (X_i, Y_i, Z_i) intorno ai tre assi ortogonali di assetto, definisco

$$R = R_{roll} \; R_{pitch} \; R_{yaw}$$

Dunque mi ricalcolo la distanza tra il drone e il punto i nel sistema di riferimento globale (x_i, y_i e z_i) con

$$\begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix} = R \begin{Bmatrix} X_i \\ Y_i \\ Z_i \end{Bmatrix}$$

Esplicitamente

$$x_i = X_i(cp \cdot cy) + Y_i(-cp \cdot sy) + Z_i(sp)$$

$$y_i = X_i(sr \cdot sp \cdot cy + cr \cdot sy) + Y_i(-sr \cdot sp \cdot sy + cr \cdot cy) + Z_i(-sr \cdot cp)$$

$$z_i = X_i(-cr \cdot sp \cdot cy + sr \cdot sy) + Y_i(cr \cdot sp \cdot sy + sr \cdot cy) + Z_i(cr \cdot cp)$$

Si tratta quindi di trasformare le distanze ottenute (che compaiono nello stessa unità di misura del dato d_i, quindi in metri) in coordinate latitudine, longitudine e altitudine (Lat_i, Lon_i e Alt_i).
Conoscendo la posizione del drone e assumendo che sulla superficie terrestre (raggio $r_T = 6371 \cdot 10^3\ m$)

$$1^\circ <=> (r_T \cdot \pi/180)\ metri$$

si può dire

$$Lat_i = Lat_d + \frac{x_i}{r_T \cdot \pi/180}$$

$$Lon_i = Lon_d + \frac{y_i}{r_T \cdot \pi/180}$$

$$Alt_i = Alt_d - \frac{z_i}{r_T \cdot \pi/180}$$

Dunque per ogni echo ottengo una coordinata (Lat_i, Lon_i, Alt_i) di un punto del terreno. Questi calcoli sono stati implementati nell'algoritmo del LIDAR e producono un file di testo a tre colonne, corrispondenti alle tre dimensioni latitudine, longitudine e altitudine. Mettendo insieme queste informazioni e facendo muovere il drone, posso costruire una nuvola di punti del terreno sottostante in tempo reale.

Capitolo 5

Inserimento del driver LIDAR nel processo di acquisizione dati

5.1 Il computer di bordo

Un aeromobile autonomo, nel corso della sua missione, deve elaborare una notevole quantità di dati. Siano essi finalizzati a mantenere il volo stabile o siano per portare avanti gli scopi della missione, i soli sensori non sono sufficienti per gestire le informazioni che vengono raccolte: si palesa la necessità di un computer a bordo.

Le caratteristiche necessarie agli scopi di missione sono la piccola dimensione e il basso peso, unitamente ad una potenza di calcolo sufficiente e delle interfacce a cui connettere i vari sensori e attuatori. Si è scelto di utilizzare una Raspberry Pi 3B+ poiché soddisfa i requisiti ed ha un prezzo abbordabile.

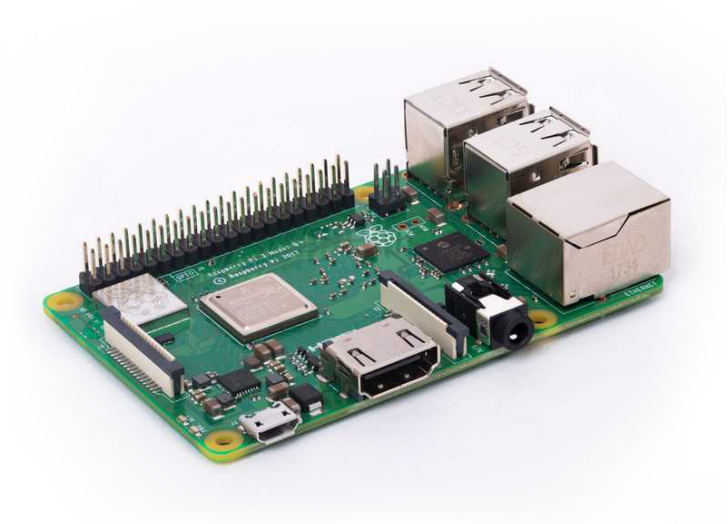
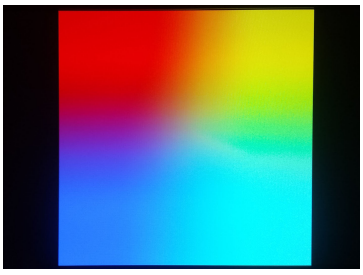


Figura 5.1: Una Raspberry Pi 3B+.

5.1.1 Installazione del sistema operativo

Si è scelto di utilizzare un sistema operativo Linux per la sua leggerezza, compatibilità e diffusione nell'ambiente dei single-board computer: in particolare si è deciso di installare *Raspbian*, il sistema operativo sviluppato per la Raspberry Pi basato sulla distribuzione Debian.
Il sistema operativo deve essere installato sulla scheda SD che sarà la memoria principale del computer. Per fare ciò ho scaricato dal mio PC l'immagine della più recente versione del software e, usando *Win32 Disk Imager*, l'ho installato sulla scheda. Ho preso in considerazione di installare *APSync*, in quanto è un sistema operativo molto simile a Raspbian ed è mirato a semplificare il setup dei companion computer; tuttavia l'ultima versione si è rivelata difettosa e non pienamente compatibile con la Raspberry Pi in mio possesso, quindi l'idea è stata abbandonata.

(a) La schermata di avvio di Raspbian.



(b) L'errore di avvio con APSync.

Figura 5.2: Il sistema operativo della Raspberry Pi

Una volta installato il sistema operativo e inserita la SD nella Raspberry Pi, il computer di bordo è pronto ad essere configurato, utilizzando uno schermo con cavo HDMI, una tastiera USB e un collegamento LAN.

5.1.2 Configurazione del computer di bordo

Il primo accesso è effettuato con il nome utente e la password default: **pi**, **raspberry**. È opportuno cambiare subito la password per motivi di sicurezza: dal terminale si digita

```
sudo raspi-config
```

per entrare nel pannello di controllo della Raspberry Pi. Dopo aver riavviato il sistema, vogliamo connettere il dispositivo a una rete internet. Con il comando

```
ls /sys/class/net/
```

vediamo la lista delle interfacce di rete, tipicamente `eth0` per connessioni LAN, `lo` per connessione *loopback*, `wlan0` per connessione wireless.
Con il comando

```
ip -4 addr show | grep global
```

otteniamo l'indirizzo *IP (inet)* per ognuna delle interfacce; nel caso della mia *wlan0* `192.168.1.111/24`, per la *eth0* `192.168.1.110/24`. Con

```
ip route | grep default | awk '{print $3}'
```

otteniamo l'indirizzo del router o *gateway address*; nel mio caso `192.168.1.254`. Eseguendo

```
cat /etc/resolv.conf
```

abbiamo l'indirizzo del *server DNS*, per me uguale a quello del router.
Per avere un IP fisso, cosa che servirà nella connessione SSH, vado a modificare la configurazione *dhcp* con

```
sudo nano /etc/dhcpcd.conf
```

aggiungendo le informazioni estrapolate in precedenza. Nel mio caso aggiungo le linee

```
interface eth0
    static ip_address = 192.168.1.110/24
    static routers = 192.168.1.254
    static domain_name_servers = 192.168.1.254
interface wlan0
    static ip_address = 192.168.1.111/24
    static routers = 192.168.1.254
    static domain_name_servers = 192.168.1.254
```

dove i numeri 110 e 111 sono scelti arbitrariamente tra 1 e 255.
Si abilita quindi la connessione manuale delle interfacce aprendo

```
sudo nano /etc/network/interfaces
```

e aggiungendo le linee

```
source-directory /etc/network/interfaces.d
auto lo
iface lo inet loopback
iface eth0 inet manual
allow-hotplug wlan0
iface wlan0 inet manual
    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

Nel caso di rete Wi-Fi è necessario inserire le credenziali. Va modificato

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

aggiungendo le righe

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=IT
network={
    ssid="*ssid della rete*"
    psk="*password*"
    scan_ssid=1
}
```

Per verificare che la connessione funzioni si può usare il comando

```
ping google.com
```

Al termine della configurazione è opportuno riavviare il sistema.

5.1.3 Operare la Raspberry Pi da remoto

La prima necessità per poter lavorare con la Raspberry Pi è comunicare con essa tramite un protocollo SSH: in questo modo sarà possibile agire su linea di comando da un computer remoto, o anche accedere alle cartelle tramite l'interfaccia grafica di *WinSCP*.
Per abilitare la funzionalità SSH è necessario configurare la Raspberry Pi con il comando

```
sudo raspi-config
```

e nella sezione *interfacing* abilitare SSH.

