UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Ingegneria Industriale DII

Corso di Laurea Magistrale in Ingegneria Aerospaziale

Implementazione e validazione di metodi di mappatura 3D basati su sensori LiDAR per la navigazione autonoma di rover

Relatore:
Ing. Chiodini Sebastiano

Laureando:
Volpin Filippo

Anno accademico 2022-2023

Sommario

Questa tesi mira all'implementazione di una libreria di mappatura volumetrica su Morpheus, un Rover mobile ideato e realizzato dall'univeristà di Padova. Il rover Morpheus è nato per testare tecnologie per la navigazione in ambienti non strutturati, in particolare per l'esplorazione marziana e lunare. É quindi equipaggiato con limitate risorse computazionali, per questo motivo viene utilizzata una libreria chiamata VoxBlox che è in grado di funzionare utilizzando solo la CPU, sfruttando in questo modo risorse limitate.
VoxBlox utlizza il metodo *Truncated Signed Distance Fields* (TSDFs) che rappresenta lo spazio 3D basandosi su voxel e consente una mappatura densa con un'alta qualità superficiale e una robustezza al rumore del sensore, rendendolo un buon candidato per l'uso in scenari destinati all'esplorazione planetaria. Inoltre al TSDFs vi è collegato un metodo che definisce ulteriori informazioni di distanza (euclidea) degli ostacoli rilevati, chiamato *Euclidean Signed Distance Fields* (ESDFs).
Alla fine di questo elaborato si sono caratterizzate le performance di VoxBlox in modo tale da dimostrarne l'efficacia per effettuare una mappa occupazionale in tempo reale e quindi essere usato per effettuare una pianificazione autonoma della traiettoria anche in ambienti planetari inesplorati.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Da sempre l'uomo si è dimostrato curioso di conoscere tutto ciò che è ignoto, analizzando ed esplorando tutto quello che lo circonda. da sempre è alla ricerca di qualcosa di nuovo da scoprire e prova a raggiungere i luoghi più remoti. Negli ultimi anni, è aumentato sempre di più l'interesse nell'esplorazione planetaria, partendo dallo studio della Luna e degli Asteroidi fino all'analizzare ogni pianeta abitabile dall'uomo ed particolar modo il Pianeta Rosso.
Per poter esplorare questi luoghi ed analizzarli, è stato necessario realizzare rover in grado di navigare autonomamente in questi ambienti sconosciuti.

La maggior parte delle missioni di esplorazione planetaria è avvenuta proprio sul Pianeta Rosso: Marte, con molte missioni di successo (mostrate in Figura 1.1). La prima missione fu 25 anni fa con il rover Sojourner (4 luglio - 27 settembre 1997), lo seguirono Spirit (2004-2010), Opportunity (2004-2019), Curiosity (2012-in corso), Perseverance (2020-in corso) ed infine Zhurong (2021-in corso). Il focus principale di queste missioni era, e continua ad essere quello di esplorare la superficie e la geologia di Marte per scoprirne la storia dell'attività idrica del pianeta, determinarne i processi geologici che hanno plasmato il paesaggio e valutarne l'abitabilità per l'uomo.

La notevole distanza tra la terra e Marte non permette il controllo dei rover in tempo reale, infatti un segnale di sola andata può impiegare fino a 20 minuti per raggiungere Marte. Per questo motivo i rover devono essere dotati di una autonomia sempre maggiore per poter raggiungere più velocemente gli obbiettivi della missione. Nonostante ai rover venga data sempre più autonomia, nella maggior parte dei casi la pianificazione di obbiettivi e le analisi dei rischi continuano ad essere gestite dagli operatori di terra, in questo modo il tempo trascorso per raggiungere un obiettivo aumenta. Per questo motivo è necessario trovare metodi sempre più efficienti per navigare in ambienti sconosciuti, aumentando ulteriormente l'autonomia .

Per permettere ad un rover di navigare in autonomia innanzitutto è necessario che sia in grado di mappare l'area che lo circonda, identificando ostacoli "superabili", come piccole rocce e ostacoli più grandi che devono essere aggirati. Questo viene effettuato grazie ad algoritmi chiamati SLAM (*Simultaneous*

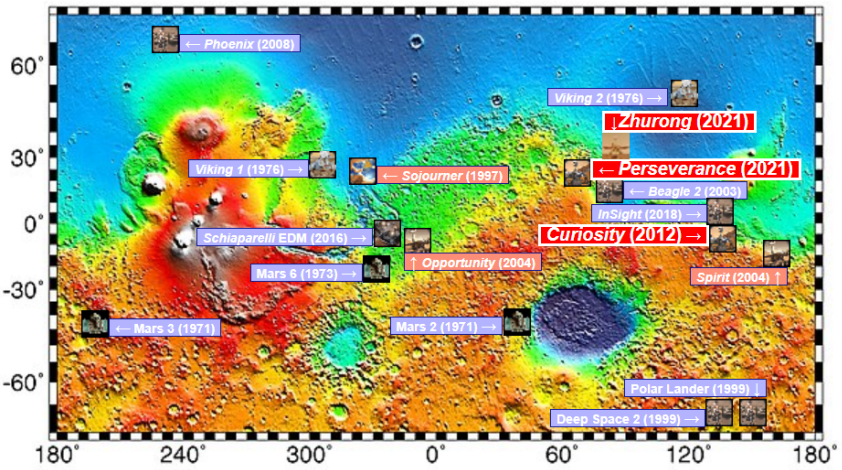


Figura 1.1: Siti di atterraggio di Lander (*robot "fissi"*) e rover(*robot "mobili"*) sulla superficie Marte [1].

Localization And Mapping) vedi Capitolo 2.

Tali algoritmi sfruttano alcuni sensori, i principali utilizzati sono i LiDAR e le Stereo Camere. La differenza principale tra questi due sensori sono i costi e le risorse energetiche e computazionali. Le stereo camere infatti sono più economiche rispetto al LiDAR, tuttavia hanno bisogno di maggiori risorse del processore e della memoria RAM per elaborare le immagini. Le immagini acquisite vengono elaborate da un computer attraverso una matrice di numeri in cui ogni elemento corrisponde direttamente all'intensità di ogni singolo *pixel*. Questi valori inoltre dipendono da un processo di formazione che coinvolge molti parametri e fonti di rumore. Per effettuare la SLAM con questo tipo di sensori, il pc di bordo dovrà elaborare una grande quantità di dati; infatti più un immagine ha un'alta risoluzione (più *pixel*) più la matrice avrà un maggior numero di elementi maggiori saranno i dati da elaborare per ogni singola immagine, questo comporta l'utilizzo di risorse molto preziose in un sistema limitato come un rover.
Attraverso l'uso dei sensori LiDAR invece la gestione delle acquisizione è molto meno onerosa in quanto il sensore fornisce direttamente una nuvola di punti che può essere elaborata immediatamente per effettuare la SLAM o la creazione di una mappa 3D dell'ambiente.

Per questo motivo in questa tesi è stato proposto l'utilizzo del LiDAR per effettuare la SLAM e la mappatura 3D con lo scopo finale di essere implementati nella navigazione autonoma.

Per poter realizzare un algoritmo di navigazione autonoma è importante soffermarsi sulle caratteristiche che esso necessita in input per poter essere più efficace. Uno degli input essenziali è quello della ricostruzione dell'ambiente esplorato dal rover, questa mappa infatti può essere creata sfruttando vari metodi e caratteristiche.
In particolare quelle richieste dalla navigazione autonoma sono le seguenti:

- Valutare la presenza di ostacoli che possano creare una collisione con il rover sia nel punto che si vuole raggiungere che nella traiettoria ipotizzata per raggiungerlo;
- Valutare, una volta individuato un punto raggiungibile dal rover all'interno dell'ambiente "conosciuto" (gia mappato), la quantità volume che sarà possibile osservare in tale punto;
- Essere in grado di identificare la posizione di specifici *target*, come ad esempio particolari rocce, in modo tale da poter abilitare funzioni avanzate capaci di ricercarli in aree specifiche.

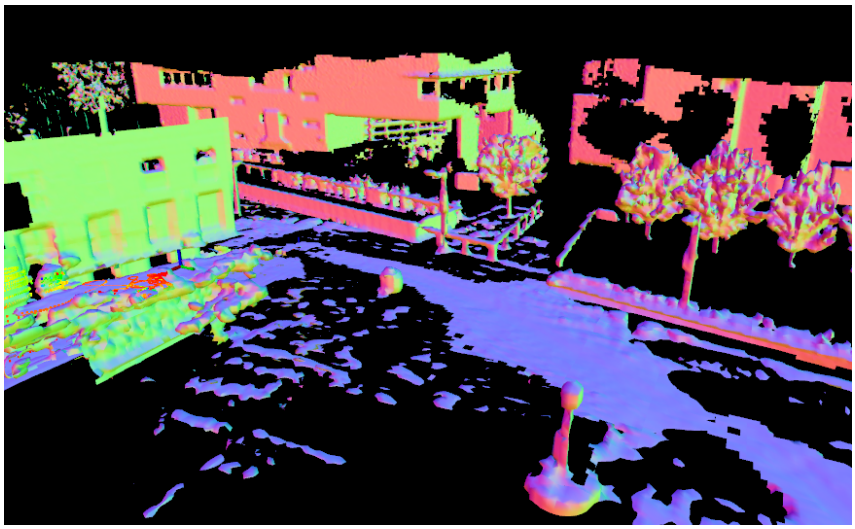


Figura 1.2: Esempio di una mesh ricreata con VoxBlox di un ambiente esterno.

Per raggiungere questi obbiettivi l'utilizzo di una mappa creata da una SLAM potrebbe non essere sufficiente in quanto questi algoritmi costruiscono una mappa sparsa, composta da punti nello spazio. É quindi necessario effettuare una ricostruzione 3D densa che sia efficace per l'utilizzo di un algoritmo di navigazione autonoma, per questa ragione è stata individuata e selezionata la libreria ***Voxblox*** [2].

Questa libreria è in grado di generare un modello tridimensionale dell'ambiente, sfruttando due "strumenti": gli *Euclidean Signed Distance Fields* (ESDFs) e i *Trucncated Signed Distance Fields* (TSDFs). Gli ESDF sono caratterizzati da una griglia di voxel dove ogni punto contiene le informazioni che riguardano la distanza euclidea dall'ostacolo più vicino, mentre i TSDFs sono griglie in cui gli elementi contengono il valore della distanza dall'ostacolo più vicino calcolato sulla distanza proiettata lungo il raggio del sensore, limitando però la distanza massima ad un valore di riferimento massima chiamato "distanza di troncamento".
La costruzione della TSDF è un processo piuttosto veloce ed è possibile sfruttarlo per realizzare una *mesh* dell'ambiente osservato, ovvero ricreare le superfici dell'ambiente tramite celle poligonali come mostrato in Figura 1.2. Questa

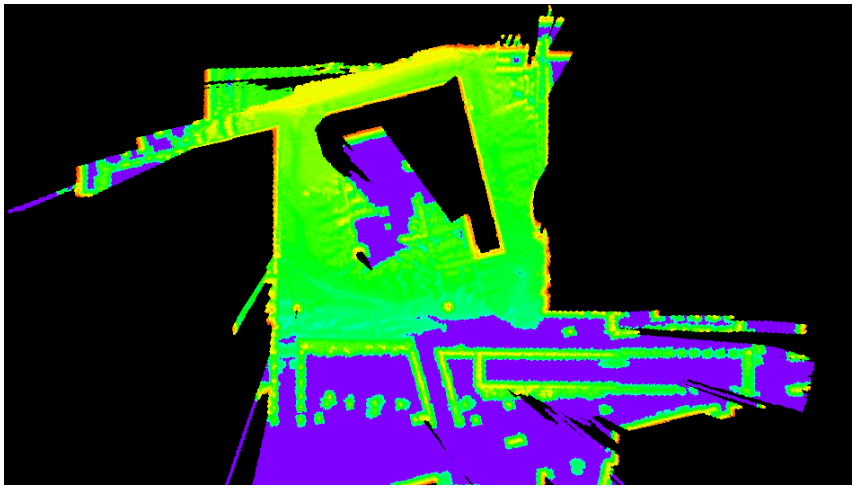


Figura 1.3: Esempio di una mappa 2D ESDF ad una quota di 0.2 [m] di una piazza all'interno dell'Università di Padova.

si potrebbe rilevare molto utile nei casi in cui sia necessario inviare la mappa infatti la mesh avrà come verrà mostrato in questo elaborato un peso piuttosto contenuto rispetto alle mappe composte dalla nucula di punti utilizzate nella SLAM. Un Esempio di una mappa ESDF 2D viene riportato in Figura 1.3, in questa mappa ogni punto ha un colore che identifica la sua distanza rispetto all'ostacolo più vicino, il colore giallo rosso sono quelli più vicini alla superficie rilevata.

Utilizzare Voxblox per la navigazione autonoma permette di ridurre il carico computazionale dell'algoritmo di navigazione in quanto il numero di dati da analizzare sulla mappa per verificare la fattibilità di una traiettoria si riduce notevolmente. Ad esempio, una volta individuata la traiettoria, nel caso di un utilizzo di una mappa occupazionale sarebbe necessario verificare tanti voxel quanti quelli presenti in una "sfera di controllo" (ovvero un area in cui si è sicuri che il rover sia in grado di passare senza imbattersi in un ostacolo), questo comporterebbe ad una grande quantità di dati da elaborare. Utilizzando una mappa ESDF invece basterà analizzare solo il centro dei voxel che sono nella traccia pianificata per poter conoscere se ad una certa distanza è presente o meno un ostacolo. In Figura 1.4 Vengono rappresentate le "sfere di controllo" nel caso delle due mappe.

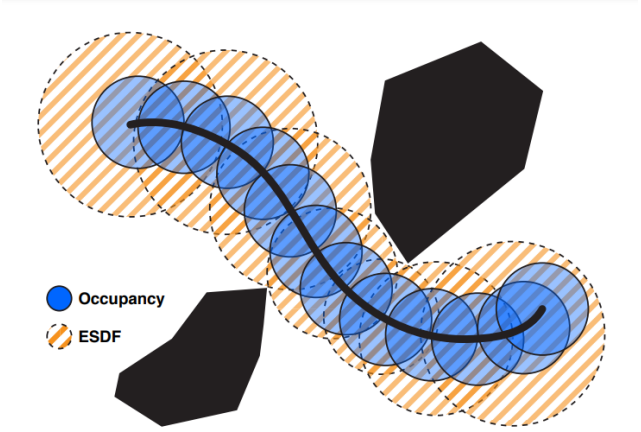


Figura 1.4: Un confronto tra il numero di Sfere "di controllo" necessarie per analizzare se una traiettoria è priva di ostacoli utilizzando una mappa ESDF (sfere arancioni) rispetto ad una occupazionale (sfere blu) [3].

1.1 Obbiettivi

Con il presente elaborato si è posto l'obbiettivo di implementare una libreria di mappatura volumetrica all'interno di un rover sviluppato da alcuni studenti dell'Università di Padova chiamato MORPHEUS (Sezione 4.1), con lo scopo di creare una mappa 3D in tempo reale un ambiente non strutturato in modo da poterlo utilizzare per la navigazione autonoma del rover stesso.
Obbiettivo indiretto: Per realizzare una mappa 3D è necessario prima effettuare una SLAM per conoscere la posizione del rover nell'ambiente, di conseguenza un ulteriore obbiettivo sarà quello di verificare che il percorso ricostruito dalla SLAM sia affidabile e paragonabile una traccia GPS.

Capitolo 2

Simultaneous Localization And Mapping: SLAM

Costruire una mappa tridimensionale densa (3D) di un ambiente in tempo reale e simultaneamente localizzarsi in tale mappa è l'obbiettivo principale della SLAM, ed è importante per poter rendere un robot autonomo nella navigazione in un ambiente sconosciuto in sicurezza. La localizzazione permette di avere un feedback per i controllo del rover e per il raggiungimento dell'obbiettivo, mentre la mappa fornisce importanti informazioni dell'ambiente circostante, come lo "spazio libero" e gli ostacoli, per pianificare la traiettoria.

Questo primo capitolo ha l'obiettivo di fornire un'introduzione per comprendere gli strumenti base del problema fornito dalla SLAM, dando un'idea complessiva dei fondamenti matematici e teorie all'interno dell'algoritmo che lo risolve.
I concetti che verranno riportati di seguito fanno riferimento a [4], [5] e [6].

Viene riportato di seguito un elenco di definizioni al fine di migliorare la comprensione delle Sezioni successive:

- Posa, è la posizione del rover definita rispetto ad una terna assoluta;
- *path*, è definito come l'evoluzione delle pose nel tempo;
- *features*, sono elementi o punti caratteristici individuati da un sensore in un determinato ambiente, viene descritto tramite le loro particolari caratteristiche ad esempio il suo colore o l'intensità;
- *landmark*, è un punto nello spazio, di solito individuato da una *feature* che viene considerato come riferimento e viene descritto attraverso la sua posizione relativa rispetto alla camera o al sistema assoluto;

2.1 Problema della SLAM

Considerando un robot mobile che si muove in un ambiente sconosciuto, questo acquisisce delle osservazioni che sono vari punti chiamati *lendmarks*, localizzati da un sensore come mostrato in Figura 2.1.
Ad ogni istante di tempo k vengono definite le seguenti quantità:

- $\boldsymbol{x}_k$: posa del veicolo, descrive la posizione e l'orientamento del veicolo
- $\boldsymbol{u}_k$: vettore di controllo applicato al tempo $k-1$ per guidare il veicolo alla posa $\boldsymbol{x}_k$ al tempo k
- $\boldsymbol{m}_i$: il vettore che descrive la posizione degli *"i" landmark* tale posizione è definita tempo variante (che quindi varia con il tempo);
- $\boldsymbol{z}_{it}$: un'osservazione presa dal veicolo della posizione degli *"i" landmarks* al tempo k.

Inoltre si definisce:

Storia della posizione del veicolo

$$\boldsymbol{X}_{0,k} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{k-1}, \boldsymbol{x}_k\} = \{\boldsymbol{X}_{0,k-1}, \boldsymbol{x}_k\}$$

Il set di tutti i landmark

$$\boldsymbol{U}_{0,k} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{k-1}, \boldsymbol{u}_k\} = \{\boldsymbol{U}_{0,k-1}, \boldsymbol{u}_k\}$$

Il set di tutte le osservazioni dei landmark

$$\boldsymbol{m}_{0,k} = \{\boldsymbol{m}_0, \boldsymbol{m}_1, ..., \boldsymbol{m}_n\}$$

Storia della posizione del veicolo

$$\boldsymbol{X}_{0,k} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{k-1}, \boldsymbol{x}_k\} = \{\boldsymbol{X}_{0,k-1}, \boldsymbol{x}_k\}$$

Nella formulazione probabilistica del problema della SLAM richiede che la distribuzione di probabilista :

$$bel(k) = P(\boldsymbol{X}_k, \boldsymbol{m} | \boldsymbol{Z}_{O,k}, \boldsymbol{U}_{0,k}, \boldsymbol{x}_0) \tag{2.1}$$

deve essere eseguita per ogni istante k. Questa distribuzione di probabilità descrive la posizione dei *landmarks* e le pose precedenti del veicolo date le registrazioni delle osservazioni precedenti, l'input dei controlli fino all'istante k assieme alla posizione iniziale del veicolo. In questa equazione inoltre è racchiuso sia il problema di localizzazione, ove la distribuzione di probabilità $P(\boldsymbol{X}_k | \boldsymbol{Z}_{O,k}, \boldsymbol{U}_{0,k,\boldsymbol{m}}, \boldsymbol{x}_0)$ assumendo nota la mappa m, sia il problema della mappatura, ove la distribuzione di probabilità $P(\boldsymbol{m} | \boldsymbol{Z}_{O,k}, \boldsymbol{U}_{0,k})$ assume come note le pose $\boldsymbol{X}_{0,k}$.

La soluzione è generalmente ricorsiva, ovvero nel primo step viene stimata la posizione, attraverso l'informazione del controllo ricevuto, mentre nel secondo viene migliorata la stima della posizione finale, includendo le misure di

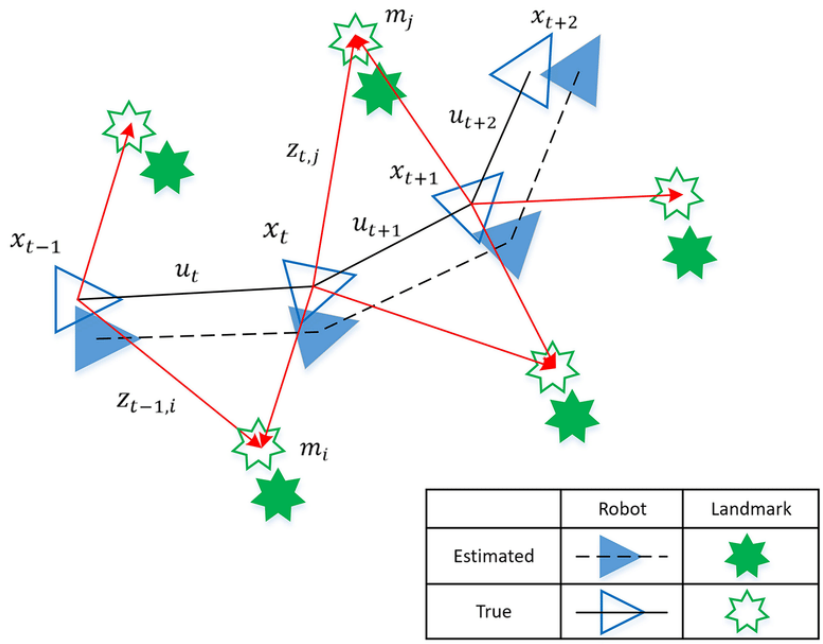


Figura 2.1: Schema con gli elementi principali che identificano il problema dalla SLAM [7].

distanza dei *landmarks* individuati.
Definendo la posa e le osservazioni al tempo k come:

$$\boldsymbol{x}_k = f(\boldsymbol{u}_k, \boldsymbol{x}_{k-1})$$
$$\boldsymbol{z}_k = g(\boldsymbol{x}_k, \boldsymbol{m})$$

Si possono definire così due importanti modelli di distribuzione probabilistica, rispettivamente quello del moto e quello delle misure di distanza (osservazioni) come segue:

$$P(\boldsymbol{x}_k|\boldsymbol{u}_k, \boldsymbol{x}_{k-1}) \tag{2.2a}$$
$$P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}) \tag{2.2b}$$

Utilizzando un processo di tipo *Markoviano*, ovvero considerando che le variabili al tempo k dipendano solo ed esclusivamente dalle variabili al tempo $k-1$, è possibile costruire il problema della SLAM in due passaggi ricorsivi, uno definito di Predizione e l'altro di Correzione.