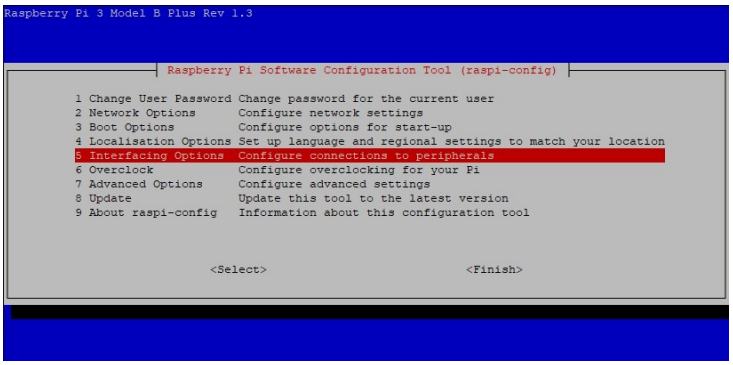


Figura 5.3: La schermata di configurazione.

Nella stessa sezione va abilitata la porta seriale mantenendo OFF il *login shell* e attivando la *serial port interface*; riavviare poi il sistema.

A questo punto si può accedere alla Raspberry Pi da un computer remoto Windows in diversi modi.

Accesso da linea di comando

Aprendo il *command prompt* e digitando nel mio caso

```
ssh pi@192.168.1.110
```

vengono richieste una conferma e la password della Raspberry Pi; inseritele, si può operare sulla riga di comando del sistema operativo del computer di bordo.
`ssh` è il protocollo di comunicazione, `pi` è il nome dell'utente con cui voglio accedere, `192.168.1.110` è l'indirizzo IP statico precedentemente configurato.

Accesso con *Putty*

Volendo evitare di inserire i valori numerici a tutti gli accessi, *Putty* è un software che memorizza i dati di accesso. Una volta impostati, si possono salvare per futuri accessi. Il risultato è fondamentalmente lo stesso dell'operazione del punto precedente.

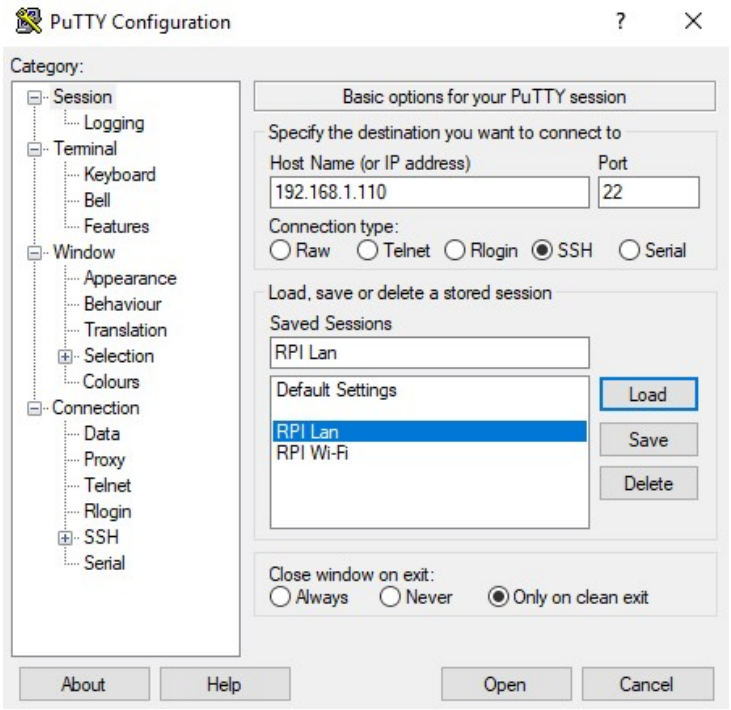


Figura 5.4: Il software Putty.

Accesso con WinSCP

Otteniamo invece un diverso approccio con il programma WinSCP. Anch'esso necessita della stessa memorizzazione dei dati che avevamo con Putty, ma, una volta stabilita la connessione, potremo accedere alle cartelle della Raspberry Pi tramite un'interfaccia grafica simile a quella del file explorer di Windows.

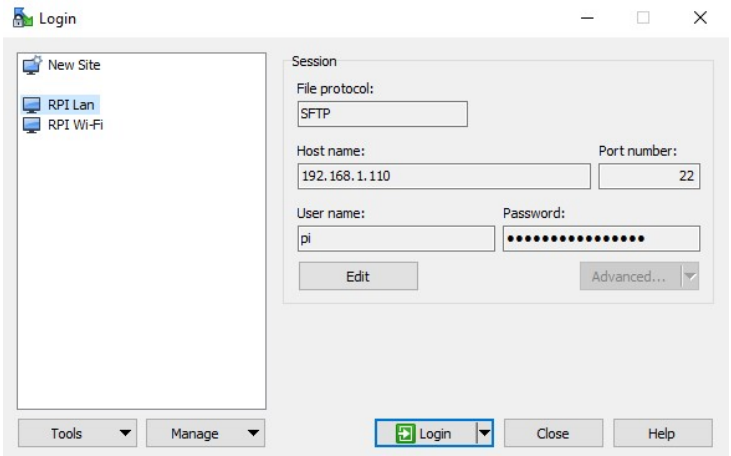


Figura 5.5: Il software WinSCP.

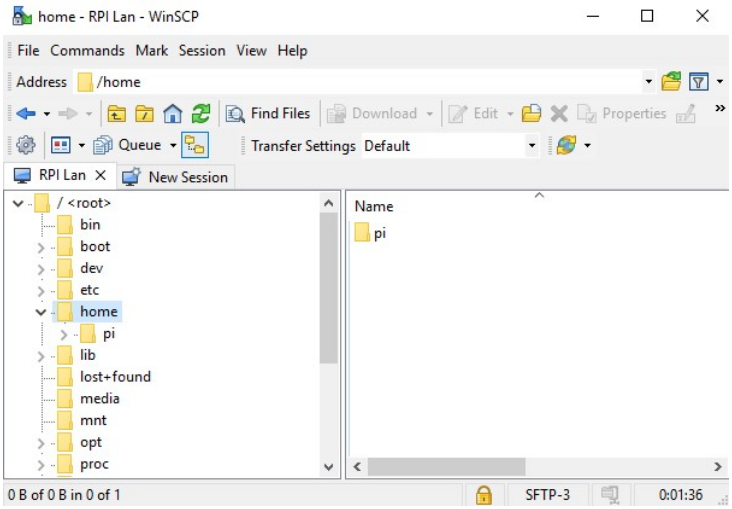


Figura 5.6: L'interfaccia con le cartelle della Raspberry Pi in WinSCP.

A questo punto si può facilmente lavorare sulla Raspberry Pi senza connettervi schermo o periferiche, anche copiando e incollando file tramite WinSCP.

5.2 Trasferimento del driver LIDAR sulla Raspberry Pi

Per far funzionare il driver sulla Raspberry Pi, bisogna predisporla con una serie di comandi.
Innanzi tutto è necessario installare un driver USB per fare riconoscere il sensore LIDAR come dispositivo di input: si usa il comando

```
sudo apt-get install libusb-1.0-0-dev
```

Va scaricato l'SDK Leddar che già contiene un esempio di driver tramite il comando

```
git clone https://github.com/leddartech/LeddarSDK.git
```

Ho svuotato la cartella

```
cd LeddarSDK-master/src/LeddarExample
```

inserendoci invece l'algoritmo *LeddarExample.cpp* da me scritto.

Nella cartella

```
cd LeddarSDK-master/src/
```

si trova il file *build.sh* che permette di compilare facilmente il codice. Se ne abilita l'eseguibilità con

```
chmod +x build.sh
```

e lo si esegue con

```
./build.sh
```

Per far partire il codice compilato è sufficiente il comando

```
./release/LeddarExample
```

Il driver aspetta di trovare i *file* di posizione e assetto derivanti dal sistema di controllo, dopodiché comincia a scrivere le coordinate del terreno che rileva, fino a quando non viene interrotto.

5.3 Scambio di dati

Come accennato in precedenza, i dati acquisiti dal LIDAR sono complementari alle informazioni di posizione fornite dal sistema di controllo. Queste ultime devono in qualche modo essere comunicate al driver Leddar, il quale, dopo averle elaborate secondo i dati acquisiti, restituirà i dati del terreno da implementare nell'algoritmo di navigazione autonoma.

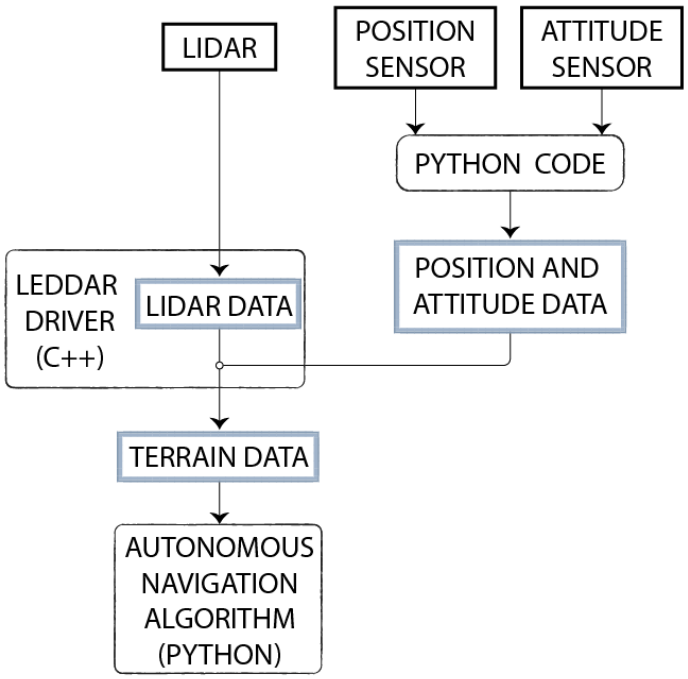


Figura 5.7: Il flusso di dati dai sensori all'algoritmo di navigazione autonoma.

È sorto un problema dovuto alla differenza del linguaggio di programmazione tra il codice per l'acquisizione di posizione/assetto e il driver del LIDAR: il primo è un codice *Python* che fa funzionare *Mavlink* e *Arducopter*[1], mentre il secondo è un codice *C++* con cui il LIDAR interagisce.
Dopo aver cercato una soluzione semplice che funzionasse anche sulla Raspberry Pi, sono arrivato alla conclusione che ognuno degli algoritmi dovesse modificare un *file* in tempo reale, in modo che il successivo potesse accedervi ed estrarne i dati. Ho testato questa

[1]Per approfondire questi argomenti si faccia riferimento alla tesi parallela di Sara Senzacqua

ipotesi ed ho verificato che è veloce a sufficienza per gli scopi del progetto. Ogni volta che un algoritmo accede ad un *file* prodotto dal precedente, apre la versione più aggiornata di questo.
In particolare, il codice Python acquisisce i dati di posizione e i dati di assetto e li sovrascrive rispettivamente sui documenti di testo *Lat-Lon-Alt.txt* e *Roll-Pitch-Yaw.txt* nella forma

latitudine (*in gradi*) *longitudine* (*in gradi*) *altezza* (*in metri*)

e

roll (*in radianti*) *pitch* (*in radianti*) *yaw* (*in radianti*)

Il driver in C++ apre i *files* e ne legge i dati ogni volta che il LIDAR compie una rilevazione: in questo modo aprirà il *file* dove i dati sono più aggiornati, anche se, per esempio, i dati di posizione non vengono sovrascritti da diversi cicli per via del fatto che il drone non si sta muovendo.
Ad ogni rilevazione i dati vengono elaborati, e la nuvola di punti del terreno viene scritta su un file di testo a tre colonne, corrispondenti alle tre dimensioni spaziali.
L'algoritmo di navigazione autonoma, scritto in Python, accede al documento, da cui estrae i dati del terreno.

Capitolo 6

Simulazione del LIDAR su una mappa DEM

Per verificare il corretto funzionamento di un algoritmo di navigazione autonoma, è impensabile applicarlo direttamente su un drone, in quanto un esito negativo potrebbe risultare in catastrofe. Per questo motivo abbiamo deciso di sviluppare un sensore LIDAR virtuale che, affiancato al simulatore di drone *SITL Ardupilot:Copter*, fornisca al sistema di navigazione dati analoghi a quelli che fornisce il driver del LIDAR da noi sviluppato.
Il sensore virtuale acquisisce da SITL dati di posizione e assetto analoghi a quelli che potrebbe ricevere da un drone reale, e calcola gli echo di distanza da una mappa DEM (Digital Elevation Model). Una volta in possesso di questi dati, li elabora in una nuvola di punti del terreno e consegna il risultato al sistema di navigazione autonoma.

Per simulare il LIDAR, voglio che un fascio virtuale di rilevazione delle distanze colpisca un terreno, dove ogni linea del fascio corrisponde a un echo del LIDAR, e il terreno è approssimato con una griglia di dati DEM. È necessario che questo avvenga con una frequenza di campionamento adatta ad acquisire un numero di dati del terreno utile alla creazione di una mappa virtuale di ostacoli.
Il metodo più utilizzato in letteratura per intersecare un fascio di rette con una superficie è il *ray tracing* [17], ma in principio risultava avere una frequenza di campionamento non sufficiente ai nostri scopi. Ho sviluppato un algoritmo basato su questo metodo, per poterlo confrontare con uno sviluppato da me.
Ho pensato ad un sistema alternativo al ray tracing, computazionalmente più leggero: di seguito lo chiamerò *metodo rapido*. Le ragioni dello sviluppo di questo metodo sono due: tentare di sviluppare un algoritmo potenzialmente più veloce anche se meno accurato, in modo da soddisfare la durata massima di acquisizione, e fornire in tempi più rapidi un algoritmo funzionante a chi si è occupato di testare la navigazione autonoma.
Infine sono tornato allo sviluppo del ray tracing, in quanto ho notato che potevano essere applicate notevoli ottimizzazioni fino a farlo diventare veloce quanto il *metodo rapido*.

6.1 Metodi di simulazione

6.1.1 Il *metodo rapido*

Per ogni posizione del drone virtuale, si ricavano le coordinate dell'intersezione di ciascun raggio con un piano orizzontale alla quota del punto sul terreno sulla verticale del drone; tenendo conto dell'intersezione, restituisce l'altitudine del punto della griglia di dati DEM più vicino a tali coordinate.
Considero il drone virtuale in posizione D, sulla verticale del punto sul terreno A, approssimato con il punto della griglia più vicino. Il LIDAR produrrà un raggio che interseca il terreno in B. Il nostro *metodo rapido* calcola l'intercetta C del raggio con il piano orizzontale su cui giace A, e la proietta in E (o il punto della griglia più prossimo), coordinata del terreno sulla verticale di C.

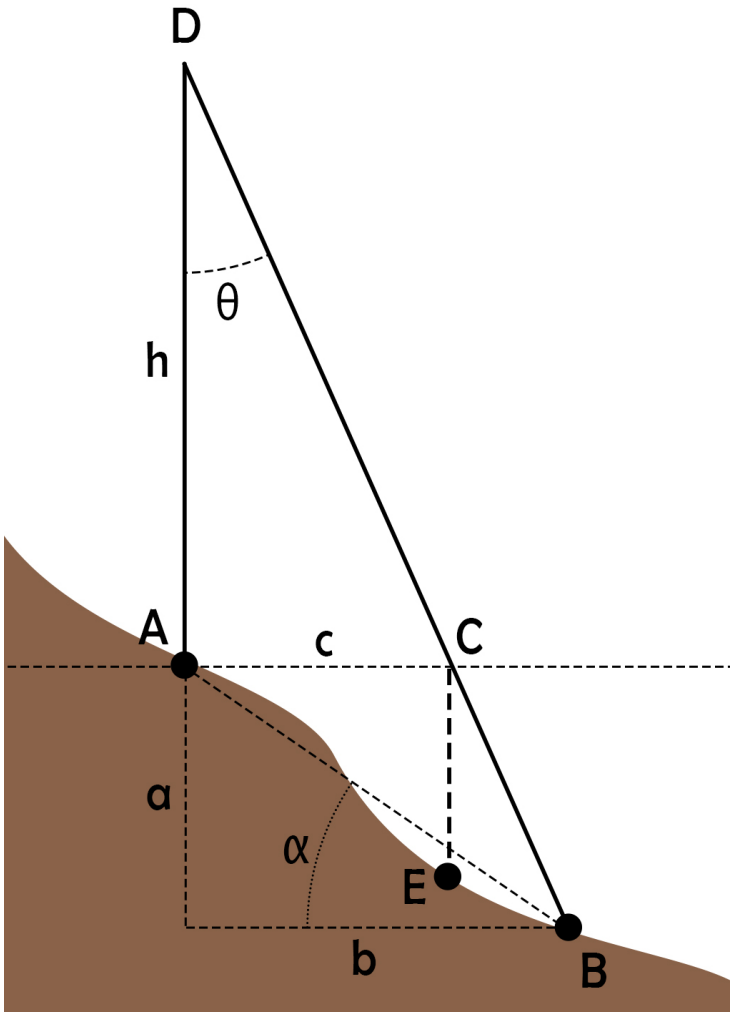


Figura 6.1: L'inaccuratezza del LIDAR simulato senza ray tracing.