Predizione: aggiornamento del tempo

$$\begin{aligned} \overline{bel}(k) &= P(\boldsymbol{x}_k, m|\boldsymbol{Z}_{O,k-1}, \boldsymbol{U}_{0,k-1}, \boldsymbol{x}_0) \\ &= \int P(\boldsymbol{x}_k|\boldsymbol{Z}_{O,k-1}, \boldsymbol{U}_{0,k-1}, \boldsymbol{x}_0) d\boldsymbol{x}_{k-1} \end{aligned} \tag{2.3}$$

Correzione: aggiornamento delle misure

$$bel(k) = \frac{\overline{bel}(k) \cdot P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m})}{P(\boldsymbol{z}_k|\boldsymbol{Z}_{O,k-1}, \boldsymbol{U}_{0,k-1})} \tag{2.4}$$

Nell'Equazione 2.3 utilizzando solamente un modello probabilistico del moto e dello stato precedente $bel(k-1)$ viene calcolato $\boldsymbol{x}_k$. La previsione dello stato al tempo k viene ricavata tramite il teorema della probabilità assoluta.

Attraverso l'utilizzo del teorema di Bayes nell'equazione 2.4 viene calcolato lo stato al tempo k aggiornando lo stato predetto in $\overline{bel}(k)$ con le misurazioni fatte al tempo k.

Gli errori sulla posizione dei *landmarks* in generale sono minori nelle distanze relative rispetto a quelle assolute e gli errori assoluti diminuiscono sempre più con l'aumentare delle osservazioni.
É possibile dimostrare questo fenomeno solo per il caso lineare con distribuzioni di tipo gaussiano [6], utilizzando un'analogia con un sistema costituito da una rete di molle come mostrato in Figura 2.2 . Ipotizzando di collegare ogni

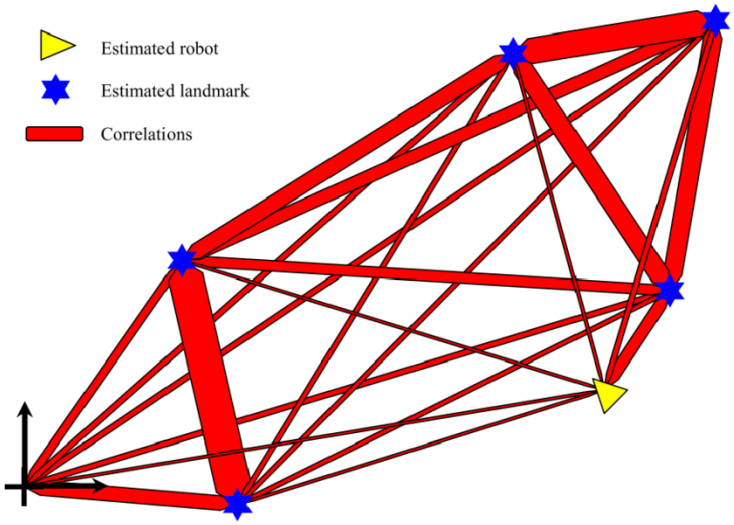


Figura 2.2: Analogia di correlazione tra i *landmarks* con una rete di molle [4]

landmarks tra di loro e con le pose che lo hanno individuato con una molla, si ha che la rigidezza della stessa è proporzionale all'accuratezza delle misure di posizione. Per ogni nuova osservazione che individua un determinato *landmark* viene aggiunto un nuovo vincolo ("*molla*") sulla posizione. Tale vincolo influenzerà la posizione assoluta del *landmark* e quella degli altri *landmarks* ad esso collegati, in base a quanto è forte la loro correlazione ("*rigidezza*"). All'aumentare dei nuovi vincoli, le correlazioni diventano sempre più forti; si hanno "*rigidezze*" più grandi, in questo modo viene ridotto l'errore di posizione dei *landmarks*.

2.2 Soluzione al problema della SLAM

Per poter risolvere il problema della SLAM è necessario formulare un appropriato modello probabilistico del moto e delle misure Equazioni 2.2.
Gli algoritmi di risoluzione più utilizzati sono l'EKF-SLAM (*Extended Kalman Filter*, Filtro di Kalman Esteso), la FastSLAM e il Graph-slam.
L'algoritmo di risoluzione che sfrutta un filtro di Kalman esteso (EKF) [8] è uno dei primi metodi utilizzati per la risoluzione del problema della SLAM. La sua semplicità di implementazione e le sue potenziali applicazioni in vari ambiti sono le caratteristiche che lo hanno reso il primo a risolvere il problema ed il più utilizzato.

Il filtro di Kalman viene usato formulando l'ipotesi che il modello di osservazione sia caratterizzato da una distribuzione dei rumori di misura con variabili Gaussiane indipendenti descritte dalle matrici di covarianza Q_k e R_k. Il modello al tempo k in accordo con le Equazioni 2.2 può essere definito come segue:

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-z}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{2.5a}$$
$$\boldsymbol{z}_k = h(\boldsymbol{x}_k, \boldsymbol{m}) + \boldsymbol{v}_k \tag{2.5b}$$

ove $f(\dots)$ e $h(\dots)$ sono rispettivamente le funzioni che descrivono rispettivamente il modello cinematico e il modello delle osservazioni (misure), mentre $\boldsymbol{w}_k$ e $\boldsymbol{v}_k$ sono i rumori gaussiani bianchi additivi associati alle matrici di covarianza $\boldsymbol{Q}_k$ e $\boldsymbol{R}_k$.
Attraverso l'uso di questi parametri e il metodo EKF è possibile calcolare il valore medio finale:

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{bmatrix} = E\left[\begin{bmatrix} \hat{\boldsymbol{x}}_k \\ \hat{\boldsymbol{m}} \end{bmatrix} \middle| \boldsymbol{Z}_{o:k} \right] \tag{2.6}$$

e la matrice di covarianza

$$\boldsymbol{P}_{k|k} = \begin{bmatrix} \boldsymbol{P}_x x & \boldsymbol{P}_x m \\ \boldsymbol{P}_x m & \boldsymbol{P}_m m \end{bmatrix} = \boldsymbol{E}\left[\begin{pmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \\ \boldsymbol{m}_k - \boldsymbol{x}\hat{\boldsymbol{m}}_k \end{pmatrix} \begin{pmatrix} \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k \\ \boldsymbol{m}_k - \hat{\boldsymbol{m}}_k \end{pmatrix}^T \middle| \boldsymbol{Z}_{o:k} \right] \tag{2.7}$$

riferite alla distribuzione di probabilità dell'Equazione 2.1.
Il calcolo avviene in due fasi:

- **Aggiornamento del tempo**

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k)$$
$$\boldsymbol{P}_{xx,k|k-1} = \nabla \boldsymbol{f} \boldsymbol{P}_{xx,k-1|k-1} \nabla \boldsymbol{f}^T + \boldsymbol{Q}_k$$

  dove $\nabla \boldsymbol{f}$ è lo Jacobiano di f valutato nel punto $\hat{\boldsymbol{x}}_{k-1|k-1}$. Come descritto nella sezione precedente, in maniera analoga questa fase corrisponde ad un passaggio di predizione in cui viene ipotizzata una distribuzione di probabilità basandosi solamente sul vettore di controllo($\boldsymbol{u}_k$).

- **Aggiornamento delle osservazioni**

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k-1} \\ \hat{\boldsymbol{m}}_k \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} & \hat{\boldsymbol{m}}_k \end{bmatrix} + W_k[\boldsymbol{Z}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1})]$$
$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} + \boldsymbol{W}_k \boldsymbol{S}_k \boldsymbol{W}^T{}_k$$

dove

$$\boldsymbol{S}_k = \nabla \boldsymbol{h} \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T + \boldsymbol{R}_k$$
$$\boldsymbol{W}_k = \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T \boldsymbol{S}^{-}1_k$$

dove $\nabla \boldsymbol{h}$ è lo Jacobiano di $\boldsymbol{h}$ valutato nel punto $\hat{\boldsymbol{x}}_{k|k-1}$. Questa fase corrisponde al passaggio di predizione in cui la posa e la mappa vengono aggiornate prendendo in considerazione le misure effettuate. La matrice $\boldsymbol{W}_k$ viene chiamata *"guadagno di Kalman"*, la sua funzione è quella di dare dare maggiore rilevanza alle misurazioni effettuate o alla stima del moto.

L'algoritmo basto sull'EKF-SLAM può essere applicabile anche a modelli non lineari, in quanto la linearizzazione avviene localmente. Se il modello pero è fortemente non lineare, possono insorgere problemi d'inconsistenza della soluzione. Infatti convergenza e consistenza sono garantite solo nel caso lineare come riportato in [6]. Di conseguenza più il modello si avvicina ad un modello lineare più è facile che si arrivi ad una convergenza della soluzione.

2.3 LiDAR-based SLAM

La maggior parte degli algoritmi SLAM che sfruttano i sensori LiDAR si basano su una struttura simile a quella del LOAM proposta in [9]. Sostanzialmente essa consiste in tre principali moduli: estrazione delle *features*, odometria e mappatura. Esistono inoltre altri algoritmi che stimano direttamente l' odometria tramite un confronto diretto dell'intera *pointcloud* in uscita dal LiDAR in due scansioni successive, come ad esempio l'algoritmo utilizzato in questo studio sperimentale:*FAST-LIO* (Capitolo 5.2.), questo processo comporta un aumento computazionale dovuto dalla grande quantità di dati da elaborare.
Tuttavia FAST-LIO utilizza un sistema basato sul kd-tree ma maggiormente sviluppato ed ottimizzato chiamato ikd-tree, capace di ridurre il carico computazionale. Il kd-tree è un metodo per immagazzinare i dati raccolti (nel caso della SLAM è la *pointcloud*) in uno schema ad albero, questo facilita l'accesso e la ricerca di particolari informazioni salvate (come la ricerca di corrispondenze tra due *pointcloud*).

Per poter dare al lettore una più facile comprensione di un tipico algoritmo LiDAR-based SLAM verrà esposto quello che è chiamato LOAM[9] in modo da fornire una panoramica del funzionamento dei vari step per la SLAM con i sensori LiDAR.

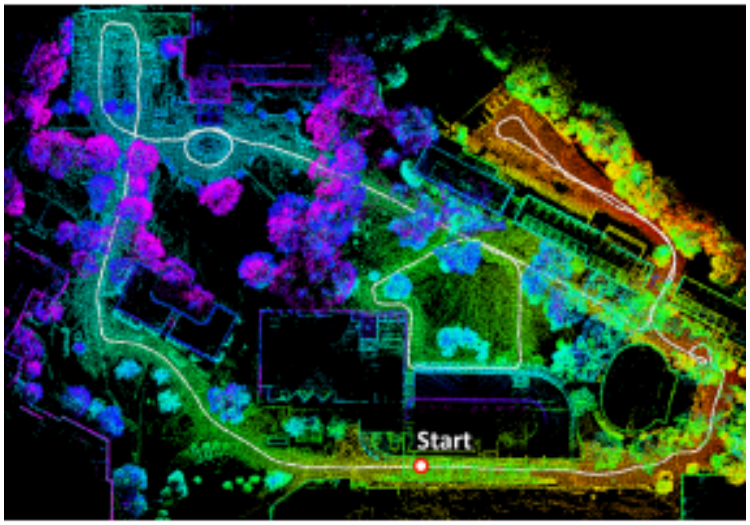


Figura 2.3: Esempio di LiDAR SLAM effettuata con un algoritmo LOAM [10].

La ricostruzione del percorso nel LOAM avviene effettuando un *matching* (trovando delle corrispondenze), tra le scansioni effettuate, ad alta frequenza e con un numero basso di punti nella *pointcloud*. Successivamente viene effettuata una stima del movimento tra ogni istante (*frame* ed il risultato ottenuto viene restituito all'algoritmo di mappatura. Questo secondo algoritmo effettua il *matching* e l'allineamento della *pointcloud* sulla mappa, tale processo avviene ad una frequenza di 1 Hz, utilizzando il metodo *scan-map matching* ([11] e[12]). Infine, la trasformazione delle pose create dai due algoritmi viene integrata per ottenere la trasformazione in output del LiDAR, con una frequenza di aggiornamento della mappa di almeno 10 Hz.

Nelle prossime sezioni si utilizzerà il simbolo P_k per indicare l'insieme della *pointcloud* ricevuta durante la scansione k, e verranno utilizzati i seguenti sistemi di riferimento:

- Il sistema di coordinate del LiDAR $\{L\}$ è definito in 3D con l'origine al centro geometrico del LiDAR. L'asse x punta verso sinistra, l'asse y punta verso l'alto e l'asse z punta in avanti. Le coordinate di un punto $i, i \in P_k$ in $\{L_k\}$ sono indicate con $X^L_{k,i}$
- Il sistema di coordinate "Mondo", anch'esso è definito in 3D e coincide con il Sistema $\{W\}$ alla posizione iniziale. Le coordinate di un punto $i, i \in P_k$ in $\{W_k\}$ sono indicate con $X^W_{k,i}$

Problema: Data una sequenza di punti fornita dal LiDAR $P_k, k \in Z^+$, calcolare il moto del LiDAR durante ogni scansione k e costruire una mappa con i punti P_k per definire l'ambiente percorso.

2.3.1 Ricerca e selezione delle *features*

Il processo inizia con la selezione ed l'estrazione di punti particolari chiamati *features* dalla nuvola di punti fornita in output dal LiDAR. La *features* sele-

zionate possono essere punti che descrivono spigoli o superfici piane [13]. Considerando i un punto in $p_k, i \in p_k$ e sia S un set di punti consecutivi che rappresentano i punti registrati dal LiDAR nella stessa scansione.

La seguente formula viene definita per valutare la "Curvatura" della superficie locale:

$$c = \frac{1}{|S| \cdot \|{X^L}_{K,i}\|} \| \sum_{j \in S, j \neq i} ({X^L}_{K,i} - {X^L}_{K,j}) \| \tag{2.8}$$

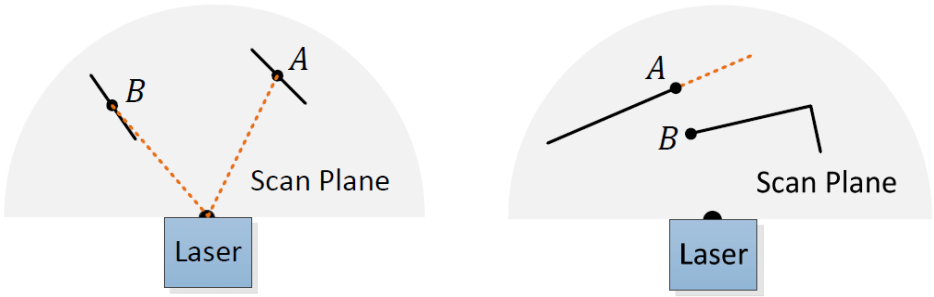
I punti in una scansione sono ordinati in base al valore di c, dopodiché tra le *features* che vengono selezionate si identificano come **punti spigolosi** le *feature* aventi il massimo valore di c, mentre i **punti piani** quelli aventi il valore di c minimo. Per distribuire ulteriormente le *features* nell'ambiente vengono separate le scansioni in 4 sottoregioni identiche. In ogni sotto-regione possono essere racchiusi al massimo 2 punti angolosi e 4 punti piani. Un punto i può essere identificato come planare o spigoloso solamente se è rispettivamente più piccolo o più grande di un valore di riferimento.

Riassumendo le *features* vengono selezionati con i seguenti criteri:

- Possono essere di due tipi, **punti spigolosi** selezionati in base al valore massimo di c e **punti piani** caratterizzati dal minimo valore di c;
- il numero di punti selezionati non può eccedere il numero di punti di una sotto-regione;
- non viene selezionato nessuno dei punti che circondano il punto selezionato;
- non vengono considerati i punti che si trovano su superfici circa parallele al raggio del laser e nemmeno quelli che rappresentano contorni di regioni "occluse" (non visibili); un esempio di questi punti è rappresentato in Figura 2.4.

2.3.2 Ricerca corrispondenze tra le *features*

L'algoritmo di **odometria** stima il moto del LiDAR ad ogni scansione. Sia t_k l'istante iniziale di una scansione(k). Alla fine di ogni scansione, la *pointcloud* acquisita dalla scansione (P_k) è riproiettata all'istante di tempo t_{k+1} si definisce tale *pointcloud* con $\bar{P}_k$ (Figura 2.5). Durante la scansione successiva $(k+1)$ la *pointcloud* $\bar{P}_k$ viene utilizzata assieme alla nuova *pointcloud* P_{k+1}, per effettuare la stima del moto del LiDAR.



(a) Punti piani (b) Punti spigolosi

Figura 2.4: In Figura (a) vengono selezionati i punti A e B che rappresentano superfici piane, tuttavia il punto B giace in una superficie quasi parallela al raggio laser inviato per trovare tale punto, di conseguenza questo punto viene considerato inaffidabile e non verrà selezionato tra le *features*.
Nell'immagine (b) invece vengono selezionati due punti angolosi ma essendo che il punto A è il contorno di un'area occlusa (linea tratteggiata), non viene selezionato [9].

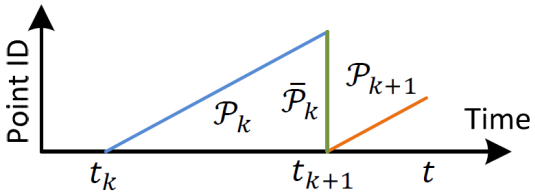


Figura 2.5: *Pointcloud* riproiettata al termine della scansione [9].

Assumendo note e disponibili le *pointcloud* P_k, P_{k+1}, si procede con la ricerca di corrispondenze tra le due *pointcloud*. Attraverso l'utilizzo del metodo descritto in precedenza vengono trovate e selezionate le *features*. Introducendo $\mathcal{E}_{k+1}$ e $\mathcal{H}_{k+1}$ rispettivamente il set di punti angolosi e planari, vengono quindi ricercate corrispondenze tra linee di bordo dalla *pointcolud* P_k ai punti in $\mathcal{E}_{k+1}$ e zone di superficie dei punti contenuti in $\mathcal{H}_{k+1}$. Si noti che all'inizio della scansione $k+1$ l'insieme di punti P_k è inizialmente vuoto, e si riempirà durante il corso della scansione. L'algoritmo stimerà ricorsivamente il movimento a 6 DOF (*Degree Of Freedom*) durante la scansione, e via via includerà più punti con il crescere di P_{k+1}. Ad ogni iterazione $\mathcal{E}_{k+1}$ e $\mathcal{H}_{k+1}$ vengo riproiettati all'inizio della scansione utilizzando la stima corrente della posa. Per ogni punto riproiettato definito con $\widetilde{\mathcal{E}}_{k+1}$ e $\widetilde{\mathcal{H}}_{k+1}$, viene ricercato un punto più vicino in $\bar{P}_k$. Per velocizzare la ricerca $\bar{P}_k$ viene "archiviato" all'interno di un 3D KD-tree.

Utilizzando le corrispondenze delle *features* trovate viene definita l'espressione che descrive la distanza da ogni *features* al suo *match* nella scansione successiva.

Per i punti angolosi $i \in \widetilde{\mathcal{E}}_{k+1}$, se (j, l) è il bordo corrispondente, $j, l \in \bar{P}_k$

la distanza punto linea è la seguente:

$$d_{\mathcal{E}} = \frac{\left|\left(\widetilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)}\right) \times \left(\widetilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,l)}\right)\right|}{\left|\left(\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right)\right|} \tag{2.9}$$

ove $\widetilde{X}^L_{(k+1,i)}$, $\bar{X}^L_{(k,j)}$ e $\bar{X}^L_{(k,l)}$ sono le coordinate dei punti i, j e l in {L}, rispettivamente.
Mentre la distanza di un punto piano $i \in \widetilde{\mathcal{H}}_{k+1}$ al suo piano corrispondente identificato da tre punti $j, l, m \in \bar{P}_k$ viene calcolata come:

$$d_H = \frac{\left|\begin{array}{c}\left(\widetilde{X}^L_{(k+1,i)} - \bar{X}^L_{(k,j)}\right) \\ \left(\left(\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right) \times \left(\bar{X}^L_{(k+1,j)} - \bar{X}^L_{(k,m)}\right)\right)\end{array}\right|}{\left|\left(\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,l)}\right) \times \left(\bar{X}^L_{(k,j)} - \bar{X}^L_{(k,m)}\right)\right|} \tag{2.10}$$

ove $\bar{X}^L_{(k,m)}$ sono le coordinate del punto m in {L}.