Considero l'errore commesso su latitudine e longitudine

$$err = b - c$$

e voglio ricavarlo in funzione della pendenza del terreno α.
Dunque

$$\begin{cases} c = h \cdot \tan\theta \\ b = \frac{a}{\tan\alpha} \\ b = (a+h) \cdot \tan\theta \end{cases}$$

Risolvendo il sistema ottengo

$$err = h \cdot \tan\theta \cdot \left(\frac{1}{1 - \tan\theta \cdot \tan\alpha} - 1 \right)$$

6.1.2 Il *ray tracing*

In sostanza, quando si parla di ray tracing, si intende calcolare l'intersezione tra un insieme di rette e piani, che nel nostro caso rappresentano i raggi del LIDAR e la mappa del terreno. La mappa DEM da cui parto non fornisce altro che una griglia di punti nello spazio tridimensionale, equispaziati in latitudine e longitudine: ho quindi trasformato la griglia DEM nell'insieme di piani necessari all'applicazione del ray tracing tramite la conversione in una mesh 3D.

La mesh

Ho creato una mesh triangolare partendo dai dati di latitudine e longitudine del terreno. Questa è memorizzata in un file di quattro righe contenenti

- il numero dei triangoli che la compongono,
- il numero di vertici di ognuno dei poligoni (nel mio caso 3, essendo una mesh triangolare),
- l'indice dei vertici che compongono ogni triangolo,
- le coordinate tridimensionali di ciascuno dei vertici.

Ho ricavato il numero di triangoli (N_{tri}) dalla quantità di dati lungo la longitudine (N_{lon}) e lungo la latitudine (N_{lat}):

$$N_{tri} = 2(N_{lon} \cdot N_{lat} - N_{lon} - N_{lat} + 1)$$

Per il calcolo dei vertici di ogni triangolo ho considerato che ogni punto non appartenente all'ultima riga o colonna fosse l'origine di due triangoli. Per il punto i, i vertici avranno coordinate

triangolo 1 :	i	$i + N_{lon} + 1$	$i + N_{lon}$
triangolo 2 :	i	$i + 1$	$i + N_{lon} + 1$

Per una più chiara comprensione si faccia riferimento alla seguente figura, dove

$$N_{lon} = N_{lat} = 4$$

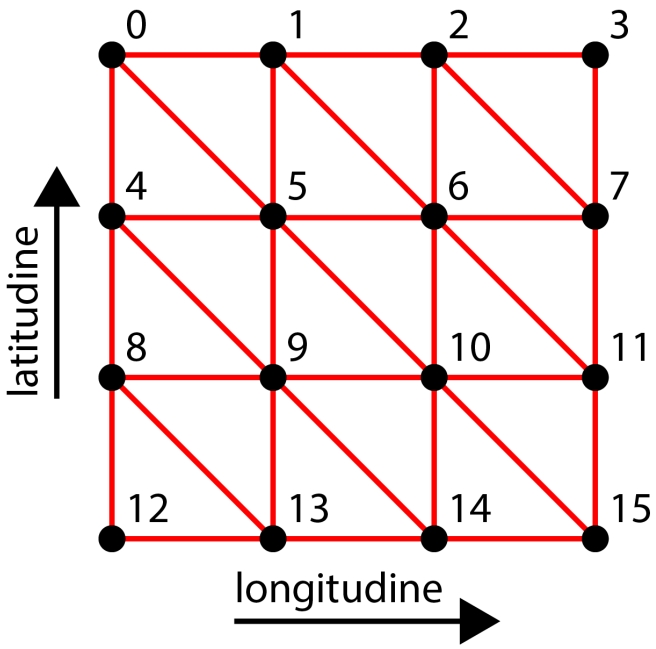


Figura 6.2: La conversione da griglia di dati in mesh.

Le coordinate di ciascuno dei vertici sono estrapolate dal file DEM.
Il documento di testo in cui la mesh è memorizzata dovrà essere letto dall'algoritmo di ray tracing.

Tracciare il raggio

Con la creazione della mesh ho stabilito i piani del terreno: devo ora definire i raggi che lo intersecano.
Per una descrizione univoca di un raggio ho bisogno di un'origine e una direzione: la prima corrisponde alla posizione nello spazio del sensore simulato, mentre la seconda la calcolo con un procedimento simile a quello descritto nel capitolo 4.
La direzione non è altro che un vettore tridimensionale di lunghezza unitaria definito come $[d_X,\ d_Y,\ d_Z]$ nel sistema di riferimento solidale al drone e $[dir_x,\ dir_y,\ dir_z]$ in quello ambiente stazionario. Per cui, riferendoci alla notazione precedente,

$$d_X = 1 \cdot \cos(\theta) \cdot \sin(\phi)$$

$$d_Y = -1 \cdot \sin(\theta)$$

$$d_Z = 1 \cdot \cos(\theta) \cdot \cos(\phi)$$

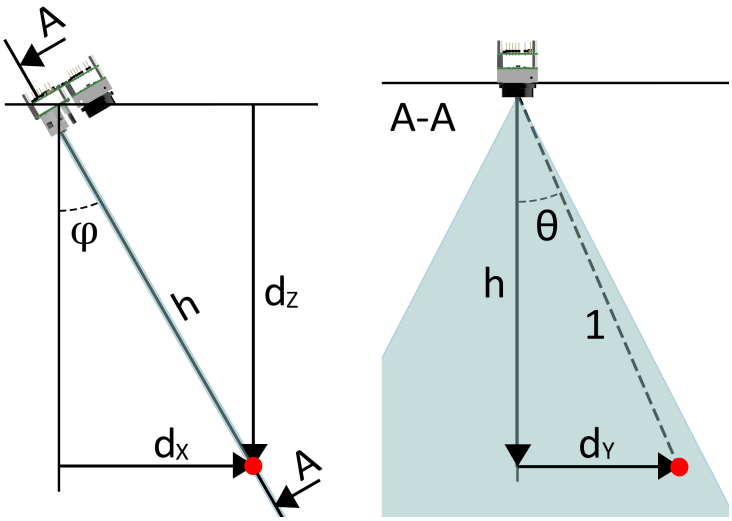


Figura 6.3: La direzione del raggio nel sistema di riferimento del drone.

Ricorrendo alle già usate matrici di rotazione tridimensionale (capitolo 4), ottengo

$$dir_x = d_X(cp \cdot cy) + d_Y(-cp \cdot sy) + d_Z(sp)$$

$$dir_y = d_X(sr \cdot sp \cdot cy + cr \cdot sy) + d_Y(-sr \cdot sp \cdot sy + cr \cdot cy) + d_Z(-sr \cdot cp)$$

$$dir_z = -[d_X(-cr \cdot sp \cdot cy + sr \cdot sy) + d_Y(cr \cdot sp \cdot sy + sr \cdot cy) + d_Z(cr \cdot cp)]$$

Una volta che si è a conoscenza di un raggio, si tenta di intersecarlo con tutti i triangoli della mesh. Ho derivato la funzione di intersezione da una fonte online [24] contenente anche la spiegazione teorica matematica.
Implementata nel mio algoritmo, per ogni coppia raggio-triangolo, fornisce una distanza dal sensore virtuale. In caso di intersezione di un singolo raggio con più triangoli, viene presa in considerazione la distanza più piccola, per simulare l'ostruzione visiva del triangolo più vicino.
Conoscendo la posizione del LIDAR $\{Lon_L, Lat_L, Alt_L\}^T$, la direzione del raggio $\{dir_x, dir_y, dir_z\}^T$ e la distanza del punto di intersezione (t), si ottengono le coordinate $\{Lon_i, Lat_i, Alt_i\}^T$ di quest'ultimo con

$$\begin{Bmatrix} Lon_i \\ Lat_i \\ Alt_i \end{Bmatrix} = \begin{Bmatrix} Lon_L \\ Lat_L \\ Alt_L \end{Bmatrix} + \begin{Bmatrix} dir_x \\ dir_y \\ dir_z \end{Bmatrix} \cdot t$$

6.2 Errore del metodo non esatto

Volendo disegnare un grafico α - *err* per il *metodo rapido*, dove α è la pendenza del terreno, pongo $h = 30\ m$, valore supposto verosimile per la missione, e $\theta = 24°$, ovvero il suo valore massimo quando il drone è stazionario senza perturbazioni. Vedo l'errore crescere esponenzialmente all'aumentare della pendenza.