2.3.3 Stima del moto

Il movimento del LiDAR è basato su valori costanti di velocità angolari e lineari durante ogni scansione. Questo permette di interpolare linearmente la trasformazione della posa ad ogni scansione per i punti che vengono ricevuti in momenti diversi.
Si consideri t l'istante di tempo corrente e t_{k+1} è l'istante in cui inizia la scansione $k+1$ del LiDAR. T^L_{k+1} è definita come la matrice di trasformazione tra $[t_{k+1}, t]$. T^L_{k+1} è caratterizzata dal moto rigido del LiDAR in 6 DOF:

$$T^L_{k+1} = [t_x, t_y, t_z, \theta_x, \theta_y, \theta_z]^T$$

ove t_x, t_y e t_z sono le traslazioni lungo gli assi x, yez del sistema {L} e rispettivamente $\theta_x, \theta_y e \theta_z$ le rotazioni angolari, che seguono la regola della mano destra. Dato un punto $i, i \in P_{k+1}$ acquisito all'istante t_i e avente matrice di trasformazione $T^L_{k+1,i}$ tra gli istanti $[t_{k+1}, t_i]$. Tale matrice di trasformazione può essere calcolata attraverso un'interpolazione lineare di T^L_{k+1} come segue

$$T^L_{k+1,i} = \frac{t_i - t_{k+1}}{t - t_{k+1}} T^L_{k+1} \tag{2.11}$$

Per trovare il moto del LiDAR è necessario stabilire le relazioni geometriche tra le *features* selezionate alla scansione $k+1$ le loro riproiezioni sia per i punti angolosi che per quelli planari(rispettivamente$\mathcal{E}_{k+1}$ e $\mathcal{H}_{k+1}$ e $\widetilde{\mathcal{E}}_{k+1}$ $\widetilde{\mathcal{H}}_{k+1}$). Utilizzando l'Equazione 2.11 si ha:

$$X^L_{k+1,i} = R\widetilde{X}^L_{k+1,i} + T^L_{k+1,i}(1:3) \tag{2.12}$$

ove $X^L_{k+1,i}$ sono coordinate del punto i in $\mathcal{E}_{k+1}$ o $\mathcal{H}_{k+1}$, e $\widetilde{X}^L_{k+1,i}$ i punti corrispondenti in $\widetilde{\mathcal{E}}_{k+1}$ $\widetilde{H}_{k+1}$, $T^L_{k+1,i}(a:b)$ è l a-esimo o b-esimo elemento di del vettore infine R è la matrice di rotazione definita tramite la *Formula di Rodrigues* [14],

$$R = e^{\widehat{\omega}\theta} = I + \widehat{\omega}\sin(\theta)\widehat{\omega}(1-\cos(\theta)) \tag{2.13}$$

ove θ è la grandezza della rotazione, ω è un vettore che rappresenta la direzione di rotazione e $\widehat{\omega}$ è a matrice inversa di ω definiti come :

$$\theta = ||T^L_{k+1,i}(4:6)|| \qquad \omega = \frac{T^L_{k+1,i}(4:6)}{\|T^L_{k+1,i}(4:6)\|} \tag{2.14}$$

Combinando le Equazioni 2.9 e dalla 2.11 alla 2.14, è possibile ricavare la relazione geometrica tra i punti angolosi presenti in $\mathcal{E}_{k+1}$ e i corrispondenti bordi come

$$f_{\mathcal{E}}(X^L_{k+1,i}, T^L_{k+1}) = d_{\mathcal{E}} \qquad i, \in \mathcal{E}_{k+1} \tag{2.15}$$

In maniera analoga utilizzando l'Equazione 2.10 è possibile stabilire la relazione geometrica tra i punti piani ed il loro piano corrispondente

$$f_H(X^L_{k+1,i}, T^L_{k+1}) = d_H \qquad i, \in \mathcal{H}k+1 \tag{2.16}$$

Il moto del LiDAR viene risolto tramite il metodo *Levenberg-Marquardt* unendo le Equazioni 2.15 e 2.16 per ogni *feature* in $\mathcal{E}_{k+1}$ e $\mathcal{H}k+1$ si ottiene la seguente funzione non lineare:

$$f(T^L_{k+1}) = d \tag{2.17}$$

ove ogni riga di f corrisponde ad una *feature* e d contiene le distanze corrispondenti calcolate. Viene più calcolata la matrice Jacobiana di f rispetto a T^L_{k+1} indicandola con $J = \partial f / \partial T^L_{k+1}$. E risolvendo 2.17 tramite iterazioni non lineari minimizzando la distanza d a zero:

$$T^L_{k+1} \leftarrow T^L_{k+1} - (J^T J + \lambda diag(J^T J))^{-}1 J^T d \tag{2.18}$$

ove λ è il fattore determinato dal metodo *Levenberg-Marquardt*

2.3.4 Mappatura con LiDAR

L'algoritmo di mappatura proposto dal LOAM ha una bassa frequenza di aggiornamento rispetto a quello dell'odometria e viene richiamato solamente una volta per scansione. Il suo funzionamento parte alla fine di una scansione $k+1$ in cui il LiDAR genera una *pointcloud* non distorta P_{k+1} e simultaneamente una Posa T^L_{k+1} contenente il moto del LiDAR durante la scansione $[t_{k+1}, t_{k+2}]$. L'algoritmo di mappatura registra e trova la corrispondenza dei punti P_{k+1} nelle coordinate mondo $\{W\}$ come mostrato in Figura 2.6.

Per spiegare la procedura si introduce Q_k come la *pointcloud* nella mappa, accumulata dopo k scansioni e sia T^W_K la Posa assoluta del LiDAR nella mappa, a fine scansione k, t_{k+1}. Con l'output derivato dall'odometria l'algoritmo di

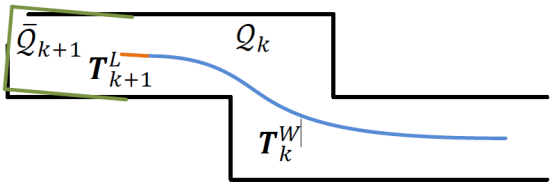


Figura 2.6: Rappresentazione del processo di mappatura del LOAM [9].

mappatura estende T_K^W per una scansione da $t_{t+1}t_{t+2}$, per ottenere T_{k+1}^W e proiettare $\bar{P}_{k+1}$ nelle coordinate mondo $\{W\}$ definite da $\bar{Q}_{k+1}$. Dopodiché l'algoritmo trova la corrispondenza dei punti $\bar{Q}_{k+1}$con quelliQ_{k+1} ottimizzando la posa del LiDAR T_{k+1}^W.

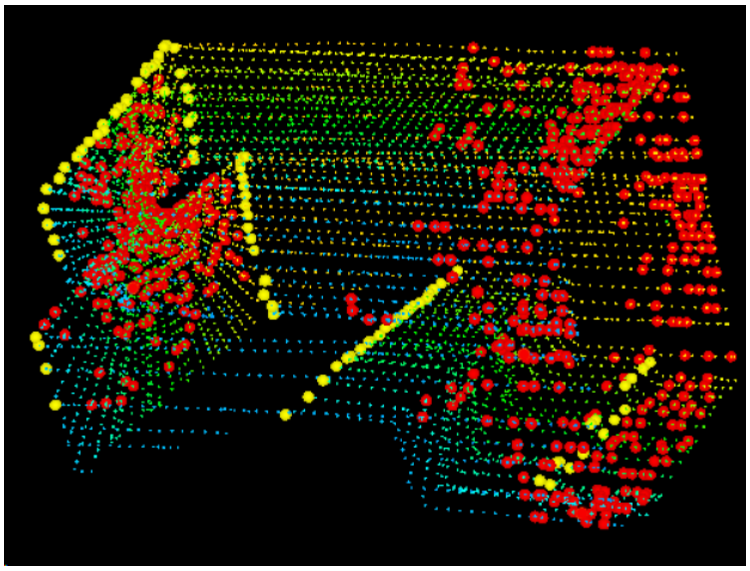


Figura 2.7: Esempio di una mappa generata. in giallo sono rappresentati i punti spigolosi e in rosso i punti piani.[9]

Le *features* vengono estratte nello stesso modo come descritto in Sezione 2.3.1 ma vengono selezionate 10 volte le *features* rispetto a quel caso.
Per trovare le corrispondenza tra ogni *features*, viene immagazzinata la nuvola di punti nella mappa Q_k in aree cubiche di 10m. I punti nei cubi che si intersecano con $\bar{Q}_{k+1}$ vengono estratti ed "catalogati" in 3D KD-tree. Vengono trovati i punti in Q_k in una certa regione attorno ad una *feature* definendo questo insieme di punti S'.

Per ogni punto angoloso vengono selezionati solo punti lungo il bordo all'interno di S', mentre per un punto piano quelli appartenenti allo stesso piano. Si calcola quindi la matrice di covarianza di S' chiamata M, e i suoi autovalori ed autovettori ripetitivamente con V ed E. Se S' è distribuita in un bordo V conterrà un autovalore significativamente grande rispetto agli altri due, ed

l'autovettore E associato ad esso rappresenta l'orientamento del bordo individuato. Se invece i punti in una regione s' che descrivono un piano avranno due autovalori più grandi ed il terzo significativamente piccolo l'autovettore associato a quest'ultimo in E rappresenta l'orientamento del piano. La posizione dei bordi e dei piani viene determinata passando per il centro geometrico della regione S'.

Per calcolare la distanza tra *features* e la sua corrispondenza vengono selezionati due punti per identificare un bordo e tre punti per un piano. In questo modo è possibile utilizzare le Formule 2.9 e 2.10. Dopodiché viene derivata l'equazione per ogni *feature* come Equazione 2.15 e 2.16, ma diversamente in ogni punto in $\bar{Q}_{k+1}$ condivide lo stesso istante t_{k+2}.

L'ottimizzazione non lineare è risolta utilizzando il metodo *Levenberg-Marquart* e $\bar{Q}_{k+1}$ viene registrato nella mappa. Per distribuire uniformemente i punti, la *pointcloud* della mappa viene ridimensionata da un filtro a griglia di voxel con la dimensione voxel di cubi da 5cm.

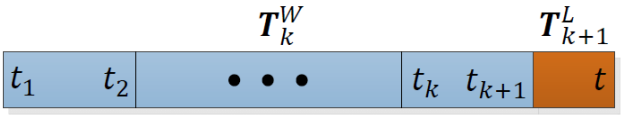


Figura 2.8: Step di integrazione dell'algoritmo di mappatura per determinare la posa del LiDAR [9].

L'integrazione delle Pose viene illustrata in Figura 2.8, la regione blu rappresenta le pose in output dalla mappatura del LiDAR, generata una volta per scansione. La regione arancione invece rappresenta la trasformata in output dell'odometria, T_{k+1}^T ad una frequenza di 10 Hz.
La posa del LiDAR rispetto alla mappa è una combinazione di due trasformazioni alla stessa frequenza dell'odometria del LiDAR.

Capitolo 3

Mappatura 3D

3.1 Voxblox

La libreria *open source* Voxblox è in grado di generare un modello tridimensionale dell'ambiente utilizzando gli *Euclidean Signed Distance Fields* (ESDFs) o i *Truncated Signed Distance Fields* (TSDFs). L'ESDF è una griglia di voxel in cui in ogni punto sono contenute le informazioni della distanza Euclidea dalle superfici che rappresentano l'ostacolo più vicino. I TSDFs invece sono griglie in cui ad ogni elemento è associato un valore pari alla distanza di un oggetto rilevato, misurata lungo la direzione del raggio laser inviato, ma ponendo un limite alla distanza massima chiamata di Troncamento. Un esempio di come è costruito un TSDF lo si può vedere nella Figura 3.1. Considerando di avere una superficie come un oggetto solido davanti al laser (in verde) tutti gli elementi che si troveranno davanti la superficie avranno un valore di distanza positivo mentre quelli dietro tale superficie avranno un valore negativo, viene cosi identificata la superficie in base a dove gli elementi cambiano di segno, nel caso in cui un elemento presenta una distanza maggiore rispetto a quella di troncamento allora quella distanza non viene considerata.

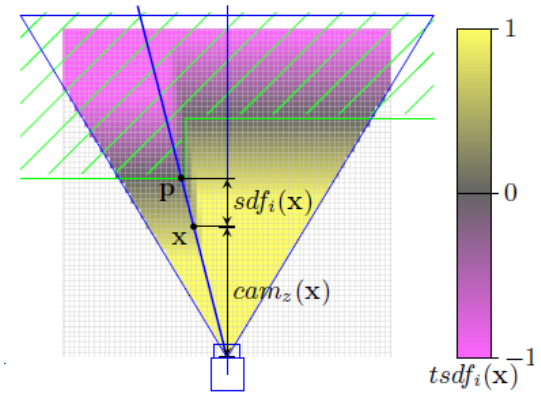


Figura 3.1: Esempio di TSDF in 2D: l'oggetto in verde rappresenta un oggetto solido [15].

La particolarità di Voxblox e il fatto che a partire dall'integrazione per generare la TSDF va a ricostruire e generare la ESDF in questo modo si è in grado di salvare in ogni singolo voxel le informazioni della distanza euclidea, quindi non solo lungo la direzione del laser ma in tutte le direzioni, dell'ostacolo più vicino ottenendo così una mappa in grado di darci maggiori informazioni rispetto ad una semplice mappa occupazionale come quella generata da octomap [16], come si può vedere in Figura 3.2.
Nelle sezioni a seguire verrà esposto come vengo ricostruite la TSDF e la ESDF in Voxblox.

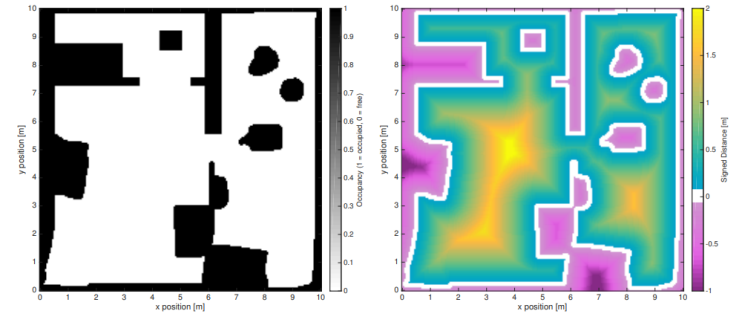


Figura 3.2: Esempio di un confronto tra una mappa occupazionale e una ESDF. La mappa occupazionale dà informazioni solo se uno spazio, è libero o occupato una ESDF è in grado di dirci a che distanza si trova il primo ostacolo, o viceversa se il voxel è occupato ci darà l'informazione della distanza dal primo voxel libero [17].

3.1.1 TSDFs

I TSDFs sono costruiti a partire dalla *pointcloud* e attraverso un processo di raycasting, che attraverso la lettura dei punti determina la distanza tra i punti all'interno di una mappa globale, e le nuove misure di distanza proiettate vengono mediate all'interno dei voxel esistenti, eliminando quelle superiori alla distanza di troncamento δ. La scelta su come costruire un TSDF può avere un forte impatto sia sulla velocità che sull'accuratezza della ricostruzione.
Due fattori importanti sono la "ponderazione" (*weighting*), ovvero il modo in cui le nuove misure vengono mediate con le misurazioni esistenti e di "fusione" (*fusion*) come vengono raggruppati i punti dei sensori; Essi giocano un ruolo importante per aumentare la velocità di ricostruzione e migliorano l'accuratezza, soprattutto con voxel di grandi dimensioni.

La strategia di *weighting* consiste nell'aggiornare tutti i voxel, attraverso una *ray-cast* dall'origine del sensore ad ogni punto della *pointcloud*, con il peso dei voxel lungo questo raggio.
Ogni volta che deve essere aggiornato un voxel quindi il processo sarà quello di misurare la distanza d tra il centro del voxel e un punto (p) dell'ambiente mappato nella direzione di un raggio che attraversa i due punti e l'origine del

sensore s, dando un valore negativo se questo si trova dietro al punto (quindi dietro una superficie) e viceversa positivo se si trova davanti ad essa. A questa distanza vine assegnato un peso w pari ad 1. Dopodiché verranno ricalcolai i valori di distanza e peso che quel voxel aveva all'iterazione precedente effettuandone una media ponderata per le distanze e considerando il valore minimo di peso ad esso associato.
Le equazioni generali che regolano l'unione sono basate sulla distanza e il peso di un voxel esistente (D e W) e dalla sua distanza e peso associati alle misure ottenute dall'osservazione di un punto "speciale" (d e w), ove d è la distanza dal dalla superficie del contorno sono:

$$d(x, p, s) = \|p - x\| sing((p - x) \cdot (p - s)) \tag{3.1}$$

$$w_{const}(x, p) = 1 \tag{3.2}$$

$$D_{i-1}(x, p, s) = \frac{W_i(x)D_i(x) + w(x, p)d(x, p, s)}{W_i(x) + w(x, p)} \tag{3.3}$$

$$W_{i+1}(x, p) = min(W_i(x) + w(x, p), W_{max}) \tag{3.4}$$

ove x è il centro del voxel corrente, p è la posizione di un punto 3D della *pointcloud* e s è l'origine del sensore. D_{i+1} e W_{i+1} sono rispettivamente la distanza e il peso aggiornati.

Per ricavare un peso più accurato vengono considerati anche gli effetti della diminuzione del peso dietro il confine dell'isosuperficie, essi variano con il quadrato della profondità di misura (z^2).
Il peso viene assegnato dalla seguente equazione:

$$W_{quad}(x, p) = \begin{cases} \dfrac{1}{z^2} & -\mathcal{E} < d \\ \dfrac{1}{z^2}\dfrac{1}{\delta - \mathcal{E}}(d + \delta) & -\delta < d < \mathcal{E} \\ 0 & d < \mathcal{E} \end{cases} \tag{3.5}$$

ove la Distanza di troncamento è $\delta = 4\nu$ e $\mathcal{E} = \nu$ e ν è la dimensione del voxel.

Le due strategie di "fusione" usate sono il *"raycasting"* e la *"mappa di proiezione"*. Il metodo *"raycasting"* proietta un raggio dal centro ottico della telecamera al centro di ogni punto osservato e aggiorna tutti i voxel che sono attraversati dal raggio, fino ad una distanza massima uguale a quella di troncamento δ. La mappatura per proiezione proietta invece i voxel nel campo di vista nell'immagine e calcola la sua distanza tra il centro del voxel e il valore di profondità nell'immagine. È significativamente più veloce, ma comporta forti errori di *aliasing* (distorsione da campionamento) per i voxel più grandi.

Il metodo di "fusione"(*mergin approach*) usato in VoxBlox è il *grupped raycasting*: ogni punto viene riproiettato nella griglia di voxel e raggruppato

con tutti gli altri punti che mappano lo stesso voxel, considerando la media del colore e la distanza tra tutti i gruppi di punti. Questo metodo aumenta la velocità di ricostruzione, senza una perdita di accuratezza rilevante.

3.1.2 ESDFs

La ESDF viene costruita a partire dalla TSDF utilizzando il valore della distanza memorizzata in ogni voxel. Tale valore può essere o no modificato in base alla *fixed band* intorno alla superficie: tutti i voxel TSDF la cui distanza soddisfa $|\nu_k| \cdot d < \gamma$, ove ν_k è il voxel TSDF, d è il valore di distanza memorizzato in ogni voxel e γ è il raggio della banda (*fixed band*) .

L'intero metodo si basa sul concetto di *fronte d'onda* (*WaveFront*): si tratta di onde che si propagano da un voxel di partenza ai suoi vicini e aggiornano le loro distanze, aggiungono voxel aggiornati alla coda del fronte d'onda e propagando ulteriormente l'*"onda di aggiornamento"* . Esistono due tipi di fronte d'onda: innalzamento (Lower) e abbassamento (Raise). Un voxel viene aggiunto alla coda di innalzamento (Raise Wevefront) quando la sua nuova distanza dalla TSDF è superiore al valore precedente memorizzato nel voxel ESDF. Questo significa che i voxel e tutti i suoi figli, devono essere invalidati. Il fronte d'onda si propaga fintanto che non ci sono più voxel con genitori che sono stati invalidati.

Il fronte d'onda di abbassamento (Lower Wavefront) inizia quando un nuovo voxel fisso entra nella mappa o un voxel precedentemente osservato abbassa il suo valore di distanza. Le distanze dei voxel vicini vengono aggiornate in base ai voxel vicini e alle loro distanze dal voxel corrente. Il metodo non aggiorna i voxel sconosciuti, memorizza se un voxel è già stato osservato o meno e si occupa della priorità di aggiornamento con una coda appropriata.

Fonti di errore nell'ESDF

Nella ricostruzione 3D si presentano due tipi di errore che possono essere cruciali per la pianificazione delle traiettorie. Il primo dipende dalla distanza proiettiva TSDF, cioè la distanza lungo il raggio della telecamera rispetto alla superficie, che di solito è uguale o superiore alla distanza reale, introducendo un errore che può essere modellato con l'equazione:

$$r_p(\theta) = d\sin(\theta) - d \tag{3.6}$$

ove $r_p(\theta)$ è l'errore di proiezione residua,d è la misura della distanza dal voxel e θ è l'angolo incidente tra la superficie e il raggio della camera. Attraverso l'utilizzo di un LiDAR la minima inclinazione che si può avere è quella calcolata in base all'altezza del sensore e alla massima distanza rilevabile. Le distanze massime individuate dai sensori LiDAR sono molto elevate in oltre considerando il fatto che utilizzando il LiDAR utilizzato in questo lavoro di tesi è posizionato sotto la scocca del rover (vedi Sezione 4.1), l'inclinazione massima

che si otterrà è molto vicina ad un inclinazione nulla. Per questo motivo viene considerata l'inclinazione minima di 0^o considerando che tale misurazione non sia fisicamente possibile, in quanto significherebbe effettuare la misurazione ad una quota infinitesimale, ci si pone quindi nel peggior caso possibile è quindi nel massimo errore effettuabile. Assumendo quindi che l'angolo di incidenza varia uniformemente tra $\pi/20$ e $\pi/2$, l'errore per un singolo voxel osservato può essere modellato come segue:

$$E[r_p(\theta)] = \int_0^{\pi/2} \frac{20}{9\pi}(d\sin(\theta) - d)d\theta = -0.4038d \tag{3.7}$$

Si può notare che d ha il limite superiore uguale alla distanza di troncamento. questo errore può essere ridotto attraverso molteplici osservazioni della stessa superficie ed aumentando la *fixed band*.

La seconda fonte di errore può essere l'assunzione della distanza quasi Euclidea nel calcolo della ESDF. Tale distanza viene misurata lungo una linea orizzontale, verticale o diagonale nella griglia, non caratterizza alcun errore nel caso in cui l'angolo ϕ, compreso tra la normale della superficie e il raggio che va dalla superficie al voxel, è un multiplo di 45^o e ha l'errore massimo nel caso si verifichi: $\phi = 22.5^o$ [18].

Se ϕ è uniformemente distribuito tra 0 e $\pi/4$ l'errore residuo $r_q(\phi)$ e il suo massimo il suo valore previsto è:

$$r_q(\phi) = \left(d - \frac{d\sin(5\pi/8 - \phi)}{\sin(3\pi/8)}\right) \tag{3.8}$$

$$r_q(\frac{\pi}{8}) = -0.0824d \tag{3.9}$$

$$\mathbb{E}[r_q(\phi)] = \int_0^{\frac{\pi}{4}} \frac{4}{\pi}\left(d - \frac{d\sin(5\pi/8 - \phi)}{\sin(3\pi/8)}\right) d\phi = -0.0548d \tag{3.10}$$

Capitolo 4

Apparato Sperimentale

Il sistema utilizzato per la sperimentazione e la navigazione sul campo consiste nel Rover MORPHEUS, il sensore LiDAR *Livox Horizon*, usato per l'acquisizione della nuvola di punti (*pointcloud*) per effettuare la SLAM e la ricostruire la mappa 3D. Inoltre è stato utilizzato un computer collegato tramite Wi-Fi alla Jetson TX1, a supporto della stessa, in particolare per poter utilizzare versioni dei pacchetti di ROS più recenti, e per la movimentazione del rover tramite un controller. Tali sistemi verranno descritti con maggiori dettagli nelle prossime Sezioni.

Figura 4.1: Rover MORPHEUS allo stato attuale

4.1 Rover MORPHEUS

Il rover MORPHEUS (*Mars Operative Rover of Padova Engineering University Students*) è stato sviluppato all'Università di Padova grazie al lavoro di un gruppo di studenti [19]. Il rover è nato con lo scopo di testare tecnologie per la navigazione autonoma, l'estrazione e campionamento di suolo e rocce, in ambienti non strutturati. In particolare per l'esplorazione Marziana e Lunare. La progettazione del rover è stata ideata con il fine di partecipare alla *European Rover Challenge* (ERC), una competizione internazionale di robotica in cui rover presentati dai vari team gareggiano affrontando delle prove basate su missioni reali dell'ESA e della NASA. La gara si svolge sulla replica di un terreno Marziano.
Negli anni sono state effettuate variazioni del design del rover al momento il rover è composto da 6 motori, 6 Arduino per il controllo dei motori, una scheda Jetson TX1 per la gestione di ROS, una stereocamera, un sensore LiDAR e un ricevitore GPS [20]. Come mostrato in Figura 4.2 il LiDAR è posizionato sotto la struttura del rover spostato a sinistra considerando la direzione di marcia, mentre la stereocamera è posizionata al centro sollevata con l'utilizzo di una staffa rispetto alla struttura.

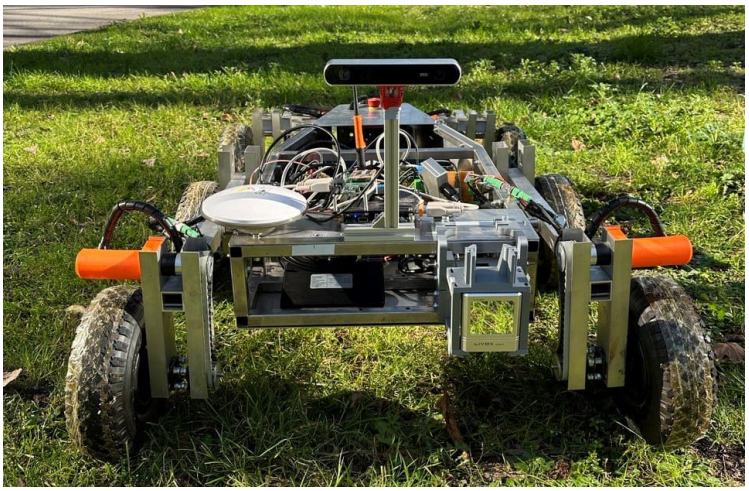


Figura 4.2: Immagine frontale del rover MORPHEUS.

I sei motori movimentano 6 ruote di 26 centimetri, attraverso un sistema di cinghie e pulegge. Le ruote hanno assi di rotazione paralleli tra loro, sono disposte a coppie su tre bilancieri, come illustrato in Figura 4.3.

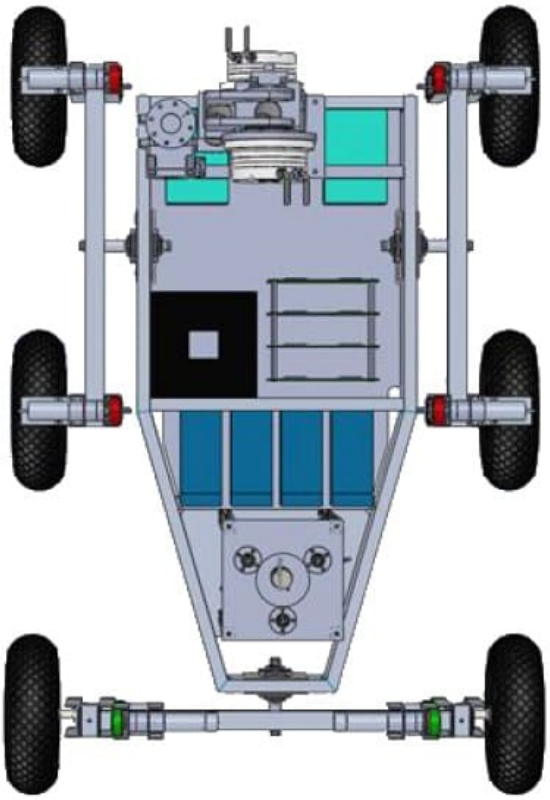


Figura 4.3: Rappresentazione CAD del rover MORPHEUS [20]

La scelta di tale configurazione deriva dalla semplicità di realizzare e controllo. Per curvare, il rover si affida ad una guida *skid-steering*, invece che utilizzare ruote sterzanti, analogamente al rover *Perseverance*. La guida *skid-steering* si basa sul controllo differenziale delle ruote dei due lati e permette di ottenere meccanismi robusti e semplici, mantenendo comunque un'elevata manovrabilità su qualsiasi tipo di terreno. Nonostante la semplicità del meccanismo le complesse interazioni tra ruote e terreno aumentano la difficoltà nel ricavare facilmente un buon modello dinamico e cinematico che descriva con accuratezza il moto del rover. Un esempio di fenomeni difficili da prevedere sono quelli dovuti allo slittamento che sono necessari durante le manovre di rotazione. É comunque possibile fare una prima analisi cinematica del moto del rover mettendo in relazione le velocità angolari delle ruote (ω_R e ω_L) con le velocità lineari (v) e di rotazione (ω) nel seguente modo:

$$\omega_R = \frac{1}{r}(v + b \cdot \omega)$$
$$\omega_L = \frac{1}{r}(v + b \cdot \omega)$$

ove r è il raggio della ruota ($13cm$) e b è la distanza della ruota dall'asse di simmetria ($35cm$). Per un analisi più dettagliata della cinematica si riamanda a [21].

Figura 4.4: Rappresentazione del motore [22].

Per quanto riguarda la parte meccanica del sistema di locomozione, essa si compone di sei ruote che vengono messe in movimento tramite una cinghia da sei motori *brushless* della Maxon, modello EC-max $30\phi 30mm$ da 60W, come mostrato in Figura 4.4. Ad essi vengono collegati dei riduttori planetari, sempre della Maxon, modello GP 32 HP $\phi 32mm$, nella variante *High Power*. Nello specifico il riduttore è composto da tre stadi e permette di ottenere un rapporto di riduzione pari a 86:1. In Tabella 4.1 si riportano le specifiche del motore e in Figura 4.5 il range operativo in funzione della durata dell'operazione eseguita.

Motore EC-max 30 $\Phi30 mm$		
Voltaggio nominale	[V]	24
Corrente a vuoto	[mA]	191
Velocit'a nominale	[rpm]	8040
Coppia nominale (max. coppia continua)	[mNm]	60.7
Corrente nominale (max. corrente continua)	[A]	2.66
Coppia di stallo	[mNm]	458
Corrente di stallo	[A]	18.8
Efficienza massima	[%]	81

Tabella 4.1: Parametri del motore [22].

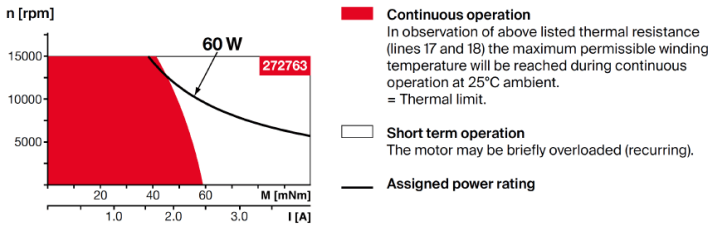


Figura 4.5: Range operativo del motore [22].

4.2 Jetson TX1

Il computer di bordo del rover è composto dalla scheda Jetson TX1, prodotta dalla NVIDIA. É caratterizzata da un computer dove CPU e GPU sono integrati all'interno di un unico processore, chiamato *embedded computer.*

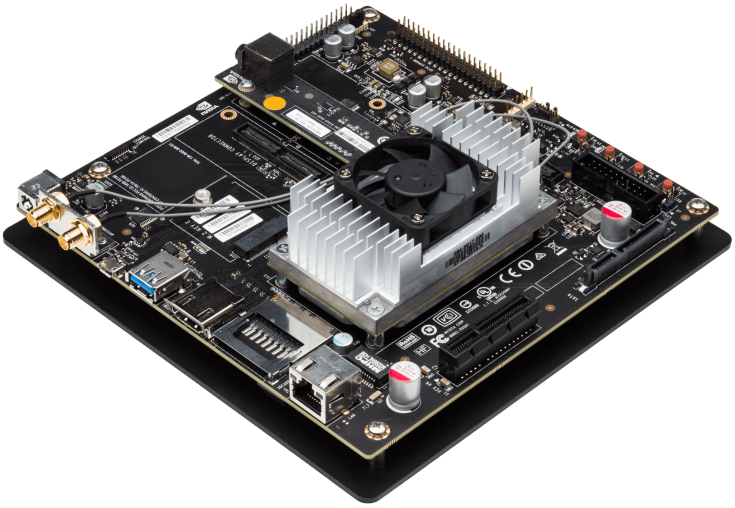


Figura 4.6: Rappresentazione della Jetson TX1 [23]

In Figura 4.7 sono elencate le principali caratteristiche della scheda, le funzioni dei bottoni presenti nella scheda le porte in input/output e alcune caratteristiche principali.

JETSON TX1 MODULE

- NVIDIA Maxwell™ GPU with 256 NVIDIA® CUDA® Cores
- Quad-core ARM® Cortex®-A57 MPCore Processor
- 4 GB LPDDR4 Memory
- 16 GB eMMC 5.1 Flash Storage
- 10/100/1000BASE-T Ethernet

JETSON CAMERA MODULE

- 5 MP Fixed Focus MIPI CSI Camera

BUTTONS

- Power On/Off
- Reset
- Force Recovery
- User-Defined

I/O

- USB 3.0 Type A
- USB 2.0 Micro AB (supports recovery and host mode)
- HDMI
- M.2 Key E
- PCI-E x4
- Gigabit Ethernet
- Full-Size SD
- SATA Data and Power

POWER OPTIONS

- External 19V AC adapter

Figura 4.7: Caratteristiche della scheda Jetson TX12 [23]

4.3 Sensore LiDAR

Il LiDAR utilizzato è *Livox Horizon*, prodotto dalla *Livox Technology Company*. É un LiDAR ad alte performance, costruito con lo scopo di utilizzarlo per i Livelli 3 e 4 di veicoli autonomi. Possiede la capacita di rilevare oggetti a lunga distanza, fino a 260 metri e con un alta densità di punti all'interno della *pointcloud* prodotta. In oltre le sue dimensioni ed il suo costo ridotto più basso rispetto ad altri LiDAR in commercio, lo rendono un ottimo candidato per l'installazione anche su piccoli veicoli.

Figura 4.8: Sensore LiDAR Livox Horizon [24].

4.3.1 Proprietà della *pointcloud* e del FOV

Horizon utilizza una tecnologia di scansione non-ripetitiva, multi-laser e multi-APD DL-Pack. In pratica sfrutta una serie di rilevatori (i fotodiodi ADP, detti *a valanga*),in grado di avere una maggior sensibilità ed accuratezza nella misurazione; ed emettitori laser, che inviano fasci laser scansionano in maniera non ripetitiva l'ambiente, ovvero cambiano la loro direzioni di puntamento in modo tale da non ripeter e sempre la stessa traiettoria. Questo garantisce che all'interno del campo di vista (FOV, *Field of View*),con l'aumentare del tempo, cresca significativamente la copertura del campo di vista (come mostrato in Figura 4.10), rivelando informazioni più dettagliate sull'ambiente circostante. Il campo di vista del LiDAR Horizon è di $81,7^o$ orizzontalmente e $25,1^o$ verticalmente come rappresentato in Figura 4.13)

In Figura 4.9 viene mostrata invece l'allocazione della nuvola di punti di *Horizon* all'interno del FOV in un periodo di $0,1s$. Al centro del FOV, la densità di scansione è più densa e la spaziatura media delle linee è di $0,2^o$ (la maggior parte delle linee è compresa tra $0,1^o$ e $0,3^o$), molto più densa rispetto ai tradizionali sensori LiDAR a 64 linee, in cui la spaziatura delle linee è compresa tra $0,3^o$ e $0,6^o$. Le due aree circolari su entrambi i lati hanno una densità di scansione inferiore, con una spaziatura media delle linee di $0,4^o$ (la maggior parte delle linee è compresa tra $0,2^o$ e $0,8^o$), in grado di competere

con i tradizionali sensori LiDAR a 64 linee entro $0,1s$.

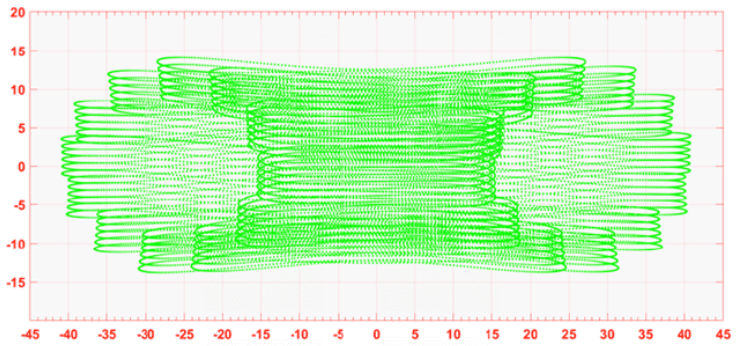


Figura 4.9: *Pointcloud* del Livox Horizon in 0.1s (coordinate in gradi ^{o}) [24].

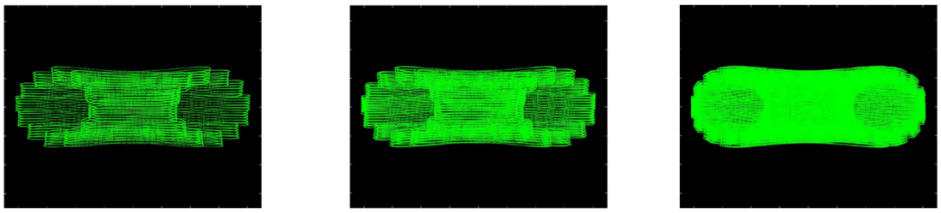


Figura 4.10: Variazione del Pattern della *pointcloud* con l'aumentare del tempo (0.1 0.2 e 0.5 secondi) [24].

In Figura 4.11 mostra la copertura del FOV dell'Horizon confrontato con altri LiDAR in commercio, che utilizzano anch'essi una tecnologia di scansione meccanica. Nel diagramma si può notare come la copertura raggiunga il 60% del FOV in un tempo di integrazione minore di 0,1 secondi come il sensore a *64-line LiDAR*. Mentre gli altri laser confrontati dopo o,1 secondi mantengono la stessa percentuale copertura, l'Horizon ottiene una copertura sempre maggiore fino ad arrivare al 100% del FOV in 0,5 secondi.
In Tabella 4.12 vengono riportate altre specifiche della *pointcloud* acquisita dal LiDAR.

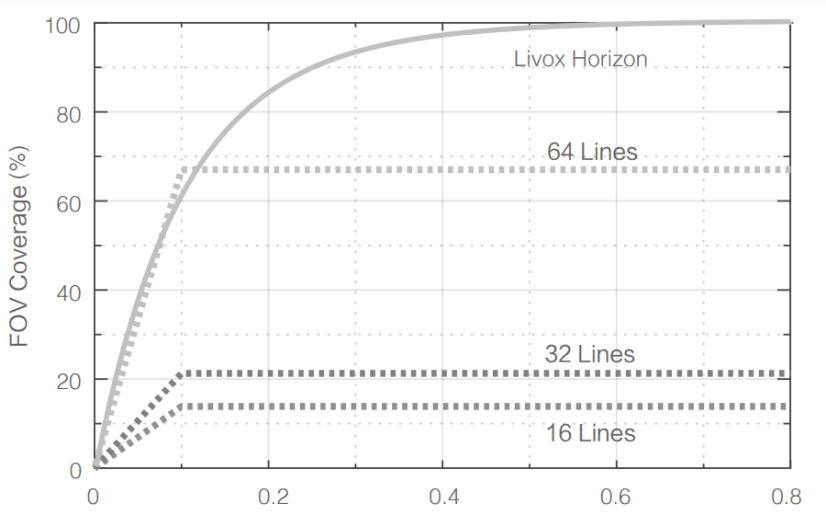


Figura 4.11: Copertura campo di vista del LiDAR [24].

Laser Wavelength	905 nm
Laser Safety	Class 1 (IEC 60825-1:2014) (Safe for eyes)
Detection Range (@100 klx)	90 m @ 10% reflectivity, 130 m @ 20% reflectivity 260 m @ 80% reflectivity
FOV	81.7° (Horizontal) × 25.1° (Vertical)
Distance Random Error	(1σ @ 20 m) < 2 cm
Angular Random Error	1σ < 0.05 °
Beam Divergence	0.28° (Horizontal) × 0.03° (Vertical)
Point Rate	240,000 points/s (first or strongest return) 480,000 points/s (dual return)
False Alarm Ratio (@100 klx)	< 0.01%

Figura 4.12: Altri dati Lidar Horizon [24].

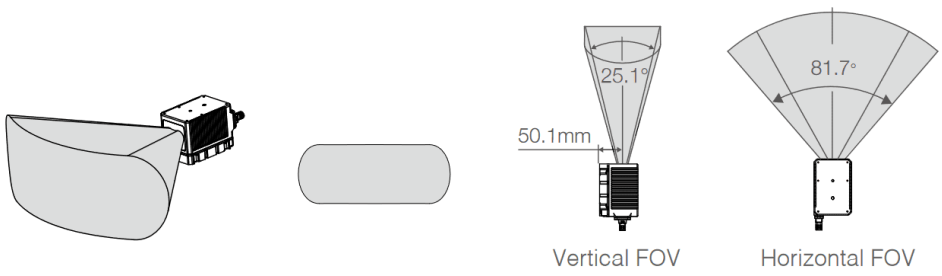


Figura 4.13: Campo di vista (FOV) del LiDAR Livox Horizon [24].

4.3.2 Piattaforma inerziale integrata (IMU) e sistemi di riferimento

Horizon inoltre ha incorporata un IMU (*Inertial Measurement Unit* o piattaforma inerziale) utilizzata per poter avere una stima iniziale dei movimenti del sensore, infatti all'interno di una IMU sono presenti sensori inerziali quali giroscopi e accelerometri in grado di misurare le accelerazione e le rotazioni effettuate. L'utilizzo di questo sistema migliora e corregge eventuali errori nella risoluzione tipica della SLAM. Il modello della piattaforma inerziale incorporata (IMU) è BMI088. Il modulo IMU può essere abilitato o disabilitato a piacere. La frequenza di spinta massima del modulo IMU è di 200Hz, ed è possibile sceglierla tramite il driver fornito da Livox.
In Figura 4.14 vengono mostrati i sistemi di riferimento della *pointcloud* in uscita dal LiDAR ($O - XYZ$) e quella del'IMU ($O' - X'Y'Z'$). In particolare l'origine O' del sistema IMU è definito a partire dalle coordinate della *pointcloud* $(-55.12, -22.26, 29.70)$ ove le distanze sono espresse in millimetri (mm).

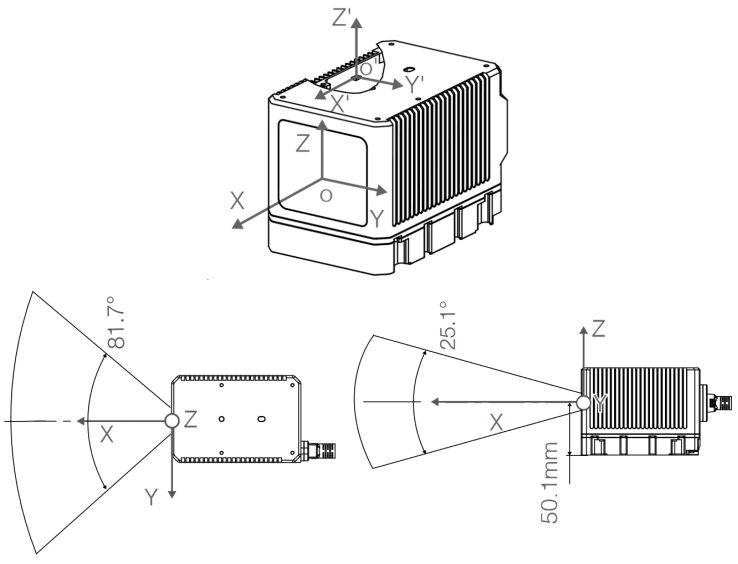


Figura 4.14: Terne del sistema IMU e della *pointcloud* [24].

4.3.3 Dati in output dal LiDAR Horizon

I dati in uscita sono costituiti dalla *pointcloud*, all'interno sono salvati tutti i punti sulle superfici o oggetti rilevati nel campo di vista del LiDAR. Ogni punto contiene le seguenti infrazioni:

- **Riflettività dell'obbiettivo**
  Espresso da 0 a 255, dove i valori tra 0 e 150 corrispondono ad oggetti con una riflessione diffusiva tra 0 e 100% mentre valori tra 151 e 255 corrispondono ad oggetti con una riflessione totale.

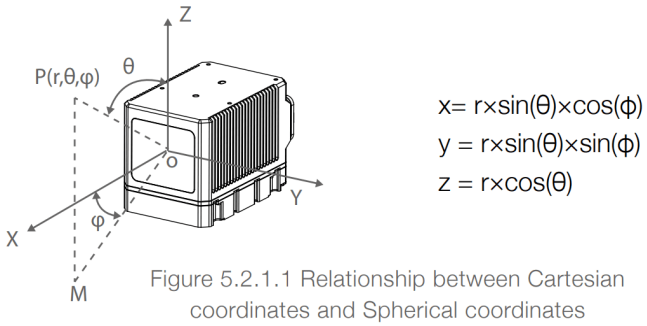


Figura 4.15: Relazioni tra coordinate Cartesiane e Sferiche. [24].

- **Coordinate del punto**
  Le informazioni delle coordinate del Livox Horizon possono essere espresse come coordinate cartesiane (x, y, z) oppure coordinate sferiche (r, θ, ϕ). In Figura 4.15 sono rappresentate la relazione tra le coordinate cartesiane e quelle sferiche.
  Si fa presente che se non ci sono oggetti presenti davanti al LiDAR oppure l'oggetto è posto ad una distanza maggiore di quella limite, in output le coordinate del punto saranno (0,0,0) per le coordinate cartesiane e (o, θ, ϕ) per quelle sferiche.
- **Tags** Indica il tipo di dato acquisito e se il punto rilevato è considerato rumore il formato del *tag* è quello in Figura 4.16

Per altri parametri ed informazioni é possibile consultare il manuale utente [24].