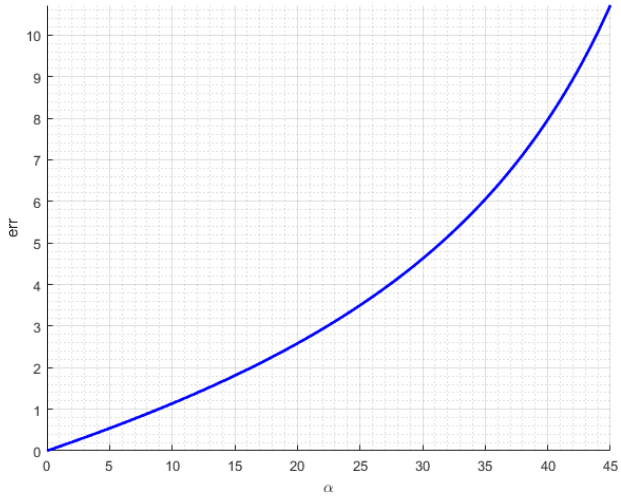


Figura 6.4: L'errore delle coordinate simulate al variare della pendenza del terreno.

Per quanto riguarda il *ray tracing*, l'accuratezza di ogni misurazione dipenderà dalla accuratezza di macchina e non dalla pendenza del terreno, dunque si può affermare che fornisce un risultato esatto.
Da queste considerazioni il ray tracing sembra il metodo vincente, ma sappiamo anche che è computazionalmente più pesante, il che comporta un aumento del tempo di calcolo, con uno svantaggio nella risoluzione della mappa DEM di partenza.
Per avere una mappa di estensione adatta alla nostra missione, devo considerare un quadrato dal lato minimo di un chilometro, altrimenti si rischia di sottostimare l'effettivo range operativo di un drone. Al tempo stesso avere una risoluzione con una dimensione di cella superiore a circa 3 metri non consente di avere un dettaglio della superficie sufficiente a simulare una situazione nella quale si trovi un ostacolo di dimensioni che interessano a questo studio sulla traiettoria del drone[1]. Questi due fattori impongono una mesh del terreno con almeno 200 elementi per lato, producendo un totale di circa 40000 celle quadrilatere o 80000 triangolari come minimo.
Il grafico seguente rappresenta i tempi che i due metodi impiegano in media per trovare le coordinate e le quote di otto punti del terreno, in funzione della dimensione di cella[2].
Sono stati trascurati i tempi *una tantum* di creazione della mesh e di lettura della mesh che per il ray tracing impiegano alcuni secondi.

[1] idealmente sarebbe augurabile una dimensione di cella decisamente inferiore, in quanto il sistema di navigazione avrebbe accesso a dati molto più dettagliati e potrebbe implementare algoritmi con movimenti più agili

[2] La simulazione è stata fatta girare in c++ su Microsoft Visual Studio e su una macchina virtuale Ubuntu con risultati simili. Ho utilizzato un computer con processore i7 del 2014 e 6GB di RAM.

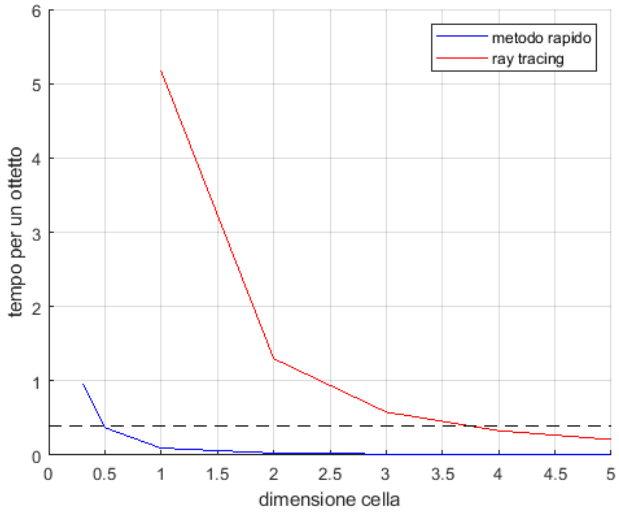


Figura 6.5: Il tempo di acquisizione in dipendenza della dimensione di cella.

Se impongo il tempo massimo di acquisizione a 0.4 secondi, un tempo che garantisce di avere in ogni occasione due misurazioni al secondo, l'errore del *metodo rapido* risulta quello riportato in figura 6.4. Questo va sommato all'approssimazione sul terreno reale dovuta ad una griglia DEM con lato 0.4 metri. L'errore del ray tracing, invece, è intrinsecamente nullo, ma va considerato l'errore indotto dal fatto che, per rientrare nei tempi, il terreno reale va approssimato con una mesh dal lato ben maggiore (3.7 metri). In un terreno frastagliato questa approssimazione può essere distante dalla realtà.
Noto quindi che fino a pendenze basse il *metodo rapido* risulta potenzialmente più accurato del ray tracing, in quanto si riferisce a dati mesh più dettagliati. Per casi di terreno pendente, conviene ricorrere al ray tracing.

Confronto le tracce lasciate dai due metodi in un primo caso su un terreno piano, in cui mi aspetto di avere le soluzioni coincidenti ed esatte

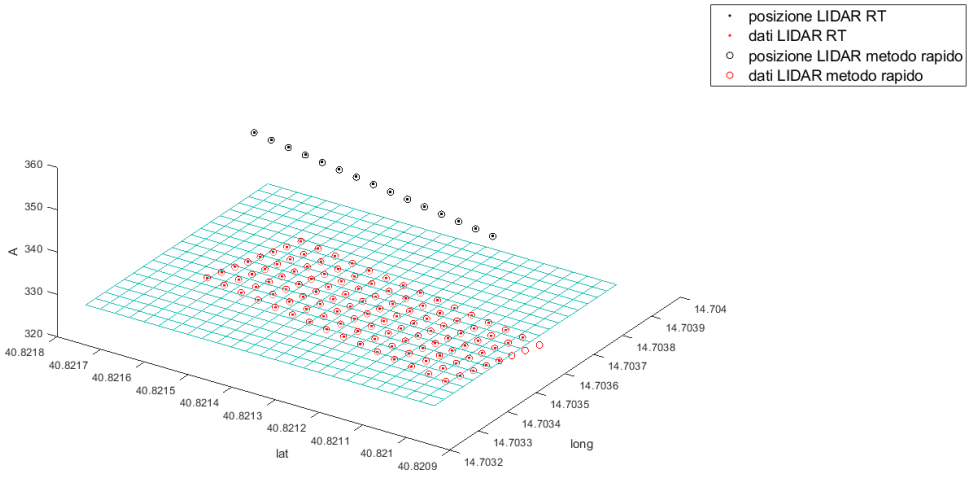


Figura 6.6: Le tracce su terreno piano.

e poi su terreno ripido, dove invece gli errori del *metodo rapido* dovrebbero emergere e il ray tracing rimane esatto in quanto il terreno non è frastagliato.

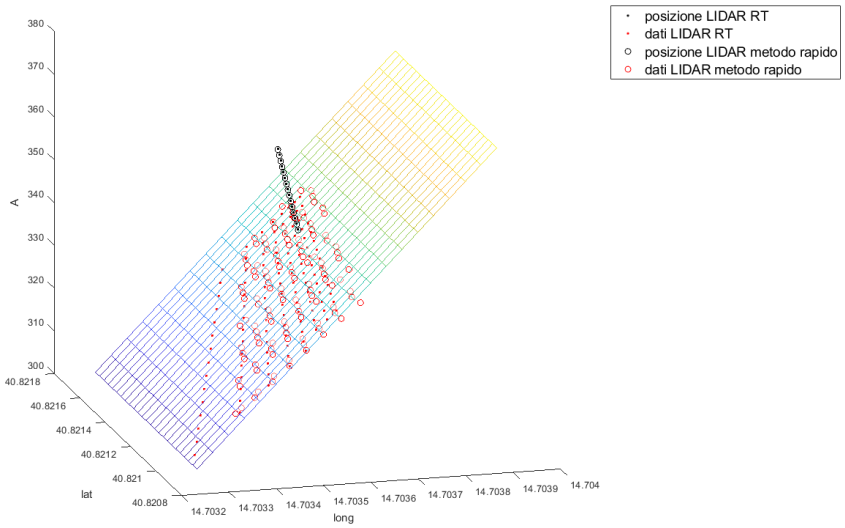


Figura 6.7: Le tracce su terreno inclinato.

Come già discusso, per il terreno piano vediamo che le tracce sono molto più simili, disegnando i punti pressoché coincidenti per ogni misurazione.
Al contrario, nel caso di terreno inclinato, vediamo che la traccia del *metodo rapido* si distribuisce innaturalmente più a destra della traiettoria, sbagliando nel non considerare che i raggi che colpiscono più in alto si allontanano meno dalla verticale del LIDAR; nel ray tracing, in questo caso soluzione esatta, vediamo la traccia più lontana dalla verticale del drone nelle zone più basse.
Il calcolo dell'errore commesso con il *metodo rapido* è stato validato confrontando i suoi risultati con la soluzione esatta.