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Reserved		Return number: 00:return 0 01:return 1 10:return 2 11:return 3		Point property based on intensity: 00:Normal 01:High confidence level of the noise 10:Moderate confidence level of the noise 11:Reserved		Point property based on spatial position: 00:Normal 01:High confidence level of the noise 10:Moderate confidence level of the noise 11:Low confidence level of the noise	

Figura 4.16: Formato dei dati della *pointcloud* in uscita dal LiDAR. [24].

Capitolo 5

Implementazione della rete Ros

La gestione del Rover viene effettuata attraverso l'utilizzo di un *framework ROS* (Robot Operating System). ROS è un insieme di *software libraries* applicate alla robotica. Questo strumento è ampiamente utilizzato nella gran parte delle applicazioni robotiche, inoltre è completamente *open source* o *multi-platform*. Un ulteriore vantaggio è il supporto continuo di una comunità molta attiva e molto grande.

Il funzionamento di ROS e dei processi dei diversi pacchetti può essere descritto attraverso uno schema a grafo dove sono presenti *topic* e *nodi*, come rappresentato in Figura 5.1. I nodi sono gli elementi che elaborano i dati processandoli e possono pubblicare o leggere le informazioni all'interno dei *topic*, che sono gli elementi dove vengono comunicati i messaggi d'informazione. Ogni nodo lavora singolarmente e non sa se sono presenti altri nodi ne tanto meno cosa stanno facendo. I *topic* sono gli elementi che contengono i dati e collegano i vari *nodi* nella rete, la loro azione è semplicemente quella di ricevere un informazione (*messaggio*) è dare la possibilità a qualunque nodo di poterla leggere.

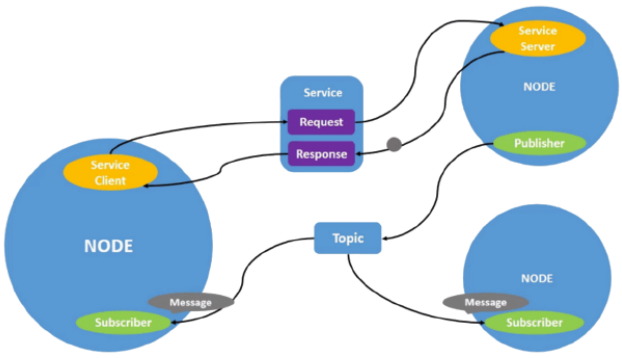


Figura 5.1: Rappresentazione di una rete ROS [25]

Il linguaggio principale di scrittura dei nodi è C++ oppure Phayton, all'interno vengono incluse le librerie scaricate dai diversi pacchetti disponibili.

Nel caso di questo elaborato sono stati utilizzati i seguenti pacchetti:

- **livox_ros_driver**, per l'utilizzo del LiDAR *Livox Horizon*;
- **fast_lio**, per l'implementazione della SLAM;
- **voxblox_ROS**, per l'implementazione e le generazioni delle mappe volumetriche 3D ;
- **locomotion**, per il controllo dei motori del rover;

I primi pacchetti elencati sono stati sviluppati da terzi e sono disponibili e facilmente reperibili dalla Wiki di ROS, mentre gli ultimi due pacchetti *locomotion* e *joystick* sono stati sviluppati appositamente per il rover MORPHEUS. In Figura 5.2 viene mostrata la rappresentazione della rete ROS realizzata: all'interno delle figure ovali sono rappresentati i nodi contenuti nei pacchetti elencati in precedenza mentre in quelle con un riquadro rettangolare sono i topic dove vengono quindi pubblicati i messaggi che i nodi si scambiano tra loro. Per la generazione di questa mappa è stato utilizzato una comando di ROS chiamato *rqt_graph*.

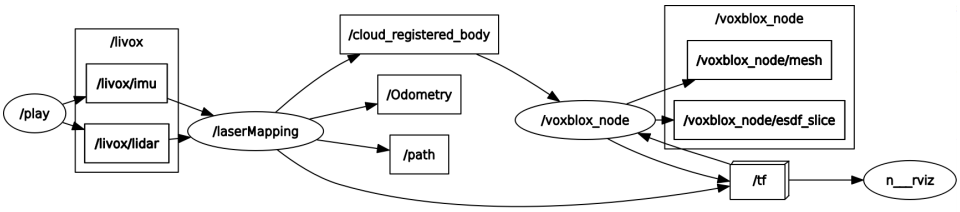


Figura 5.2: Rappresentazione rete ROS realizzata

5.1 Pacchetto livox_ros_driver

Il pacchetto livox_ros_driver è una libreria ROS [26], in grado di connettere i sensori LiDAR e leggerne i dati in uscita, sviluppata della Livox. É assimilabile ad un driver che effettua la comunicazione con il sensore e ne rilascia in output la *pointcloud* scansionata dal LiDAR.
Questo pacchetto è in grado di restituire la *pointcloud* in vari formati:

- *Livox pointcloud2* (Tabella 5.2);
- formato personalizzato da Livox (Tabella 5.2).

Formato	**oggetto**	**descrizione**
float32	x	Asse X [m]
float32	y	Asse Y [m]
float32	z	Asse Z [m]
float32	intensità	Il corrisponde alla riflettività, tra 0.0 e 255.0
uint8	tag	Livox tag
uint8	line	Numero di laser nel LiDAR

Tabella 5.1: Struttura dati salvati nella *poincloud2*

Formato C++	**oggetto**	**descrizione**
Header	header	ROS *standard message header*
uint64	timebase	Istante di tempo del primo punto
uint32	point_num	Numero totale di punti nella *pointclouds*
uint8	LiDAR_id	ID del dispositivo LiDAR
uint8[3]	rsvd	Uso riservato
CustomPoint[ ]	points	Dati *pointcloud*

Tabella 5.2: Struttura dati personalizzata da Livox, CustomPoint[ ] è una struttura descritta in Tabella 5.3.

Formato C++	**oggetto**	**descrizione**
uint32	offset_time	Offset dal "*tempo base*"
float32	x	Asse X [m]
float32	y	Asse Y [m]
float32	z	Asse Z [m]
uint8	reflectivity	Riflettività,tra 0 e 255
uint8	tag	livox tag
uint8	line	numero di laser nel LiDAR

Tabella 5.3: Struttura dei dati salvati in CustomPoint[ ]

5.2 Pacchetto Fast_lio 2

Il pacchetto FAST_LIO 2 [27](*Fast LiDAR-Internal Odometry*) è una libreria ROS open source [28], progettata per avere un'alta efficienza computazionale nella ricostruzione dell'odometria inerziale sfruttando sensori LiDAR. É basato sul pacchetto FAST_LIO [29] con alcune migliorie per renderlo più efficiente e veloce. Per effettuare l'odometria questo pacchetto unisce i dati della nuvola di punti acquisita dal LiDAR e quelli di una piattaforma inerziale (IMU), utilizzando filtro di Kalman iterato per permettere una navigazione robusta anche se presenta movimenti bruschi ed l'ambiente è caratterizzato da un alto rumore.

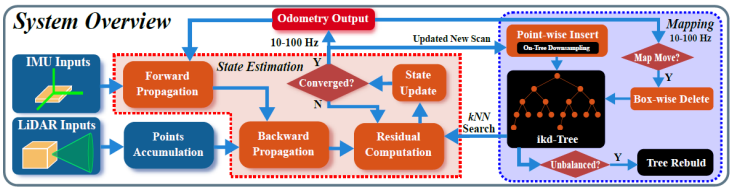


Figura 5.3: Schema di funzionamento di FAST_LIO 2 [27].

In particolare esso possiede le seguenti caratteristiche:

- Ottimizzazione dell'odometria tramite il *Fast Iterated Kalman Filter*;
- Ricerca KD-Tree parallela in modo da diminuire il tempo computazionale di richiesta;
- Mappatura incrementale sfruttando ikd-Tree [30], che permette una velocità piu alte fino a raggiungere frequenze del LiDAR di 100 Hz;
- Odometria diretta (sulla mappa) utilizzando l'intera nuvola di punti in uscita dal lidar (dando la possibilità di disattivare il rilevamento delle *features*), ciò permette una migliore precisione;
- Poiché non viene utilizzata la funzione dell'estrazione delle *features*, FAST_LIO può supportare diversi modelli e tipologie di LiDAR includendo sia quelli rotanti (Velodyne,Ouster) che fissi (Livox Avia, Hoizon, MID-70) e può facilmente essere esteso per l'utilizzo di più LiDARs assieme;
- Supporto a IMU esterne;
- Supporto di processori basati su ARM tra cui Khadas, VIM3, Nivida TX2, Raspberry Pi 4B (8G RAM).

In Figura 5.3 è rappresentato lo schema di funzionamento di FAST_LIO. In input all'algoritmo si ha la *pointcloud* fornita dal LiDAR e le informazioni dalla piattaforma IMU. L'algoritmo tramite dei processi iterativi fornirà la mappa generata e l'odometria ricostruita, un esempio di mappa e percorso ricostruito è rappresenta in Figura 5.4.

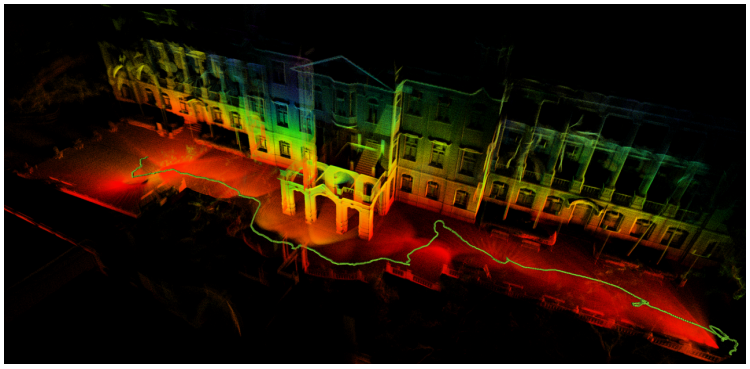


Figura 5.4: Esempio della SLAM effettuata da FAST_LIO [29].

5.3 Pacchetto VoxBlox

Il pacchetto Voxblox è una libreria *open source* di ROS [31]. É stato sviluppato dall' ETH di Zurigo per generare un modello volumetrico tridimensionale dell'ambiente in tempo reale, basandosi sulla *Truncated Signed Distance Field* (TSDF) e costruire in maniera incrementale *Euclidean Signed Distance Field* (ESDF).
É stato sviluppato con lo scopo di funzionare in un MAV (*Micro Aerial Vehicles*), per risolvere il problema di pianificazione di una traiettoria in tempo reale in ambienti sconosciuti. Il problema nasce dal fatto che sia la pianificazione che la mappatura devono essere effettuate in tempo reale e ciò comporta un notevole costo computazionale.

L'algoritmo richiede in input una *pointcloud*, che in questo caso di studio è fornito in uscita dal LiDAR Livox Horizion, e una stima della sua posizione, fornita in questo caso dall'algoritmo di SLAM utilizzato (FAST_LIO).

Voxblox si differenzia dalle altre librerie che utilizzano SDF (*Signed Distance Field*) per le seguenti ragioni:

- è possibile effettuare l'integrazione con l'utilizzo di un *sigle core* della CPU;
- è in grado di generare diverse tipologie di mappa (contenenti diversi tipi di voxel)
- serializzazione utilizzando *"protobufs"*
- possiede diversi modi per gestire la "ponderazione" (*weighting*, vedi Capitolo 3);
- diversi metodi per inserire le informazioni della posa di ogni scansione
- facilmente estensibile ad integratori desiderati
- presenta la capacità di costruire la *Euclidean Signed Distance Field* (ESDF) direttamente dalla TSDF.

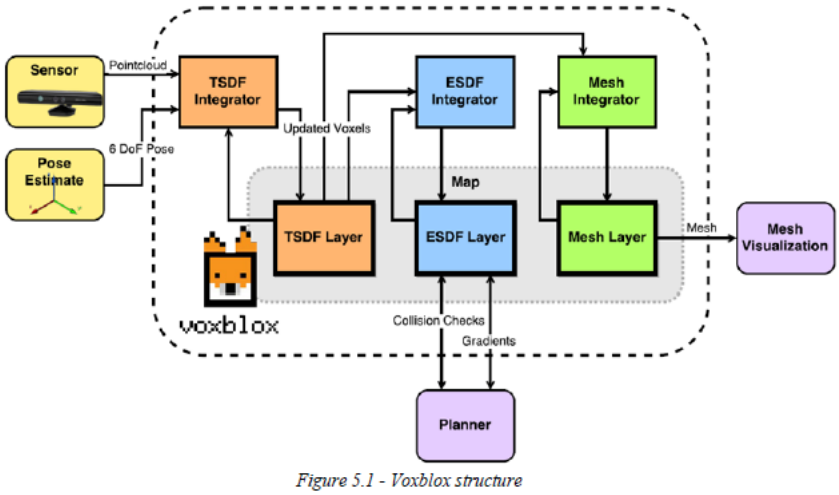


Figura 5.5: Schema di funzionamento di Voxblox [2]

5.4 Pacchetto locomotion

Questo pacchetto è stato sviluppato appositamente per il rover e serve a controllare i motori delle ruote del rover, dati un comando di velocità lineare e uno di velocità angolare. In particolare il pacchetto e composto da due nodi:

- ***loc base controller***: converte i comandi delle due velocità in un vettore di tre numeri. Il primo valore serve a far compiere una delle cinque operazione per il nodo successivo (0→Nessuna Operazione, 1→Setup dei motori, 2→Aziona i motori, 3→Controlla status motori, 4→Chiudi i motori), il secondo e il terzo sono rispettivamente le velocità dei motori di sinistra e di destra.
- ***debug Arduino***: gestisce le cinque operazioni descritte sopra e converte i messaggi di velocità dei motori di sinistra e destra nelle PRF (*Pulse Repetiton Frequency*)da fornire agli Arduino che controllano i motori.

In questo caso i comandi di velocità vengono assegnati in una scala da zero a uno. Questo è pensato per comandare il rover con il joystick per il quale non esiste una vera e propria conversione tra segnale dello stesso e velocità in metri al secondo.

Capitolo 6

Risultati

In questo capitolo verranno esposti i risultati e le analisi effettuate per poter validare gli algoritmi selezionati (FAST_LIO, per la SLAM, e Voxblox per la ricostruzione 3D della mappa) per raggiungere e soddisfare gli obbiettivi imposti in questo elaborato. Successivamente verranno discussi i risultati ottenuti esponendone gli errori riscontrati e le soluzioni adottate per migliorare la ricostruzione.

Per effettuare un'analisi completa e per poter testare e validare l'algoritmo di ricostruzione 3D sono stati registrati più percorsi in ambienti diversi. Si possono suddividere in due categorie: in ambienti chiusi e limitati, che simulano scenari simili a quelli che si possono verificare all'interno di gallerie o grotte, in cui gli spazi sono ristretti, e in ambienti esterni "all'aperto" che simulano invece scenari più simili a quelli dell'esplorazione in ambienti ampi.
Tuttavia si fa presente che per simulare ambienti più simili a quelli presenti su Marte sarebbe necessario portare il rover in ambienti desertici dove sono presenti molti meno punti di riferimento nell'ambiente se non quelli di ostacoli a terra e le malformazione del terreno stesso o al più pareti rocciose all'interno di un cratere ad esempio.
In particolare sono stati effettuati i seguenti test:

- All'interno dei corridoi dove è presente il laboratorio del rover MORPHEUS;
- All'interno di una Piazza nel complesso dove sono presenti gli edifici della facoltà di psicologia dell'Università di Padova e la residenza ESU Nord Piovego.
- Attraversando un ponte pedonale che porta dal lungoargine Piovego al Parco Europa di Padova;

L'acquisizione dei dati è stata effettuata direttamente utilizzando il computer di bordo del rover MORPHEUS. Solitamente gli algoritmi analizzati in questo elaborato devono essere utilizzati "online", ovvero in tempo reale, per poter ricostruire la mappa e darla in input all'algoritmo di pianificazione, in maniera tale da poter effettuare la navigazione autonoma.
Per poter effettuare vari test mantenendo lo stesso percorso effettuato e variando i paramatri utilizzati per la ricostruzione, in modo tale da poterne

confrontare i risultati, è stata effettuata l'acquisizione con il rover ma si è utilizzando successivamente un computer esterno, per la creazione della mappa 3D (mesh e ESDF) e la lavorazione dei dati.

La ricreazione del processo di generazione della mappa in tempo reale è stato effettuato tramite una simulazione sfruttando il comando di ROS: *"**rosbag**"*. Utilizzando l'opzione "*record*" del comando è possibile salvare e registrare tutte le informazioni dei dati pubblicati nei "*topic*" attivi all'interno della rete ROS per ogni istante di tempo. Tale operazione è stata effettuata all'interno del pc del rover durante le acquisizioni effettuate. Successivamente con l'opzione "*play*" è possibile ripubblicare le informazioni dei "*topic*" ad ogni istante, simulando in questo modo la ricezione di tali informazioni in tempo reale.

6.1 Ricostruzione percorso (SLAM)

In questa sezione vengono analizzati i percorsi ricostruiti tramite la SLAM di FAST_LIO ed effettuate delle verifiche sull'affidabilità della ricostruzione. In particolare per i due percorsi esterni la traccia ricreata è stata confrontata con le immagini satellitari e con una traccia GPS ottenuta tramite un smartphone, mentre nel caso del percorso interno la mappa ricostruita verrà confrontata con la mappa generata. Grazie a questi confronti sarà possibile definire la veridicità e la bontà dei percorsi ottenuti con la SLAM. Verranno esposti inoltre i differenti parametri utilizzati per la ricostruzione dei percorsi interno ed esterno.

6.1.1 Percorso interno: Il corridoio

Il primo precorso ad essere analizzato è stato il corridoio adiacente al laboratorio. Dopo il corretto settaggio dei parametri di FAST_LIO è stato ricostruito il percorso in maniera ottimale, come è possibile vedere nella Figura 6.1 nonostante fosse presente una lieve inclinazione del corridoio finestrato sulla destra.

Gestione degli errori

Data la natura stessa dell'ambiente i punti rilevati dal LiDAR hanno tutti una distanza, di profondità, molto ridotta inoltre la presenza di molte superfici riflettenti come le finestre e il pavimento lucido hanno reso il processo di ricostruzione del percorso molto più complesso. Infatti inizialmente si sono riscontrate difficoltà nella ricostruzione soprattutto nelle fasi in cui è stata effettuata una manovra per invertire il senso di marcia del rover (come si vede in Figura 6.2). Questa manovra è caratterizzata da molteplici fonti di errore tra le quali sono state individuate le seguenti:

- un velocità di rotazione più elevata della direzione del LiDAR, ciò comporta che i punti individuati dal rover in un unica scansione si riferiscono ad una posa sempre diversa di conseguenza rende più difficile la ricostruzione della posa;

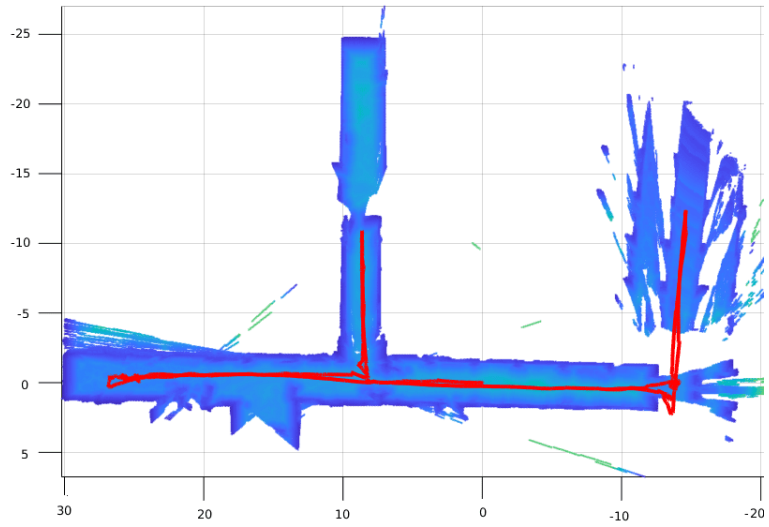


Figura 6.1: Ricostruzione percorso nel corridoio

- la stretta vicinanza del sensore alla parete, questo assieme alla caratteristica stessa del LiDAR utilizzato ovvero con un FOV limitato, porta ad avere in quegli istanti una nuvola di punti concentrata in un punto;
- mancanza di elementi di riferimento, come accennato al punto precedente il FOV limitato non garantisce di avere un numero sufficiente di punti per poter effettuare la ricostruzione dell'odometria del rover.

Per poter migliorare quindi l'odometria sono stati aumentati i parametri di precisione, diminuendo il valore del filtro sulla dimensione dei voxel della mappa, ed è stata disattivata la funzione della ricerca delle *feature*, in quanto la mancanza di un numero sufficiente di *feature*, non permetteva una ricostruzione della posa correttamente. Disabilitando questa funzione quindi la SLAM viene effettuata utilizzando l'intera nuvola di punti del sensore.
In particolari i parametri chiave utilizzati sono quelli riportati in Tabella 6.1.

Parametri FAST_LIO per ambiente ristretto	
feature extrat enable	0 (False)
point filter num	1
filtre size surf	0.5
filtre size map	0.2

Tabella 6.1: Parametri utilizzati per effettuare la SLAM nel corridoio del dipartimento.

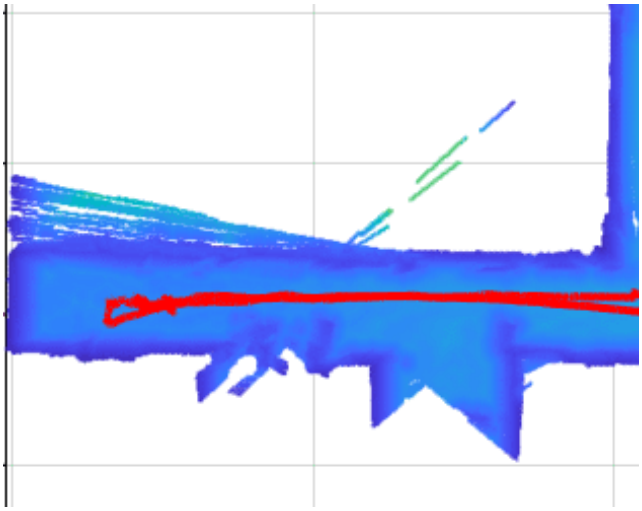


Figura 6.2: Manovra di inversione di marcia effettuata.

6.1.2 Percorsi Esterni

Piazzale nel dipartimento di psicologia

Il secondo percorso registrato ed analizzato è un percorso chiuso all'interno di un piazzale. Il percorso è stato ricostruito in maniera ottimale, come mostrato in Figura 6.3 con l'immagine satellitare, infatti il percorso è stato chiuso correttamente e la sua traccia è comparabile a quella effettuata. In tale immagine viene in oltre mostrata una tracia GPS (percorso in blu) realizzata con l'ausilio di uno smartphone, tuttavia la bassa precisione del dispositivo e la presenza di grandi edifici molto vicini, non rende la traccia un ottimo parametro di confronto, nonostante questo si può vedere che le due traccie hanno un andamento simile.

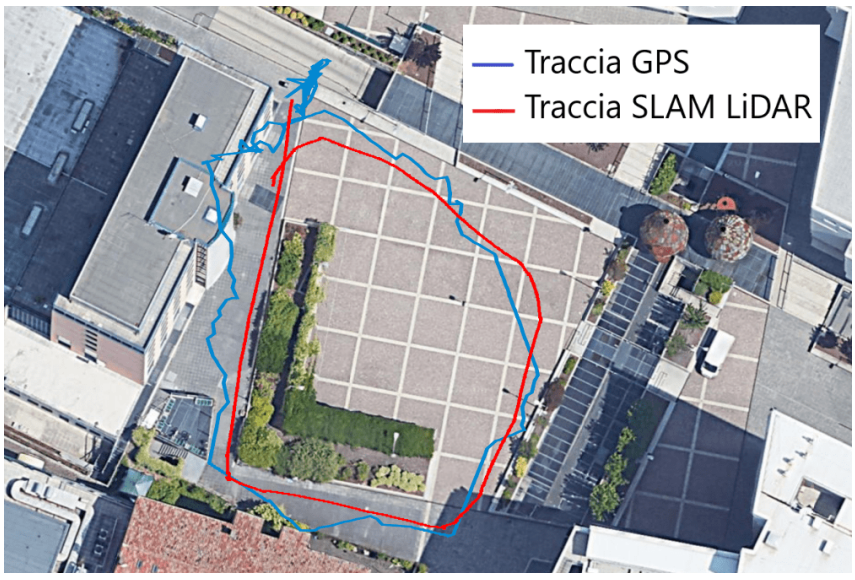


Figura 6.3: Confronto della traccia ricostruita con la SLAM (in rosso) con le immagini satellitari e la traccia ottenuta dal GPS dello smartphone.

Gestione degli errori

In questo caso utilizzando i parametri standard dell'algoritmo di SLAM utilizzato il percorso è stato ricostruito con buoni risultati, tuttavia il lieve dislivello che la piazza presenta ha causato un lieve errore nella quota dalla posizione di partenza rispetto a quella di arrivo. Tramite una leggera diminuzione dei parametri di dimensioni tale dislivello è stato corretto e risultato accettabile. I parametri utilizzati sono riportati nella Tabella 6.2.

Parametri FAST_LIO per ambienti all'aperto	
feature extrat enable	0 (False)
point filter num	3
filtre size surf	0.6
filtre size map	0.4

Tabella 6.2: Parametri utilizzati per effettuare la SLAM nel nella piazza del dipartimento di Psicologia.

Ponte pedonale del Parco Europa

Il terzo percorso ricostruito è quello effettuato attraversando un ponte pedonale che unisce il lungoargine Piovego al parco Europa. Il percorso è stato perfettamente ricostruito grazie all'utilizzo dei parametri utilizzati nella SLAM della piazza ovvero quelli riportati in Tabella 6.2.

Figura 6.4: Confronto della ricostruzione del percorso tramite la SLAM con le immagini satellitari e il GPS.

Nella Figura 6.4 è possibile vedere un confronto con le immagini satellitari e con i dati GPS anch'essi acquisiti tramite l'utilizzo dello smartphone. In questo percorso rispetto al caso precedente la tracci GPS è più accurata,

infatti l'ambiente a sgombra da elementi di disturbo del segnale. Il doppio confronto con le immagini satellitare e con la traccia GPS, mostra che la traccia ricostruita è confrontabile e ciò permette di validare anche per questo caso l'algoritmo SLAM utilizzato.

6.2 Mappatura 3D

In questa fase vengono esposti ed analizzati i risultati ottenuti con l'utilizzo di Voxblox. in particolare verrà esposto:

- La generazione della mappa ESDF caratteristica di Voxblox per la pianificazione;
- Le mesh realizzate dell'ambiente, ponendo attenzione sulle dimensioni dei voxel utilizzate per la ricostruzione;

6.2.1 Mesh 3D di Voxblox

Voxblox è in grado di generare una mesh utilizzando la mappa TSDF ricreando cosi l'ambiente in una mappa 3D in alta definizione visibile in maniera efficace anche ad un occhio umano. Un esempio di questa mappa è mostrato in Figura 6.5, ricostruita durante la navigazione nella piazza del dipartimento di Psicologia.

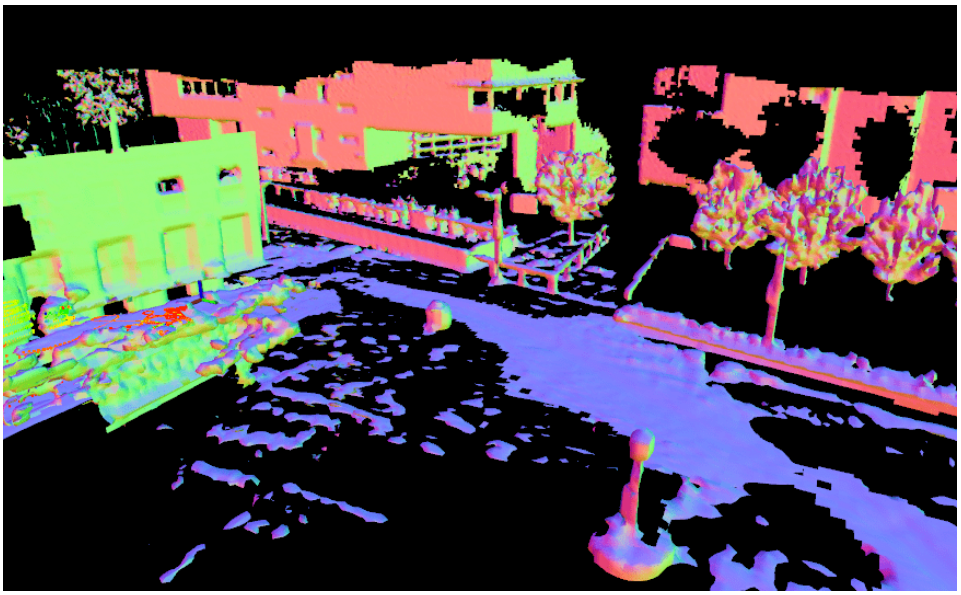


Figura 6.5: Rappresentazione della mesh generata della piazza del dipartimento di Psicologia.

Le caratteristiche che sono state analizzate e valutate riguardano l'impatto della dimensione dei voxel utilizzati e il peso della mappa. In particolare più la dimensione dei voxel è piccola più dettagli vengono rappresentati nella mappa ma ciò ne consegue che la dimensione della mappa cresce considerevolmente.

Una volta simulata la ricostruzione della mappa come se fosse stata fatta in tempo reale, è stata salvata la mesh generata anch'essa in tempo reale, tramite in nodo *server* di ROS fornito da Voxblox chiamato **generate_mesh**. Come si può vedere in Tabella 6.3, sono stati analizzati tre dimensioni sempre riferite al percorso effettuato nella piazza ovvero con una dimensione dei voxel di 0.1, 0.15 e 0.2 $[m]$.

Come si può vedere rispetto alla dimensione di 0,2 $[m]$ il peso delle mappe da 0.15 e da 0.1 $[m]$ sono aumentate rispettivamente dei 172% e del 230% . Si fa presente inoltre che la mappa generata da una dimensione dei voxel di 0.1 $[m]$ a differenza delle altre due ha ottenuto un ricostruzione parziale dell'ambiente, non mappando quindi una buona parte della mappa, questo è stato dovuto da una mancanza di potenza del computer utilizzato. Nonostante questo tale mappa è caratterizzata da un peso maggiore del doppio rispetto a quella con voxel da 0.2 $[m]$.

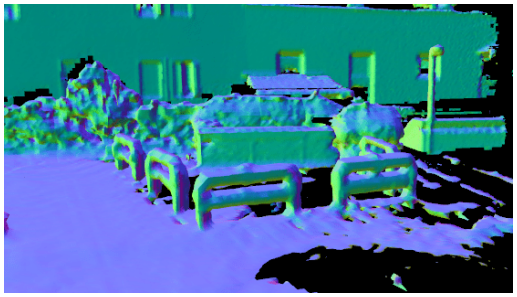
Dimensione voxel	**Peso della mesh**
0.10 $[m]$	84.3 Mb
0.15 $[m]$	62.9 Mb
0.20 $[m]$	36.5 Mb

Tabella 6.3: Confronto della dimensione dei voxel con il peso delle mesh generate.

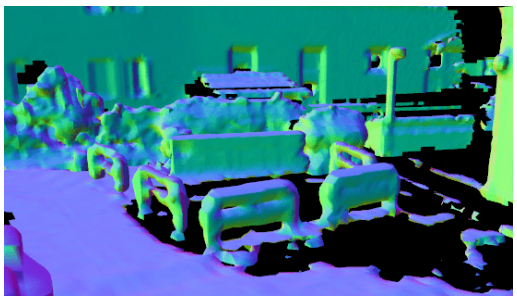
Come già anticipato in precedenza la dimensione dei voxel influisce sulla rappresentazione dei dettagli mappati dell'ambiente. Nelle Figure 6.6, 6.6b, 6.6c e 6.6d, è mostrato come vengono mappati e ricostruiti le transenne presenti nella piazza ed confrontarle con una foto acquisita dalla stereo camera a bordo del rover MORPHEUS (Figura 6.6a).
Da queste immagini si può vedere come l'ostacolo è stato individuato e ricostruito in tutti e tre i casi. Ma solo tramite l'utilizzo di una dimensione dei voxel pari a 0.1 $[m]$ è possibile distinguere chiaramente anche i fori interni, e non come se fosse un parallelepipedo pieno. Questo in base alle esigenze può essere più o meno utile; ad esempio se il veicolo è abbastanza piccolo è in grado di passare sotto a questo ostacolo senza doverlo aggirare, questo porta un miglioramento delle tempistiche nel raggiungere un obbiettivo e consumando meno risorse energetiche. Nel caso di un rover di dimensioni maggiore invece rilevare dettagli cosi piccoli può essere considerato,in alcuni casi, un dispendio di energie computazionali eccessivo.

Considerando che in un ambitene esterno molto ampio effettuare una ricostruzione tramite l'ESDF e il TSDF, esposte alla Sezione 3.1, con un accuratezza elevata richiede un costo computazionale maggiore si è ritenuto opportuno utilizzare una dimensione di voxel più alta ovvero di 0.2 $[m]$. Questa scelta ha permesso di ottenere delle ricostruzioni degli ambienti esterni con ottimi ri-

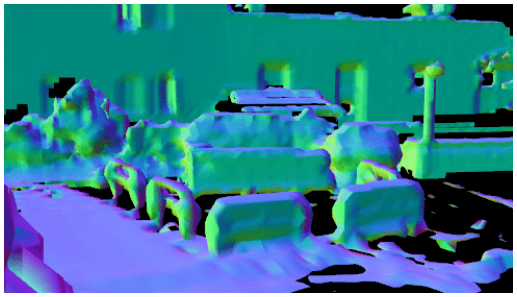
(a) Immagine di riferimento



(b) Dimensione dei voxel di 0.1 $[m]$



(c) Dimensione dei voxel di 0.15 $[m]$



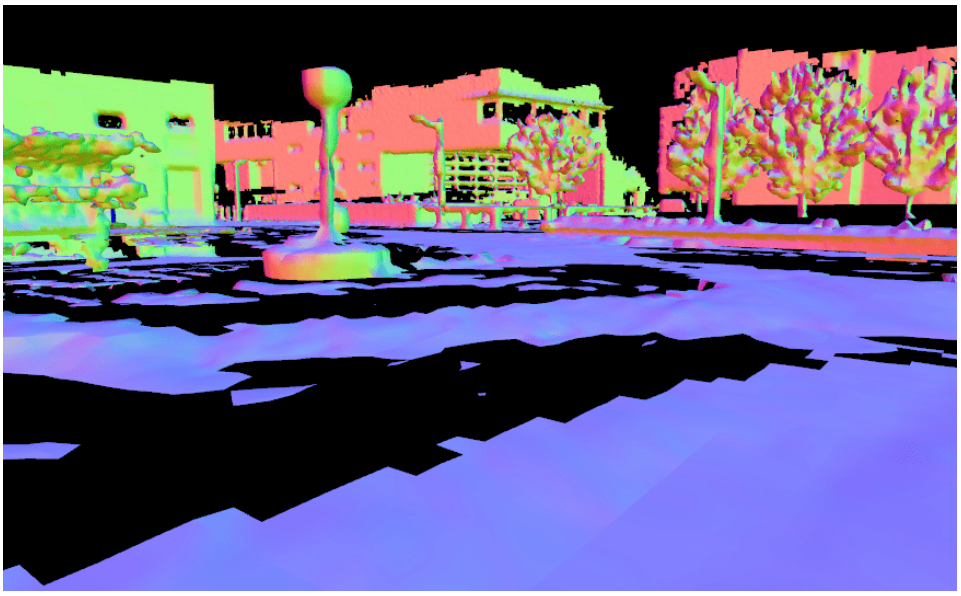
(d) Dimensione dei voxel di 0.2 $[m]$

Figura 6.6: Confronto delle mesh generate al variare della dimensione dei voxel nella TSDF.

sultati. Di seguito verranno esposti vari confronti tramite un confronto con le immagini della stereocamera del rover e delle immagini satellitari 3D ottenute da *Google Earth*, relativi agli ambienti analizzati.

Nelle Figure 6.7 e 6.8 viene mostrata la Piazza dietro al dipartimento di Psicologia attraverso i due punti di vista appena descritti. In particolare si può notare che la dimensione dei voxel scelta è risultata sufficiente a ricostruire fedelmente l'ambiente. Dalla mesh generata è possibile individuare e riconoscere tutti gli elementi come la forma dei palazzi, di alberi e ostacoli presenti come ad esempio il lampione al centro della piazza.

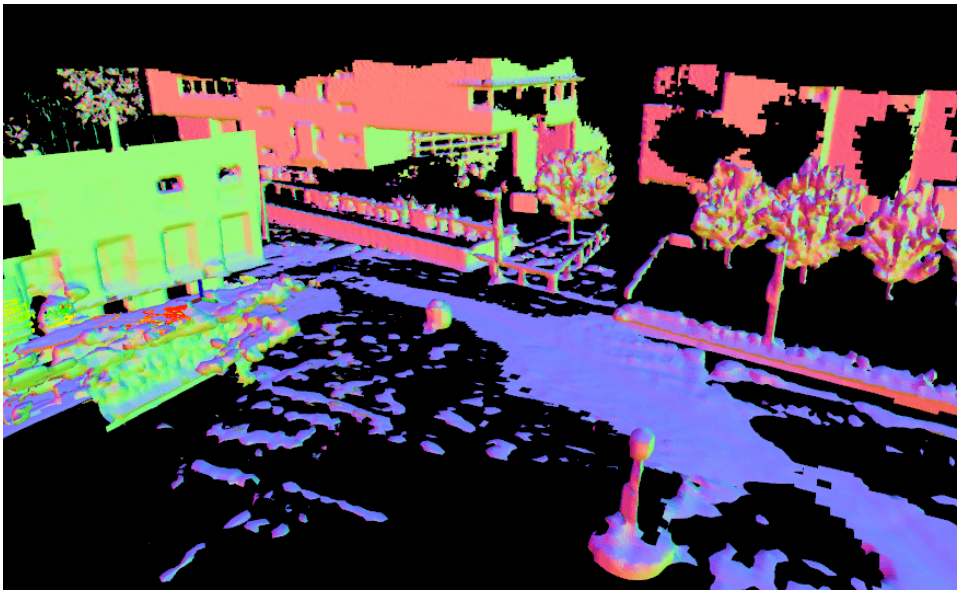
(a) Immagine della piazza.



(b) Mesh creata con voxel di 0.2[m].

Figura 6.7: Confronto della mesh ricreata con un'immagine della piazza.

(a) Immagine satellitare della piazza.

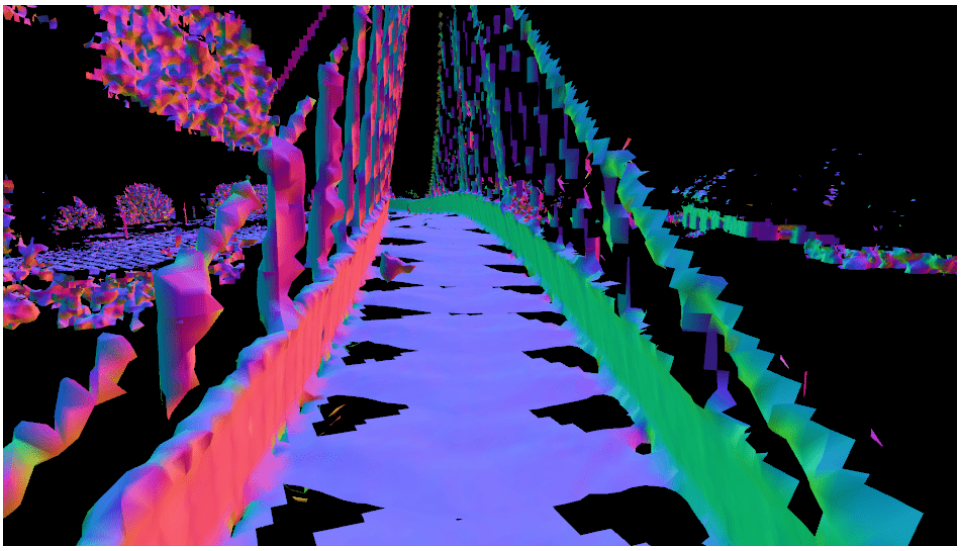


(b) Mesh creata con voxel di 0.2[m].

Figura 6.8: Confronto della Mesh ricreata vista dall'alto con un'immagine satellitare della piazza.

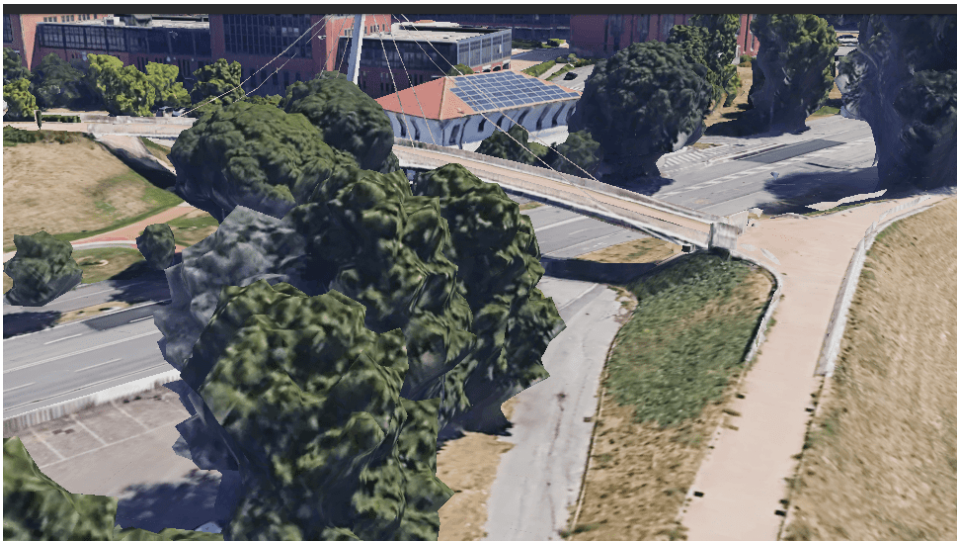
Nelle Figure 6.9 e 6.10 vengono mostrate e messe a confronto le mesh relative al percorso effettuato sul Ponte del Parco della Musica di Padova. In queste immagini si può notare come vengano mappati anche dettagli più piccolo come ad esempio gli stralli del ponte, ovvero le corde che lo sorreggono. Nel caso in esame l'individuazione e ricostruzioni di questi elementi non è essenziale per la navigazione del rover, in quanto la loro posizione non è tale per cui risulterebbe un ostacolo per rover MORPHEUS, in ogni caso lo potrebbe essere importate se il veicolo utilizzato fosse ad esempio un drone. In ogni caso la possibilità di rilevare oggetti cosi piccoli è comunque una funzione molto utile, infatti se questi elementi fossero un ostacolo per la navigazione del rover e non venissero individuati porterebbero ad una collisione e di conseguenza ad un possibile danneggiamento del rover stesso.

(a) Immagine del ponte.

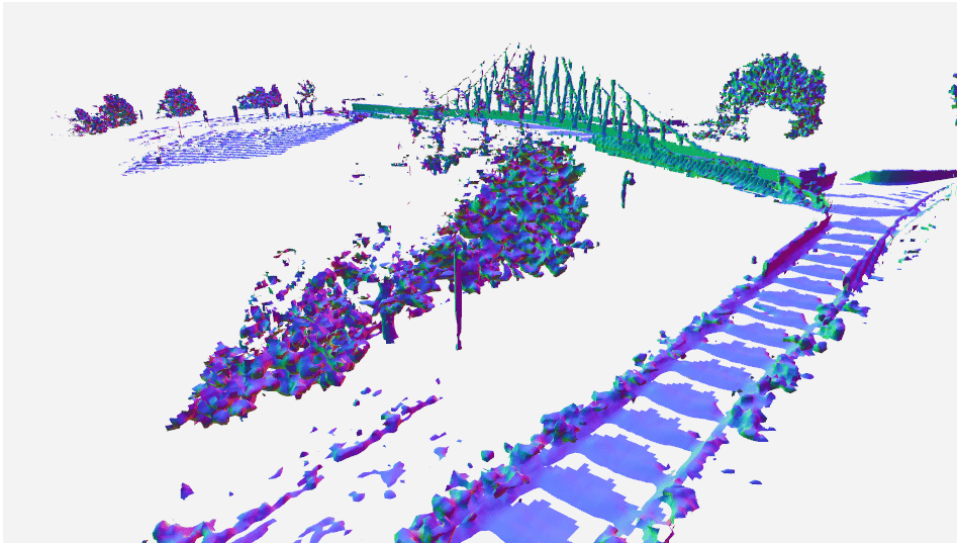


(b) Mesh creata con voxel di 0.2 $[m]$.