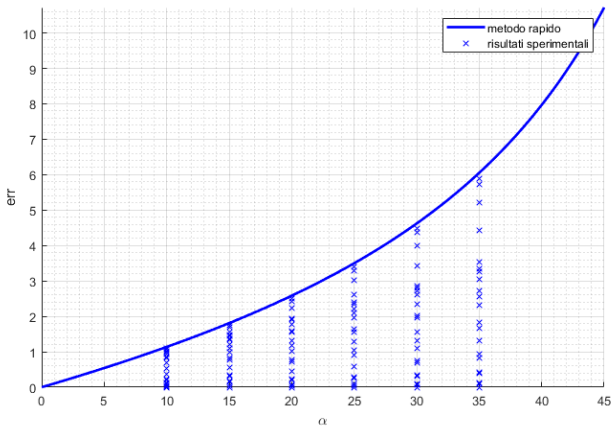


Figura 6.8: I dati sperimentali per la validazione dell'accuratezza del *metodo rapido*.

I risultati che ho ottenuto con i due metodi non hanno prodotto delle accuratezze soddi-

sfacenti, dunque ho deciso di ottimizzare il ray tracing, fino a svilupparne una versione che possa operare su una mesh molto dettagliata in un tempo breve.

6.3 Ottimizzazione del ray tracing

Il ray tracing si può considerare esatto, ma permane il problema dei lunghi tempi di acquisizione per mesh composte da tanti elementi.
Per una mesh di un'area con lato 1 km e celle quadrate di 5 metri, senza nessun tipo di ottimizzazione, si ha un tempo di calcolo per un ottetto di dati pari a circa 0.47 secondi[3]. Se diminuisco la dimensione delle celle a 2.5 metri i tempi aumentano a circa 1.75 secondi. Ho ritenuto opportuno implementare delle ottimizzazioni al ray tracing in modo da velocizzare il processo di acquisizione dei dati.

6.3.1 Massima distanza planare

L'ottimizzazione più semplice da implementare è un controllo sulla distanza di ogni triangolo della mesh prima di verificarne l'intersezione. Si impone una distanza massima dal drone sul piano di latitudine e longitudine all'interno della quale applicare effettivamente il ray tracing.

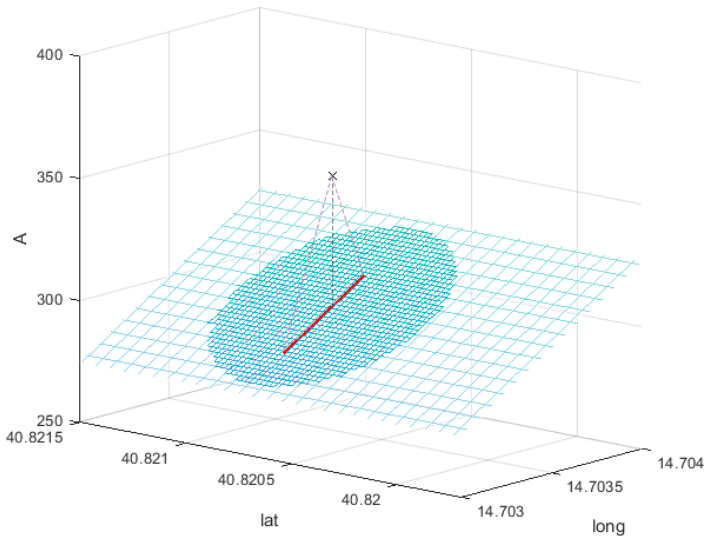


Figura 6.9: La porzione di mesh nella quale si applica il ray tracing.

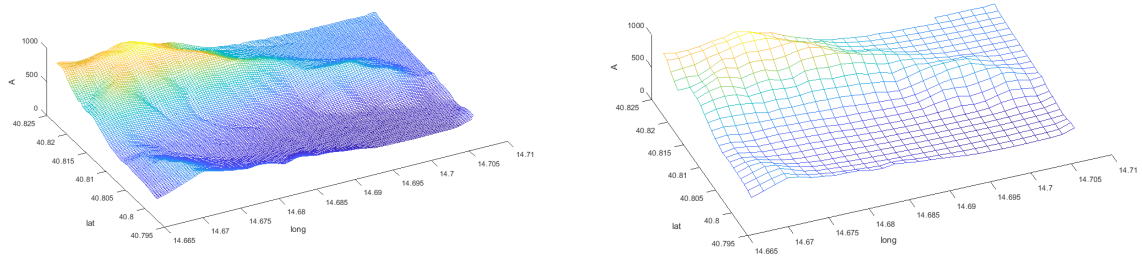
Il controllo avviene su ogni triangolo: quando l'elemento rientra nella distanza limite, viene aggiunto alla lista dei tentativi di intersezione con il raggio; in caso contrario non

[3]La simulazione è stata fatta girare in c++ su Microsoft Visual Studio e su una macchina virtuale Ubuntu con risultati simili. Ho utilizzato un computer con processore i7 del 2014 e 6GB di RAM.

verrà intersecato.

6.3.2 Mesh *frattale*

L'idea alla base di questo metodo consiste nel creare due mesh del terreno DEM di partenza: una al massimo livello di dettaglio, ovvero considerando tutti i punti DEM; una a risoluzione minore, che chiamerò *mesh semplificata*, considerando come nodo, ad esempio, un punto ogni cinque in entrambe le direzioni della griglia.



(a) Non semplificata.

(b) Semplificata.

Figura 6.10: Due mesh a diverso livello di dettaglio.

La prima scansione del ray tracing avviene sulla mesh semplificata, che contiene un numero di triangoli molto minore rispetto alla mesh intera. Di seguito, intorno ad ogni punto tracciato, viene ritagliata una piccola porzione della mesh più dettagliata: su questa nuova *mini-mesh* di ritaglio si ripete il ray tracing.

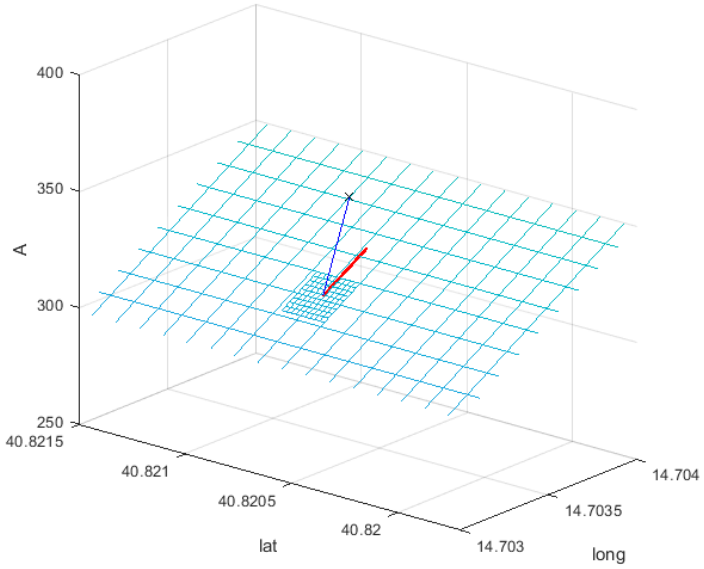


Figura 6.11: *Mini-mesh* creata per ogni echo sulla mesh semplificata.

6.3.3 Validazione

Ho testato gli algoritmi su un piano inclinato, confrontandone i risultati con le intersezioni geometriche teoriche tra le linee uscenti dal LIDAR virtuale e il piano. I risultati di questa verifica sono visibili nella figura seguente, dove le linee rosse connettono i punti individuati dal ray tracing, i punti neri sono i dati teorici e le `x` sono le posizioni del LIDAR.

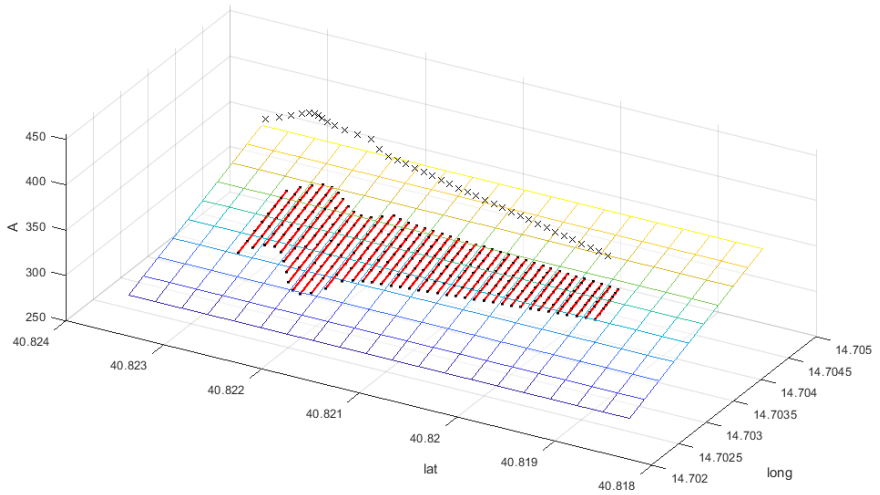


Figura 6.12: La corrispondenza tra il LIDAR ray tracing e i risultati teorici.