Figura 6.9: Confronto della mesh ricreata con un'immagine del ponte.



(a) Immagine satellitare del ponte.



(b) Mesh creata con voxel di 0.2[m].

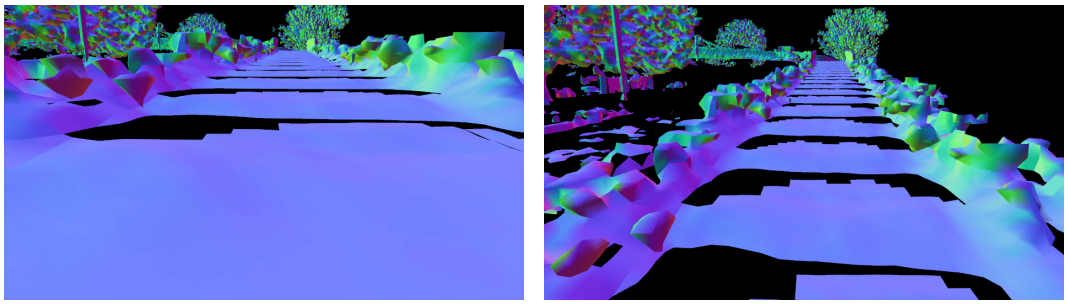
Figura 6.10: Confronto della mesh ricreata vista dall'alto con un'immagine satellitare del ponte.

Errori nelle mesh ricreate

Analizzando le immagini delle mesh ricostruite è possibile vedere come alcune superfici non siano stata ricostruite perfettamente in particolare il terreno è rappresentato a chiazze.
Questo tipo di errore è probabilmente causato da vari fattori alcuni di questi possono essere:

- La frequenza di aggiornamento della mesh ricreata da voxblox, infatti questa è stata setta a 0.5 secondi per cui una frequenza di aggiornamento della mesh più alta porterebbe ad una ricreazione della superficie più uniforme;

- La posizione del LiDAR, è un altro fattore che può giocare un ruolo importante. Infatti come si può vedere nella Figura 6.11a la mesh del terreno ricreata sembrerebbe uniforme tuttavia se si cambia punto di vista e guardandola quindi da un punto piu alto della mappa (Figura 6.11b) si nota che sono presenti zone del terreno non ricostruite generando cosi una superficie non uniforme.
- La tipologia del LiDAR usato, infatti esso è caratterizzato da un campo di vista limitato, ciò potrebbe aver causato una bassa densità della nuvola di punti al suolo e quindi per poter generare una mesh con una superficie uniforme del suolo. In Fiugura 6.12 è possibile vedere come la densità di punti misurati nel suo sia molto alta in una piccolissima zona davanti al rover ma allontanandosi si ha una densità di punti sempre minore.
- La frequenza di aggiornamento e la dimensione dei voxel della mesh, si è visto infatti come aumentando la frequenza di aggiornamento della mesh e riducendo a 0.15 $[m]$ si ha un lieve migliore ricostruzione del terreno infatti in questo caso la zona in cui il rover transita ha una copertura quasi completa, tuttavia appena ci si scosta da essa la superficie si presenta ancora ricostruita a tratti come si può vedere in Figura 6.13.



(a) Mesh generata vista ad una piccola altezza, paragonabile a quella del sensore.

(b) Mesh generata vista da una quota più alta rispetto a quella del sensore.

Figura 6.11: Errori della mesh dalla quota del rover a da un altezza piu elevata.

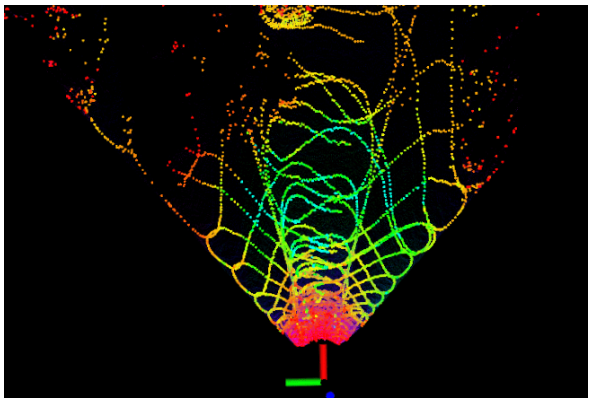


Figura 6.12: Distribuzione dei punti rivelati sul terreno dal sensore LiDAR.

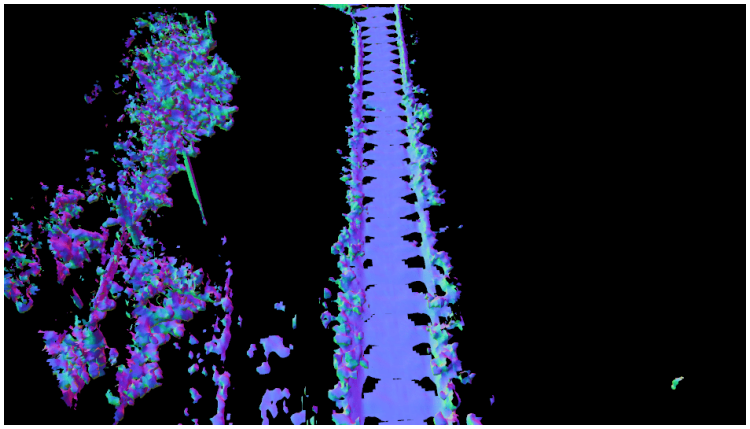


Figura 6.13: Miglioramento della ricostruzione della superficie del terreno ottenuto con un aggiornamento della mappa ogni 0,1 secondi e una dimensione di voxel di 0,15 $[m]$.

Percorso interno

Per quanto riguarda invece la traccia effettuata all'interno , ovvero il corridoio date le piccole distanze e la presenza di dettagli più piccoli si è preferito utilizzare una dimensione dei voxel più piccola ovvero di 0.05 $[m]$ la mappa ottenuta viene mostrata in Figura 6.14, e in Figura 6.15 viene mostrato un confronto con un'immagine della stereocamera del rover.

La scelta della dimensione del voxel utilizzata è stata effettuata considerando un confronto con una mappa realizzata con voxel di 0.1 $[m]$ e quella di 0.05$[m]$, infatti in Figura 6.16, è possibile vedere come la qualità di ricostruzione della mesh con voxel di 0,1 $[m]$ non sia quella ottimale alla ricostruzione dell'ambiente.

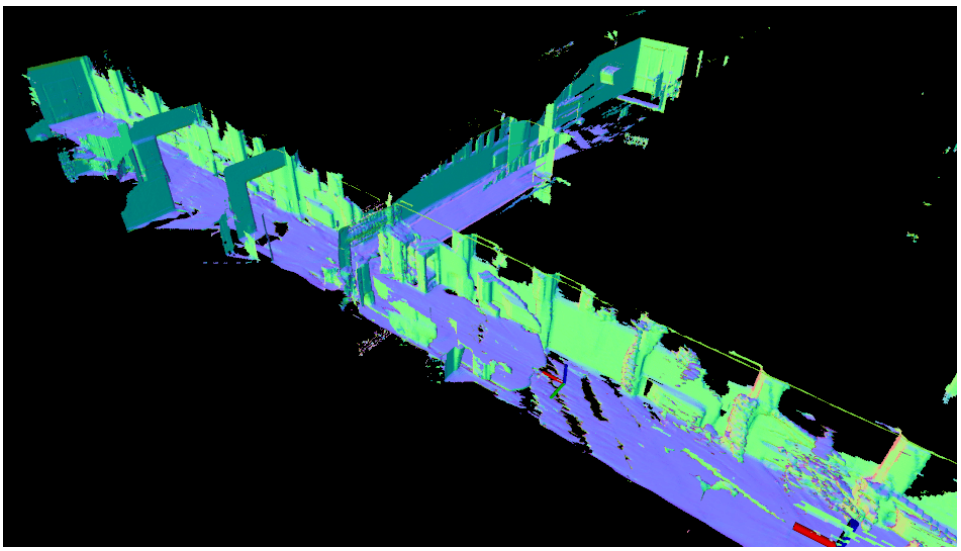
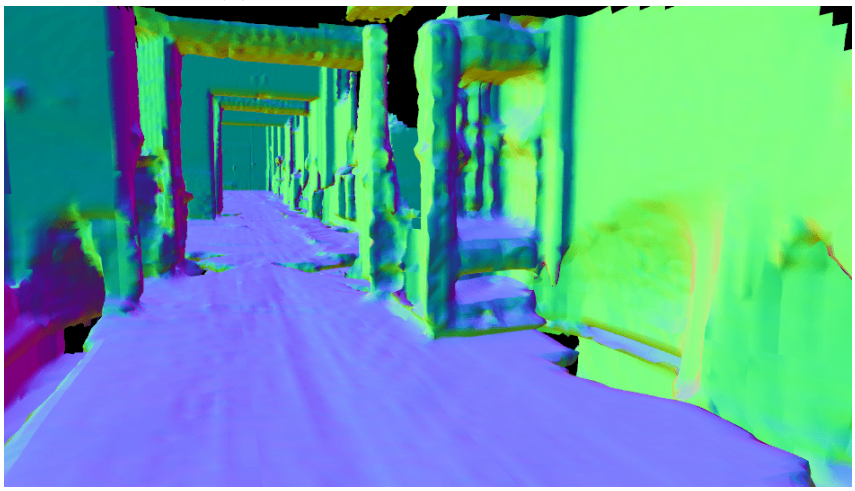


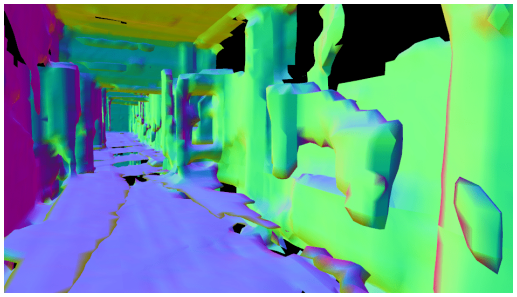
Figura 6.14: Rappresentazione della mesh del Corridoio in 3D.

(a) Immagine della stereocamera.

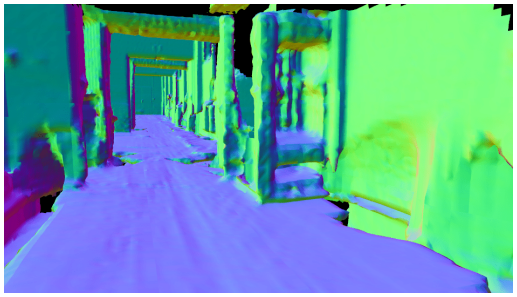


(b) Mesh generata del corridoio.

Figura 6.15: Confronto della mesh ricreata con un'immagine del corridoio.



(a) Mesh creata con voxel di 0.1 $[m]$.



(b) Mesh creata con voxel di 0.05 $[m]$.

Figura 6.16: Confronto della mesh con dimensioni di voxel diversi all'interno del corridoio.

Errori nelle mesh del corridoio ricreate

Una particolare errore nella ricostruzione del corridoio e stato riscontrato, come è avvenuto nella SLAM (vedi Sezione 6.1.1), nella zona del corridoio in cui sono presenti un gran numero di finestre. La riflessione su questa superficie ha posizionato dei punti all'esterno del corridoio come mostrato in Figura 6.17, con una conseguente generazione di una parete dietro la superficie del vetro. Tale errore può essere risolto sicuramente tramite altri sensori installati nel rover come ad esempio rilevatori di prossimità in grado di non avere errori di riflessione.

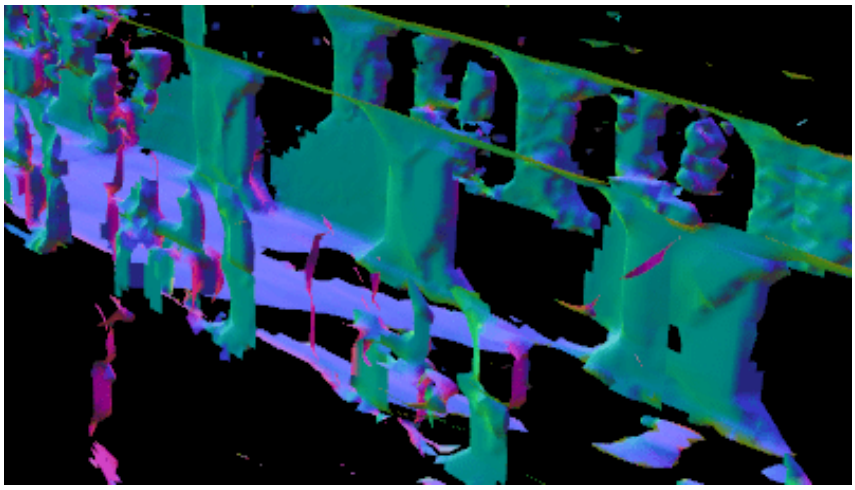


Figura 6.17: Errori causati dalla presenza delle finestre

6.2.2 Mappa TSDF ed ESDF generata

Le mappe TSDF e ESDF essendo mappe 3D basate su una griglia anch'essa 3D hanno una difficile visualizzazione all'occhio umano. Infatti visivamente sono griglia di puntini aventi ognuno un colore diverso che rappresenta l'informazione in esso contenuta. In Figura 6.18 viene mostrata la complessità di visualizzazione della mappa ESDF in 3D. Nel caso della mappa TSDF è quella relativa alla distanza dall'ostacolo misurata lungo il laser inviato dal rover, dando quindi un informazione della distanza solamente in profondità rispetto la vista del rover. Mentre nel caso della ESDF ad ogni punto viene correlata la distanza Euclidea dell'ostacolo più vicino, di conseguenza è possibile sapere che in una sfera avente raggio tale distanza non vi è la presenza di alcun ostacolo.

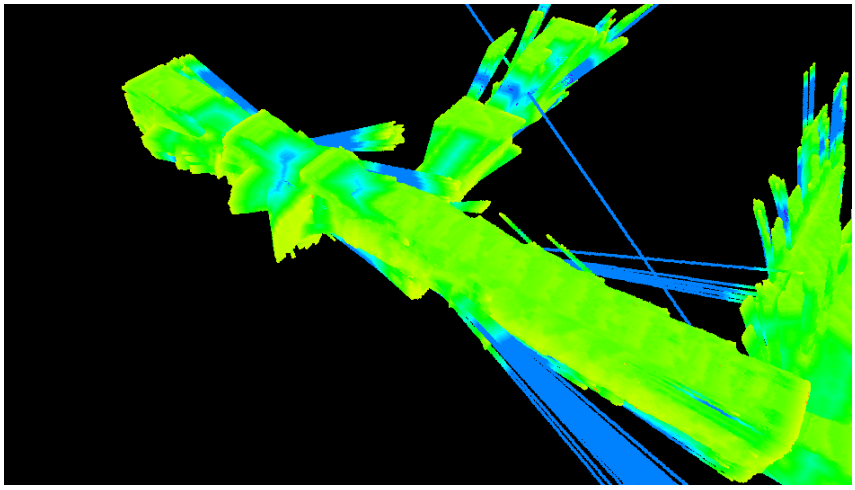


Figura 6.18: Mappa ESDF in 3D.

Per poter verificare la correttezza della mappa generata è possibile utilizzare una funzione definita in voxblox "*slice*" in grado di visualizzare solamente un piano di tale mappa ad una altezza desiderata, ottenendo cosi una mappa 2D più facilmente visualizzabile ed analizzabile. In Figura 6.19 viene mostrata la mappa 2D del corridoio ad una altezza di 30 cm dal pavimento del corridoio mentre in Figura 6.20 è rappresentata la mappa 2D TSDF. In queste due mappe il colore dei punti varia al variare del valore di distanza salvato in esso la scala inizia con un valore di distanza più alto (più lontano dall'ostacolo) con il colore viola fino ad arrivare rosso che individua l'ostacolo vero e proprio. I colori nel mezzo che effettuano la transazione tra i due sono rispettivamente il blu, il verde e poi il giallo.

Nelle due immagini è possibile visualizzare immediatamente la differenza tra le due mappa in quanto quella della TSDF al centro dove le pareti e gli ostacoli sono distanti identifica una distanza alta dall'ostacolo più vicino, mentre nella ESDF si ha un valore di distanza dall'ostacolo più basso indicando quindi

la presenza di un ostacolo piuttosto vicino, infatti "l'ostacolo" individuato più vicino è il pavimento.

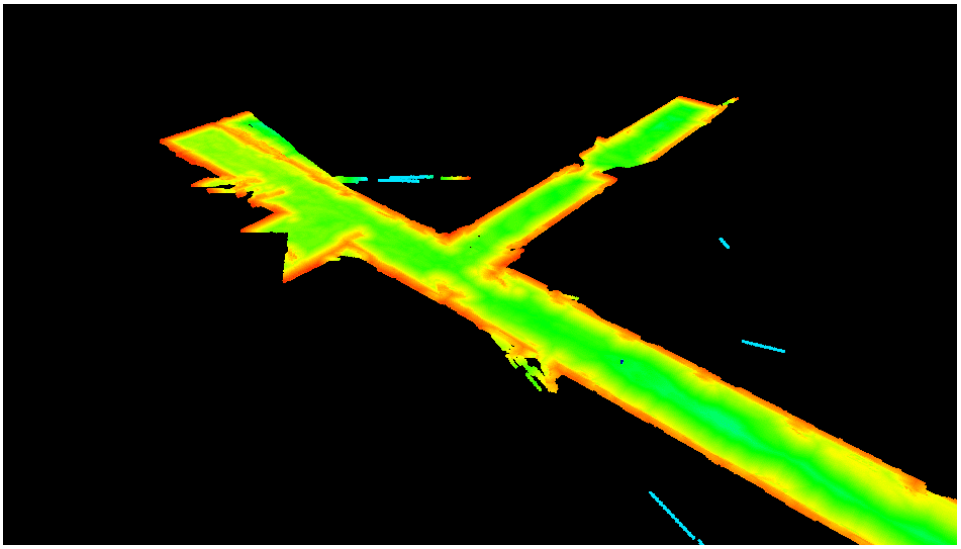


Figura 6.19: Mappa 2D ESDF del corridoio.

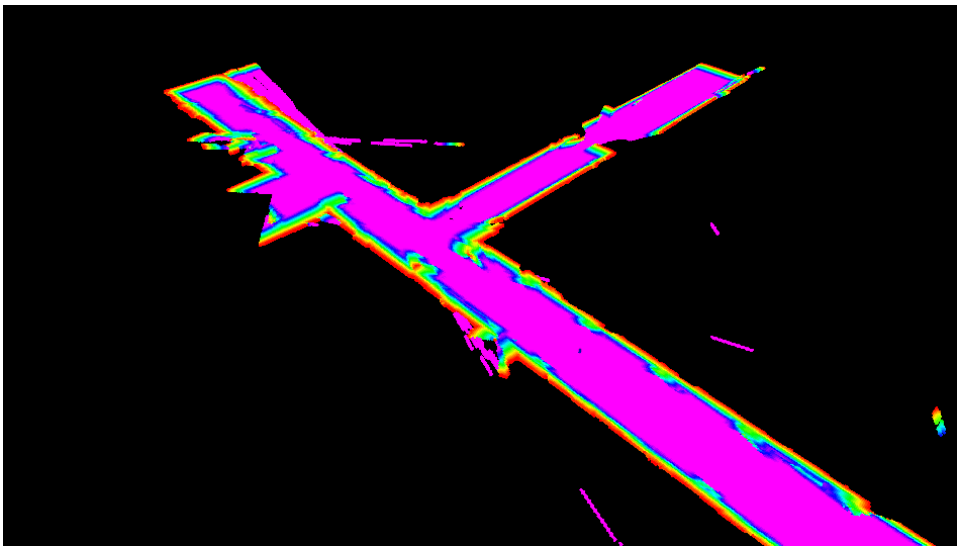


Figura 6.20: Mappa 2D TSDF del corridoio.

6.3 Analisi dell'errore della Mappa ESDF ricostruita

Per poter validare al meglio la mappa ESDF ricostruita è stata analizzata quella effettuata nella piazza del dipartimento di Psicologia, infatti grazie alla presenza di molteplici punti di riferimento individuabili all'interno e la possibilità di un confronto tramite i dati delle immagini satellitari è risultata un ottima candidata. Sono state generate diverse mappe 2D ad una diversa altezza in modo tale da individuarne successivamente la migliore da utilizzare per verificare l'errore effettuato dalla ricostruzione.

Un primo particolare che si può notare nelle diverse mappe è causato dalla presenza del lieve dislivello tra il punto di partenza e l'angolo della piazza diagonalmente opposto.
Infatti se si visiona la mappa 2D ad una quota bassa come il caso di Figura 6.21 nella parte in alto a destra viene individuata una zona gialla-rossa che indica quindi la presenza di un ostacolo vicino, tuttavia questo è riferito solamene alla presenza del terreno essendo che la quota in quel punto è più alta di quello di partenza e molto vicina alla quota a cui è stata generata la mappa ESDF 2D. Infatti nella Figura 6.22 che rappresenta una quota più alta mostra come la zona presenti un colore verde indicando la presenza di una superficie vicina ma più distante rispetto a quella rilevata al caso precedente.