I punti cadono esattamente sulle linee: la distanza scalare media tra i punti teorici e i punti ottenuti con il ray tracing è di circa $4 \cdot 10^{-4}$, ovvero l'accuratezza ottenibile con le

cifre significative imposte ai dati che l'algoritmo produce.

6.3.4 Risultati

Massima distanza planare

Ponendo la dimensione delle celle a 2.5 metri e una distanza di controllo della mesh a 50 metri dalla posizione del drone, il tempo di acquisizione di un ottetto ammonta a circa 0.77 secondi: con tale dimensione di cella, senza ottimizzazioni impiegava circa 1.75 secondi.
I problemi di questa ottimizzazione sono due: i tempi di acquisizione ancora non permettono le due misurazioni al secondo prefissate, e non si può essere certi che i punti da rilevare siano effettivamente all'interno della zona scansionata. Nella figura seguente un angolo di beccheggio di 45° fa finire fuori dal raggio di rilevazione la maggior parte degli echoes, che quindi non verranno acquisiti.

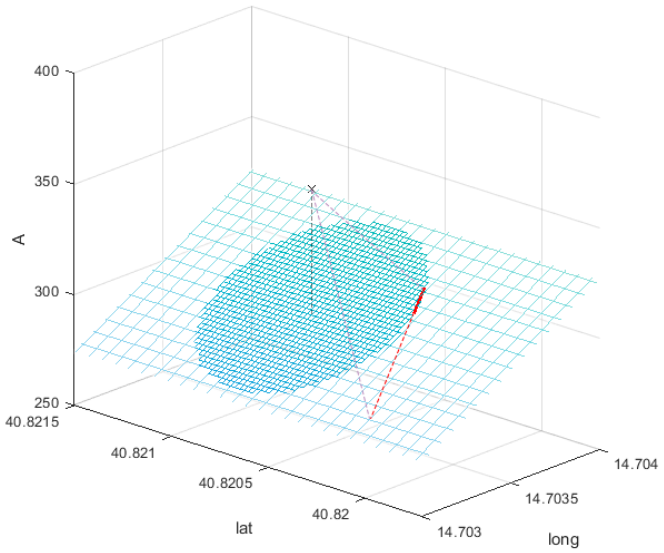


Figura 6.13: Con angoli di assetto alti gli echoes escono dal raggio di azione.

Inoltre, in caso di elevata pendenza, la situazione peggiora, in quanto gli echoes nella direzione della discesa saranno molto più lontani dal LIDAR.
Per evitare ciò, avrei potuto semplicemente ingrandire il raggio di azione, facendo però venire meno il grande vantaggio computazionale di dover scansionare su una mesh piccola.

Mesh *frattale*

Questo tipo di ottimizzazione permette all'algoritmo di impiegare circa 0.10 secondi per ogni ottetto. I tempi sono ricavati nelle stesse condizioni citate in precedenza, con mesh di

partenza 400x400 e celle di 2.5 metri, e una mesh semplificata con fattore 5, ovvero 80x80 con celle di 12.5 metri. La somma dei tempi per le otto scansioni della mesh semplificata 80x80 e delle otto mini-mesh 10x10 ammonta a circa 0.19 secondi per ogni ottetto, ovvero il 10.9% del ray tracing non ottimizzato e il 24.7% dell'ottimizzazione a massima distanza planare.
Il limite di questa ottimizzazione si incontra in mappe molto frastagliate.

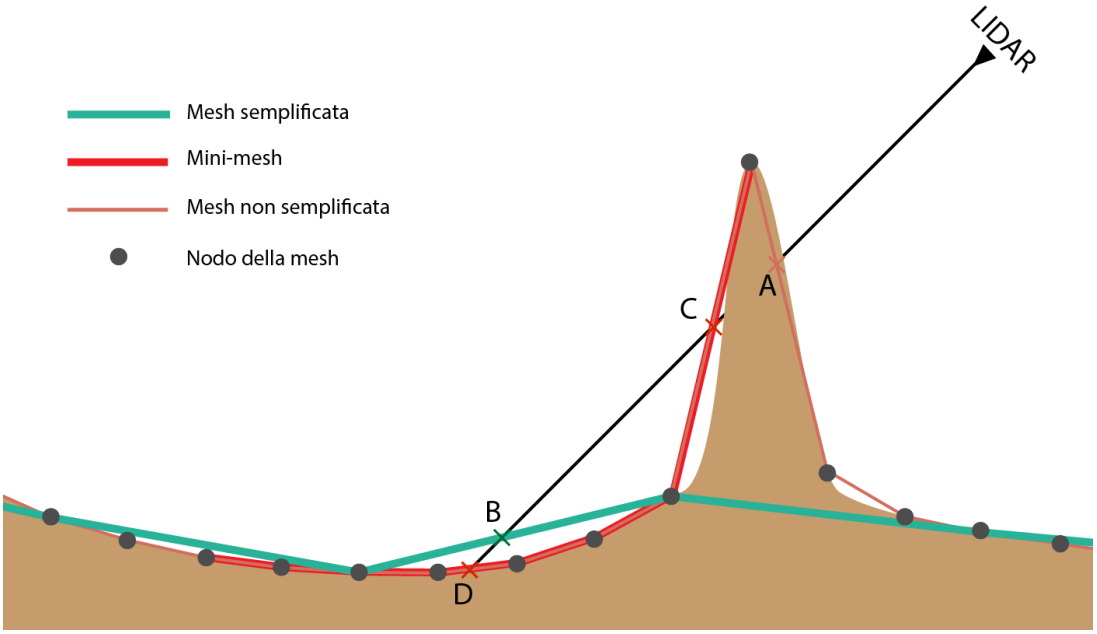


Figura 6.14: L'errore generato sulla mappa frastagliata.

Il ray tracing non ottimizzato individuerebbe le intersezioni con la mesh semplificata in A, C e D, scegliendo come punto finale A, in quanto è il più vicino dei tre.
Secondo il procedimento descritto, l'algoritmo ottimizzato individua il punto B scansionando la mesh semplificata, e intorno ad esso costruisce la *mini-mesh*. Al momento di intersecare il raggio con questa, troverà solamente i punti C e D, poiché A è situato fuori da essa. Il punto finale risulta erroneamente C.
Una soluzione è aumentare il lato della *mini-mesh*, ma, come già detto in precedenza, bisogna fare i conti con l'aumento dei tempi di acquisizione.

Capitolo 7

Conclusione

Per concludere non posso fare a meno che citare nuovamente la tesi di Sara Senzacqua [5] in quanto i risultati finali non possono prescinderne.
Grazie al nostro lavoro congiunto, abbiamo fatto volare il drone virtuale su una mappa da noi costruita (figura 7.1) in modo da avere ostacoli sulla traiettoria. Con il sensore LIDAR simulato puntato in avanti a 45°, abbiamo acquisito i dati durante tutto il volo, che è consistito in due giri sopra la mappa (figura 7.2), sotto forma di nuvola di punti. Il primo giro ad una altitudine superiore agli ostacoli, in modo da acquisire dati del terreno ma non influenzare la traiettoria del drone, il secondo che, attraverso i dati acquisiti, evitasse gli ostacoli che si stagliavano davanti ad esso.
Una volta in possesso dei dati del terreno sotto forma di nuvola di punti, abbiamo ricostruito una mappa DEM (figura 7.3) confrontabile con quella originaria.

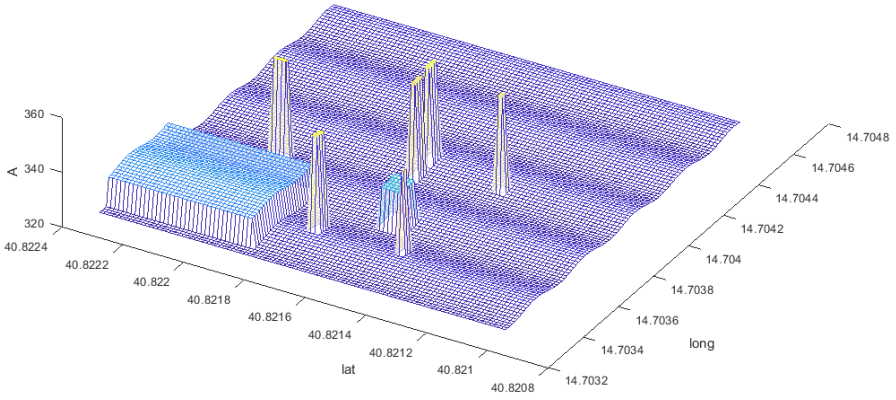


Figura 7.1: La mappa DEM popolata di ostacoli.