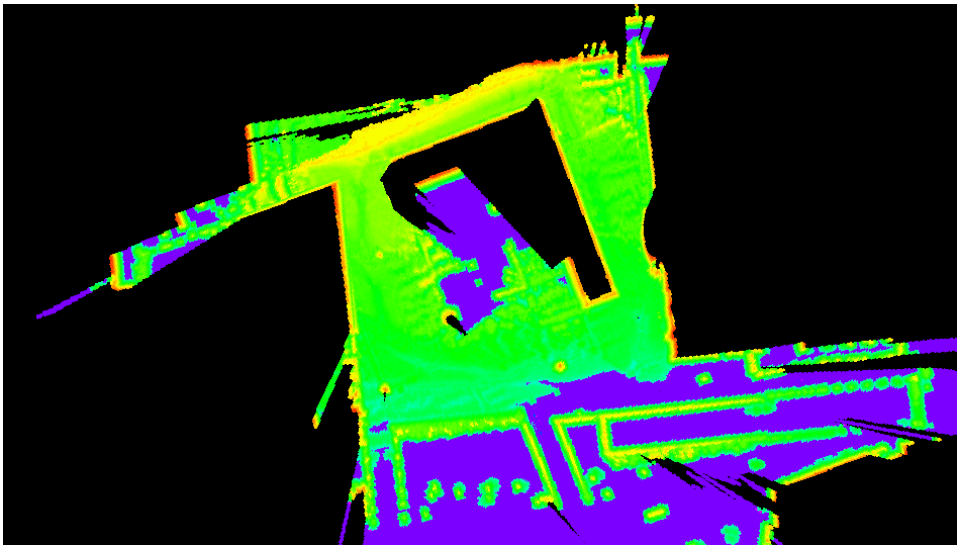


Figura 6.21: Mappa ESDF in 2D ad un altezza di 0.2 $[m]$

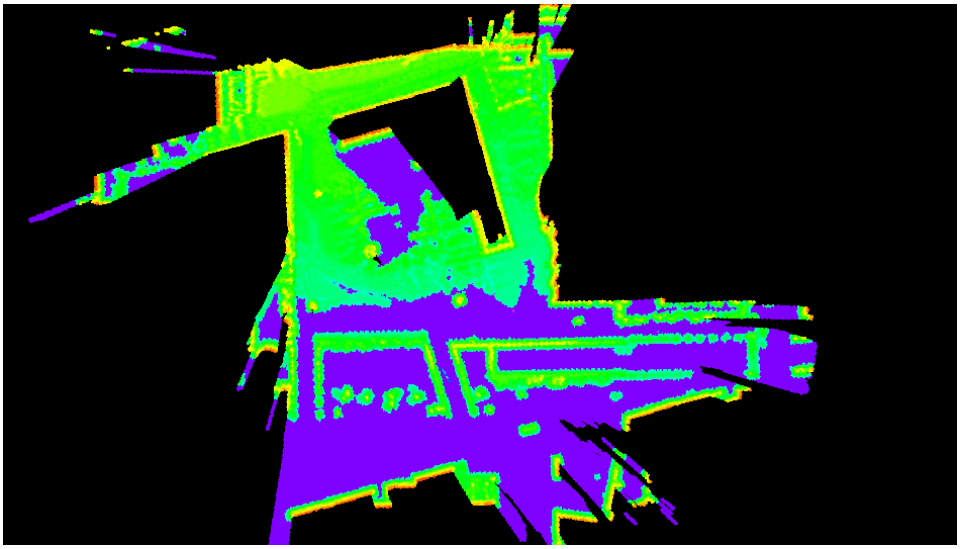


Figura 6.22: Mappa ESDF in 2D ad un altezza di 0.4 $[m]$

Per poter effettuare le analisi è stata utilizzata una mappa che dalla quota iniziale ha un altezza di 0.5 metri rappresentata in Figura 6.23. Si è scelta questa mappa in quanto si è rilevata quella che più rappresentava al meglio l'intera piazza e che presentava più nettamente il maggior numero di ostacoli e punti di riferimento.

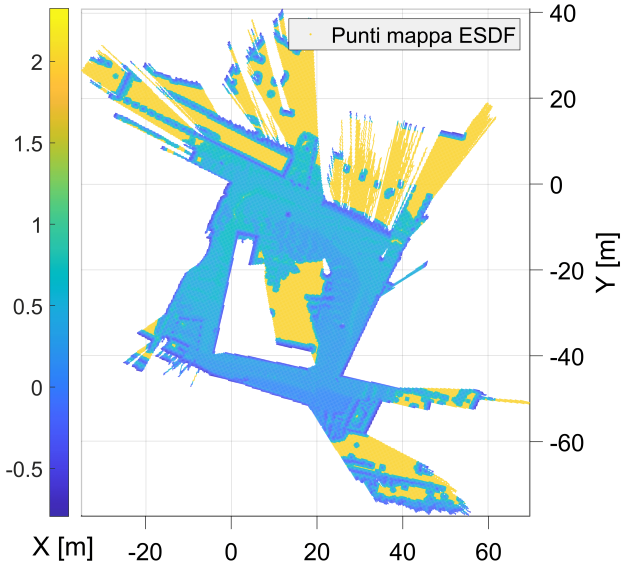


Figura 6.23: Mappa ESDF ad una quota di 0.5 [m] utilizzata per le analisi di validazione.

Sono stati individuati poi 25 punti ben identificabili all'interno sia della mappa ESDF (Figura 6.24) che dalle immagini satellitari (Figura 6.25).

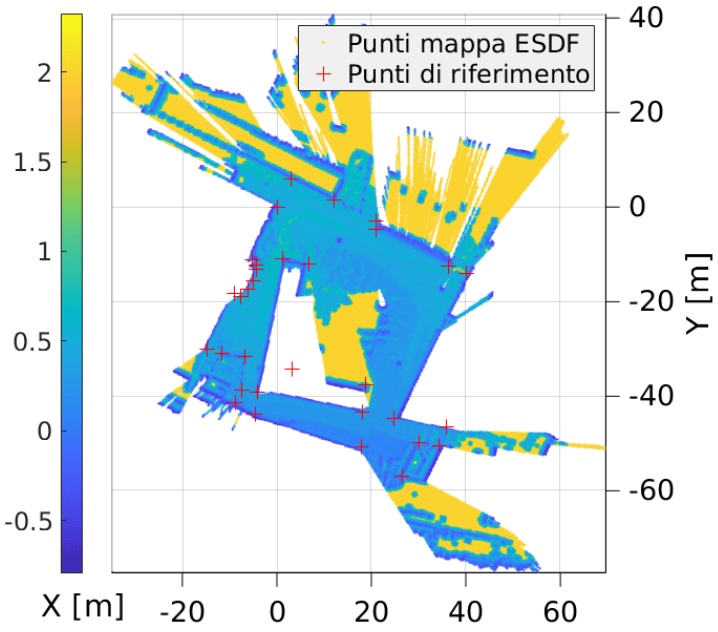


Figura 6.24: Mappa ESDF con i punti di riferimento individuati per il confronto con immagini satellitari.

Figura 6.25: Punti di riferimento per il confronto individuati nelle immagini satellitari.

Attraverso l'utilizzo delle funzioni di Matlab *"estgeotform3d"*, dando in input le corrispondenze tra i punti individuati nelle due mappe, è stato possibile calcolare una matrice di trasformazione e un vettore di traslazione che ha orientato e posizionato le due mappe in maniera tale da minimizzare l'errore di distanza tra le coppie di punti correlate. La mappa ESDF è stata poi trasformata in coordinate geografiche per effettuare un confronto visivo con le immagini satellitari. In Figura 6.26 viene mostra la sovrapposizione della mappa con le immagini satellitari e si può notare come in corrispondenza delle

zone blu scure della mappa ESDF, che corrispondo ad un valore vicino allo zero e di conseguenza molto vicini ad un ostacolo,sono presenti edifici o ostacoli come ad esempio il lampione al centro della piazza.

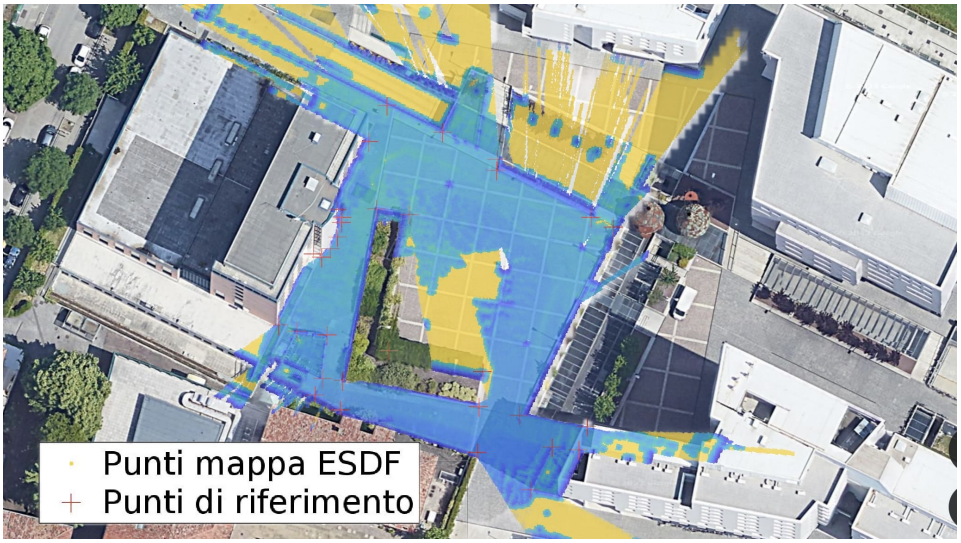


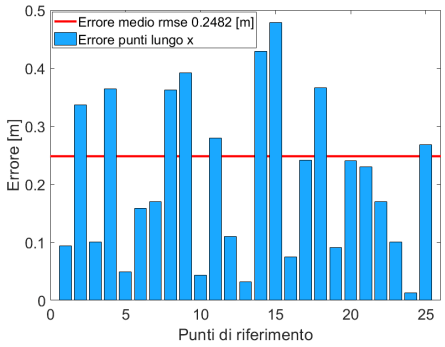
Figura 6.26: Sovrapposizione della mappa ESDF generata con le immagini satellitari, per effettuare un confronto visivo.

Analisi dell'errore

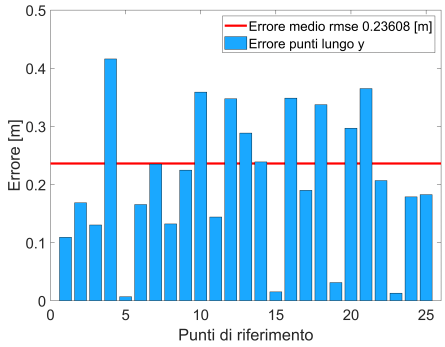
Sono stati poi calcolati gli errori dei punti sulla mappa rispetto a quelli delle immagini satellitari calcolandone l'errore medio per ogni coordinata, in x si è ottenuto un errore quadratico medio di 0.248 mentre in y l'errore è 0.236, e per la distanza effettiva dei due punti ottenendo cosi un errore quadratico medio delle distanze delle coppie di punti di:

$$rmse_{totale} = 0.3525[\text{m}]$$

Vengono riportati in Figura 6.27a e 6.27b i grafici che rappresentano gli errori dei punti rispetto all'errore medio per ogni coordinata. Mentre in Figura 6.28 la distribuzione degli errori totali, ove nell'asse delle ascisse vengono riportati i valori degli errori e in ordinate il numero di punti che ha effettuato tale errore.



(a) Errori delle dei punti lungo l'asse x rispetto al valore medio (rmse).



(b) Errori delle dei punti lungo l'asse y rispetto al valore medio (rmse).

Figura 6.27: Errori dei punti rispetto agli assi x e y relazionati all'errore quadratico medio l'ungo tale asse.

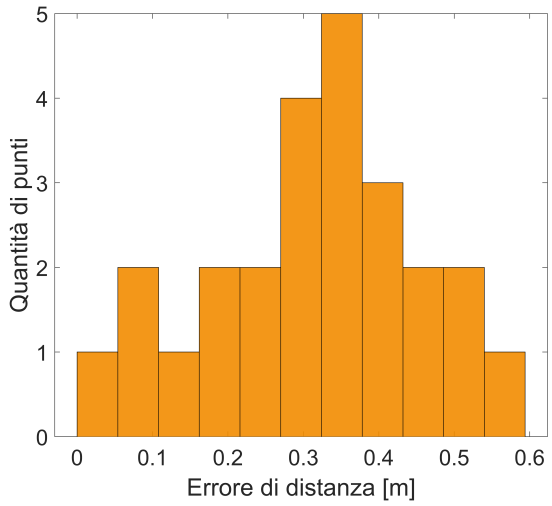


Figura 6.28: Distribuzione del numero di errori rispetto al valore d'errore ottenuto

Capitolo 7

Conclusioni

In questo lavoro di tesi sono state implementate e analizzate due librerie ROS, con il LiDAR equipaggiato dal rover MORPHES, grazie al quale si sono ottenuti buoni risultati sia nella SLAM che nella ricostruzione 3D con Voxblox con l'obbiettivo di poterli sfruttare per la navigazione autonoma.
Sulla base dei test effettuati ne è emerso come in base all'ambiente scelto, alla tipologia e alla dimensioni del veicolo utilizzato per la navigazione sia richiesta una configurazione diversa dei parametri. In particolare, è necessario aumentare l'accuratezza al diminuire delle dimensioni del veicolo o dell'ambiente, nel caso preso in esame uno dei fattori che ne migliora l'accuratezza è la riduzione della grandezza dei voxel. Tuttavia per poter navigare autonomamente il rover deve essere in grado di raggiungere l'obbiettivo muovendosi per evitare gli ostacoli in autonomia ed effettuare le scelte sulla sicurezza di un possibile percorso, per questo l'accuratezza deve comunque essere molto elevata in quanto anche i più piccoli ostacoli, eventualmente non individuati possono causare una collisione con il rover compromettendone ne la missione.

Come spiegato, i fattori chiave per ottenere una navigazione autonoma ottimale sono l'affidabilità, la precisione nella ricostruzione del percorso e della mappa e la capacità di distinguere quali sono i percorsi più sicuri da percorrere.
Per soddisfare questi requisiti è necessario che il carico computazionale richiesto al computer di bordo del rover, sia ridotto al minimo in modo tale quindi da risparmiare l'energia limitata a disposizione.I due algoritmi analizzati e testati in questo lavoro di tesi vanno incontro a queste richieste in quanto sfruttando le risorse del processore con tecniche innovative ed efficienti contenendo il costo computazionale senza però perdere accuratezza.

La libreria di voxblox implementata fornisce vari strumenti che possono migliorare e facilitare la pianificazione della traiettoria. In particolare le mappe TSDF e ESDF sono strumenti molto efficienti infatti sono in grado di fornire maggiori dettagli dell'ambiente circostante. Come spiegato infatti la loro struttura composta da una griglia tridimensionale di voxel racchiude in ogni punto le distanze dell'ostacolo più vicino e questo ne facilita la ricerca di una traietto-

ria più sicura. La mesh generata dalla TSDF può essere molto utile per alcune operazioni che il rover non è in grado di compiere in autonomia e è necessario l'intervento umano. In questi casi l'utilizzo di una mesh è sicuramente un elemento più facile da interpretazione rispetto ad una mappa composta da una nuvola di punti. Inoltre il peso della mesh è piuttosto contenuto, ad esempio nel caso della piazza preso in esame caratterizzato da una dimensione di voxel di 0,15 [m] ha un peso di 63 Mb l'equivalente circa di 6/7 foto con una risoluzione in HD. Il peso ridotto ne semplifica l'invio dei dati dato che generalmente si ha a disposizione un tempo piuttosto limitato per poter inviare i dati e di conseguenza possono essere mandati solamente una quantità di dati contenuta.

Durante questo lavoro è emerso come il modello del laser e la sua posizione di installazione sul rover è importante e influisce sulla ricostruzione della mappa 3D. Per migliorare e rendere più efficiente la ricostruzione una soluzione può essere quella di posizionare il LiDAR ad una altezza più elevata rispetto a quella attuale. Questo cambiamento non solo può essere un benefico per migliorare la mappatura 3D ma una posizione più elevata del sensore permetterebbe di vedere aree poste oltre ad ostacoli di piccole dimensioni consentendo quindi al rover di mapparle ancor prima di superali.Altre possibili soluzioni per testare la generazione della mappa di Voxblox potrebbe essere quello di utilizzare un sensore diverso con un campo di vista maggiore. Ad esempio con un LiDAR a 360 gradi sarà possibile mappare un volume maggiore nel minor tempo.

Le criticità più rilevanti riscontrate in questo elaborato sono quelle relative alla ricostruzione corridoio interno. La causa è da ricercarsi nella presenza di numerevoli fonti di rumore come la presenza di un pavimento lucido e grandi vetrate non ha stato possibile mappare correttamente l'ambiente generando quindi un alto rischio per la navigazione del rover in tale ambiente. Questo tipo di problematiche possono essere risolte ad esempio tramite delle tecniche di *"sensor fusion"*, ovvero con l'utilizzo di più sensori, in questo modo è possibile eliminare le fonti di errore riscontrate e rendere la navigazione del rover più sicura.

Grazie all'utilizzo di un ambiente di simulazione tutti i risultati ottenuti in questo elaborato sono ricostruiti come se avvenissero in tempo reale, e questo li rende perfetti nella navigazione autonoma. Un possibili sviluppo futuro di questo elaborato è quindi l'implementazione di algoritmi di pianificazione della traiettoria che sfruttino le potenzialità delle mappe generate da Voxoblox esposte in questo elaborato.

Bibliografia

[1] «Libya Wikipedia.» (), indirizzo: https://en.wikipedia.org/wiki/Rover_(space_exploration).

[2] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart e J. Nieto, «Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning,» in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1366–1373. DOI: 10.1109/IROS.2017.8202315.

[3] H. Oleynikova, Z. Taylor, M. Fehr, J. Nieto e R. Siegwart, *Voxblox: Building 3D Signed Distance Fields for Planning*, nov. 2016.

[4] H. Durrant-Whyte e T. Bailey, *Simultaneous localization and mapping: part I*, *IEEE Robotics & Automation Magazine*, vol. 13, n. 2, pp. 99–110, 2006. DOI: 10.1109/MRA.2006.1638022.

[5] T. Bailey e H. Durrant-Whyte, *Simultaneous localization and mapping (SLAM): part II*, *IEEE Robotics & Automation Magazine*, vol. 13, n. 3, pp. 108–117, 2006. DOI: 10.1109/MRA.2006.1678144.

[6] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte e M. Csorba, *A solution to the simultaneous localization and map building (SLAM) problem*, *IEEE Transactions on Robotics and Automation*, vol. 17, n. 3, pp. 229–241, 2001. DOI: 10.1109/70.938381.

[7] X. Lei, B. Feng, G. Wang, W. Liu e Y. Yang, *A Novel FastSLAM Framework Based on 2D Lidar for Autonomous Mobile Robot*, *Electronics*, vol. 9, p. 695, apr. 2020. DOI: 10.3390/electronics9040695.

[8] J. Mochnac, S. Marchevsky e P. Kocan, *Bayesian filtering techniques: Kalman and extended Kalman filter basics*, apr. 2009. DOI: 10.1109/RADIOELEK.2009.5158765.

[9] J. Zhang e S. Singh, *LOAM : Lidar Odometry and Mapping in real-time*, *Robotics: Science and Systems Conference (RSS)*, pp. 109–111, gen. 2014.

[10] T. Shan e B. Englot, «LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain,» ott. 2018, pp. 4758–4765. DOI: 10.1109/IROS.2018.8594299.

[11] M. Bosse e R. Zlot, «Continuous 3D scan-matching with a spinning 2D laser,» in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 4312–4319. DOI: 10.1109/ROBOT.2009.5152851.

[12] Y. Li e E. B. Olson, «Structure tensors for general purpose LIDAR feature extraction,» in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1869–1874. DOI: 10.1109/ICRA.2011.5979567.

[13] Y. Li e E. Olson, «Extracting general-purpose features from LIDAR data,» giu. 2010, pp. 1388–1393. DOI: 10.1109/ROBOT.2010.5509690.

[14] R. M. Murray, S. S. Sastry e L. Zexiang, *A Mathematical Introduction to Robotic Manipulation*, 1st. USA: CRC Press, Inc., 1994, ISBN: 0849379814.

[15] D. Werner, A. Al-Hamadi e P. Werner, «Truncated Signed Distance Function: Experiments on Voxel Size,» vol. 8815, ott. 2014, pp. 357–364, ISBN: 978-3-319-11754-6. DOI: 10.1007/978-3-319-11755-3_40.

[16] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss e W. Burgard, *OctoMap: An efficient probabilistic 3D mapping framework based on octrees*, *Autonomous Robots*, vol. 34, apr. 2013. DOI: 10.1007/s10514-012-9321-0.

[17] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto e R. Siegwart, «Signed Distance Fields: A Natural Representation for Both Mapping and Planning,» en, in *RSS 2016 Workshop: Geometry and Beyond - Representations, Physics, and Scene Understanding for Robotics*, Ann Arbor, MI: University of Michigan, 2016. DOI: 10.3929/ethz-a-010820134.

[18] U. Montanari, *A Method for Obtaining Skeletons Using a Quasi-Euclidean Distance*, *Journal of the ACM (JACM)*, vol. 15, pp. 600–624, 1968.

[19] S. Chiodini, A. Carron, M. Pertile et al., «MORPHEUS: A FIELD ROBOTICS TESTBED FOR SOIL SAMPLING AND AUTONOUMUS NAVIGATION,» dic. 2015.

[20] D. Paganini, «MORPHEUS: ideazione e progettazione del rover di Ateneo,» Tesi di laurea, Dipartimento di Ingegneria Industriale DII, A.S. 2016/2017.

[21] E. Morellatoi, «Algoritmi e software di controllo per la trazione del rover "Morpheus",» Tesi di laurea, Dipartimento di Ingegneria Industriale DII, A.S. 2015/2016.

[22] «Maxon Group.» (), indirizzo: https://www.maxongroup.com/.

[23] «NVIDIA Jetson TX1 Supercomputer-on-Module Drives Next Wave of Autonomous Machines.» (), indirizzo: https://developer.nvidia.com/blog/nvidia-jetson-tx1-supercomputer-on-module-drives-next-wave-of-autonomous-machines/.

[24] Livox. «Livox Horizon.» (2019), indirizzo: https://www.livoxtech.com/3296f540ecf5458a8829e01cf429798e/assets/horizon/Livox%20Horizon%20user%20manual%20v1.0.pdf.

[25] «Understanding nodes.» (), indirizzo: https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html.

[26] «livox_ros_driver.» (), indirizzo: http://wiki.ros.org/livox_ros_driver.

[27] W. Xu, Y. Cai, D. He, J. Lin e F. Zhang, *FAST-LIO2: Fast Direct LiDAR-Inertial Odometry*, *IEEE Transactions on Robotics*, vol. 38, pp. 1–21, ago. 2022. DOI: 10.1109/TRO.2022.3141876.

[28] «FAST_LIO.» (), indirizzo: https://github.com/hku-mars/FAST_LIO.

[29] W. Xu e F. Zhang, *FAST-LIO: A Fast, Robust LiDAR-inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter*, *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, mar. 2021. DOI: 10.1109/LRA.2021.3064227.

[30] Y. Cai, W. Xu e F. Zhang, *ikd-Tree: An Incremental K-D Tree for Robotic Applications*, feb. 2021.

[31] «Voxbox.» (), indirizzo: https://github.com/ekes/voxbox.