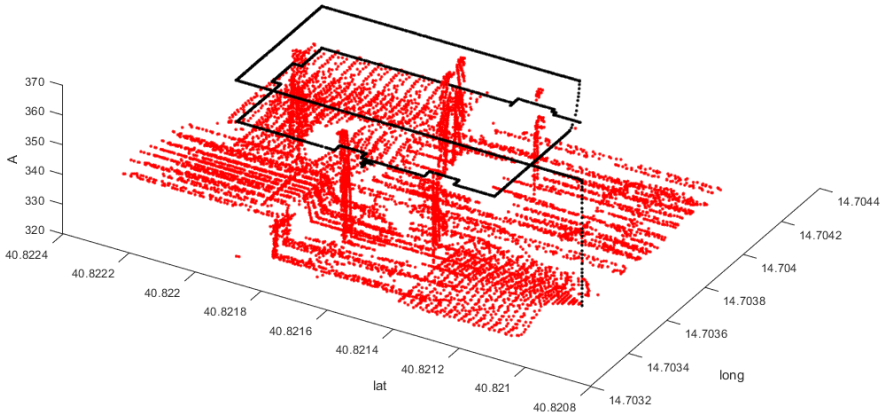


Figura 7.2: I punti acquisiti dal sensore LIDAR durante il volo.

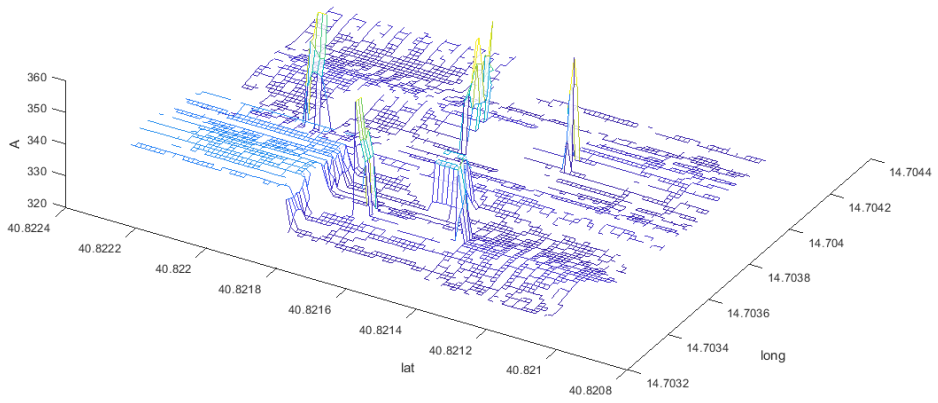


Figura 7.3: La mesh del terreno ricostruita a partire dai dati LIDAR.

Il sistema è in grado di navigare su un terreno incognito senza impattare con ostacoli che si trova sul cammino, restituendo una mappa del terreno dettagliata.
Gli algoritmi della simulazione sono stati pensati per essere facilmente adattabili ad un caso reale, in quanto gli input e gli output sono analoghi a quelli che sono forniti e richiesti dal sistema di controllo di un drone e dal sistema di navigazione. Non ci è stato possibile eseguire test sperimentali con strumentazione reale. Per effettuare il monitoraggio di incendi, il sistema dovrà implementare dei sensori per individuare le zone bruciate, un sistema di comunicazione, un sistema di controllo.
Inoltre è consigliabile montare sul drone altri sensori di posizione oltre al singolo LIDAR,

in quanto, con la disposizione attuale, esso non è adatto ad individuare ponti o strutture sospese all'altezza del velivolo.

Il software prodotto nella tesi può essere utilizzato per eseguire una mappatura digitale di qualsiasi terreno, e può risultare utile per tutte le situazioni in cui ci sia la necessità ma non la possibilità di fare volare un drone equipaggiato con sensore LIDAR.

Bibliografia

[1] EFFIS. [Online]. Available: https://effis.jrc.ec.europa.eu/static/effis.statistics.portal/effis-estimates/EU

[2] CNET. [Online]. Available: https://www.cnet.com/news/californias-fires-face-a-new-high-tech-foe-drones/

[3] Universidad Carlos III de Madrid. [Online]. Available: https://www.uc3m.es/ss/Satellite/UC3MInstitucional/en/Detalle/Comunicacion_C/1371271588512/1371216052687/Drones_for_early_detection_of_forest_fires

[4] Intellias Automotive. [Online]. Available: https://medium.com/@intellias/the-ultimate-sensor-battle-lidar-vs-radar-2ee0fb9de5da

[5] S. Senzacqua, "Pianificazione della traiettoria di un apr tramite mappatura digitale del terreno per il monitoraggio incendi." Politecnico di Torino, 2019.

[6] B. Schwarz, "Mapping the world in 3d," *Nature Photonics*, no. 4, pp. 429 – 430, 2010.

[7] C. S. W. Luke Wallace, Arko Lucieer, "Evaluating tree detection and segmentation routines on very high resolution uav lidar data," in *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52. IEEE, 2014, pp. 7619 – 7628.

[8] S. B. P. L. Jonathan Lisein, Marc Pierrot-Deseilligny, "A photogrammetric workflow for the creation of a forest canopy height model from small unmanned aerial system imagery," *Forests*, vol. 4, no. 4, pp. 922 – 944, 2013.

[9] G. G. P. D. J. M. A. Lefsky, W. B. Cohen and Harding, "Lidar remote sensing for ecosystem studies," *Bioscience*, vol. 52, no. 1, pp. 19 – 30, 2002.

[10] H. L. H. Michael Tulldahl, "Lidar on small uav for 3d mapping," in *Electro-Optical Remote Sensing, Photonic Technologies, and Applications VIII; and Military Applications in Hyperspectral Imaging and High Spatial Resolution Sensing II*, G. Kamerman, O. Steinvall, G. J. Bishop, A. Killey, and J. D. Gonglewski, Eds., vol. 9250, International Society for Optics and Photonics. SPIE, 2014, pp. 28 – 41.

[11] N. Michael, S. Shen, K. Mohta, Y. Mulgaonkar, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro, "Collaborative mapping of an earthquake-damaged building via ground and aerial robots," *Journal of Field Robotics*, vol. 29, no. 5, pp. 832 – 841, 2012.

[12] A. F. Elaksher, S. Bhandari, C. A. Carreon-Limones, and R. Lauf, "Potential of uav lidar systems for geospatial mapping," in *Lidar Remote Sensing for Environmental Monitoring 2017*, U. N. Singh, Ed., vol. 10406, International Society for Optics and Photonics. SPIE, 2017, pp. 121 – 133.

[13] C. Carreon-Limones, A. Rashid, P. Chung, and S. Bhandari, *3-D Mapping using LIDAR and Autonomous Unmanned Aerial Vehicle*, 2017.

[14] J. Teng, J. Vaze, D. Dutta, and S. Marvanek, "Rapid inundation modelling in large floodplains using lidar dem," *Water Resources Management*, vol. 29, no. 8, pp. 2619 – 2636, 2015.

[15] LeddarTech. [Online]. Available: https://leddartech.com/lidar/leddar-vu8-solid-state-lidar-sensor-module/

[16] S.-H. . L. I.-P. . C. K.-A. Kim, Seong-Joon ; Min, "Geometric modeling and data simulation of an airborne lidar system," *Journal of the Korean Society of Surveying, Geodesy, Photogrammetry and Cartography*, vol. 26, no. 3, pp. 311 – 320, 2008.

[17] Scratchapixel. [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-polygon-mesh

[18] T. Opsahl, T. V. Haavardsholm, and I. Winjum, "Real-time georeferencing for an airborne hyperspectral imaging system," in *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XVII*, S. S. Shen and P. E. Lewis, Eds., vol. 8048, International Society for Optics and Photonics. SPIE, 2011, pp. 290 – 295.

[19] S. Parkes, M. Dunstan, I. Martin, P. Mendham, and S. Mancuso, "Planet surface simulation for testing vision-based autonomous planetary landers," in *57th International Astronautical Congress, IAC 2006*, vol. 3. American Institute of Aeronautics and Astronautics, 2006, pp. 1512 – 1520.

[20] LeddarTech. [Online]. Available: https://github.com/leddartech/LeddarSDK.git

[21] LeddarTech. [Online]. Available: https://usermanual.wiki/Document/leddarvuandconfiguratoruserguide.473841576/html

[22] LeddarTech. [Online]. Available: https://leddartech.com/app/uploads/dlm_uploads/2017/05/Spec-Sheets-LeddarVu-4decembre2017-web.pdf

[23] Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Rotation_matrix#In_three_dimensions

[24] Scratchapixel. [Online]. Available: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/ray-triangle-intersection-geometric-solution