Università degli Studi di Padova

Dipartimento di Ingegneria Industriale

Corso di Laurea Magistrale in
Ingegneria Aerospaziale

Confronto di metodi SLAM basati su LiDAR mediante ambiente virtuale

Relatore:
Prof. Marco Pertile

Laureando:
Luca Girolimetto
2045219

Anno Accademico 2023/2024

A mio nonno,
che non c'è più, ma che ancora oggi
mi ispira ad essere quello che sono

Abstract

La SLAM (*Simultaneous Localization And Mapping*) è un processo computazionale che consente di mappare e simultaneamente di localizzare un generico dispositivo in movimento, come ad esempio un UAV (*Unmanned Aerial Vehicles*), in un ambiente ignoto. La SLAM basata su LiDAR (*Light Detection And Ranging*) utilizza impulsi laser, i quali, tramite la generazione di nuvole di punti tridimensionali, garantisco molti dati a disposizione dell'algoritmo.

In questa tesi per prima cosa viene sviluppato un ambiente virtuale di test in Matlab-Simulink tramite il quale sono estratte le misurazioni LiDAR di un drone in movimento e, in contemporanea, sono estratti i dati generati da una IMU (*Inertial Mesurament Unit*).

La prima parte della tesi si concentra su una breve introduzione teorica degli algoritmi base che consento la LiDAR-SLAM. Si procede illustrando i metodi LiDAR-SLAM già implementati in Matlab quali ICP e LOAM, introducendo per entrambi l'ottimizzazione del grafico di posizione che consente la chiusura del loop. Si confrontano quindi l'efficacia e la velocità di esecuzione di questi algoritmi, nonché la loro capacità di ricostruzione dell'ambiente scansionato. Si procede poi illustrando un semplice algoritmo LIO (*LiDAR Inertial Odometry*), il quale processa in modo separato (debole) i dati estratti dall'IMU. L'integrazione di un sistema separato permette di irrobustire e migliorare gli algoritmi presentati precedentemente.

Viene infine effettuato uno studio di sensibilità a diversi parametri, entrati in gioco nelle simulazioni qui presentate. Lo studio permetterà di osservare il comportamento dei diversi codici al variare di alcuni parametri fondamentali, per capirne l'influenza e determinarne il corretto settaggio.

Abstract

SLAM (*Simultaneous Localization And Mapping*) is a computational process that allows mapping and localizing simultaneously a generic moving device, such as an UAV (*Unmanned Aerial Vehicles*), in an unknow environment. LiDAR-based SLAM (*Light Detection And Ranging*) uses laser pulses, which, from 3D point cloud generation, guarantee lots of data for the algorithm.

Firstly, in this thesis is developed with Matlab-Simulink a virtual environment for testing, through which are extracted LiDAR measurements of a moving drone and, simultaneously, are generated and extracted IMU (*Inertial Measurement Unit*) measurements.

The first part of the thesis focuses on a brief theoretical introduction of basis LiDAR-based SLAM algorithm. Then are illustrated LiDAR-SLAM methods already implemented in Matlab, such as ICP e LOAM, introducing for both the pose graph optimization for loop closure. Then are compared efficiency and speed of these algorithms, as well as their ability of reconstructing the scanned environment. It then proceeds to show a simple LIO (*LiDAR Inertial Odometry*) algorithm, which processes in a separate way (loosely) the data extracted from IMU. The loosely integration of a separate system allows to make robust and speed up the algorithms presented previously.

Finally, a sensitivity study is carried out on various parameters that are needed in the simulations presented before. The study will allow us to observe the behavior of the different codes when some fundamental parameters vary, to understand their influence and determine their correct setting.

Indice

Capitolo 1

Introduzione

1.1 Cos'è la SLAM

Solo l'anno scorso più di 100.000 robot e veicoli autonomi sono stati prodotti nel mondo [1]. Tra questi molti erano UAV (*Unmanned Aerial Veichel*), che svolgono i più disparati compiti, tra i quali ispezione del territorio, trasporto merci, ricerca superstiti in territori difficoltosi, monitoraggio per manutenzione edifici e molto altro. Il mercato dei dispositivi autonomi è in rapida crescita grazie all'aumento della domanda. Traendo vantaggio da questa crescita, e soprattutto dalla disponibilità sempre più ampia di sensoristica a basso costo e potenza computazionale compatta, molti studi recenti sono stati condotti per permettere a veicoli, robot e velivoli autonomi di operare senza l'intervento umano in modo efficiente, preciso e sicuro. Per fare ciò un notevole contributo è sicuramente da annoverare alla SLAM (*Simultaneous Localization And Mapping*) e a tutti gli studi ad essa collegati.

La ricostruzione dell'ambiente in cui un veicolo si sta muovendo è di fondamentale importanza per la corretta localizzazione e movimentazione dello stesso in tale ambiente. Velivoli UAV autonomi di ricognizione, per scopi militari, o ancora veicoli quali macchine senza pilota o con sistemi di guida automatizzata e robot (siano essi industriali o domestici) necessitano di queste informazioni per orientarsi correttamente nello spazio. La localizzazione garantisce quindi le informazioni di feedback necessarie ai controlli di bordo del dispositivo, mentre la generazione di una mappa può fornire informazioni sull'ambiente per permettere

la pianificazione di traiettorie.

La SLAM è un processo computazionale che consente, tramite l'acquisizione strumentale di dati, di mappare un ambiente ignoto e simultaneamente di localizzare un generico dispositivo nella mappa così generata, ricostruendone la traiettoria percorsa. Esso si divide appunto in due fasi:

- **Localizzazione**, che consente di stimare la posizione e direzione del sensore/dispositivo, ricostruendone la traiettoria percorsa
- **Mappatura**, che consente di ricostruire una mappa, bi o tridimensionale dell'ambiente osservato.

Stimare la posizione di un dispositivo in un ambiente di cui si conosce la mappa e, viceversa, ricostruire una mappa conoscendo la traiettoria percorsa sono problemi relativamente facili. Non lo è altrettanto ricostruire simultaneamente la mappa e la traiettoria di un dispositivo in movimento in un ambiente ignoto o parzialmente ignoto. Da questo problema, iniziato a presentarsi con l'avvento della robotica moderna, si sono sviluppate una serie di formulazioni matematiche e algoritmi computazionali atti alla sua risoluzione.

1.2 Tipologie di SLAM

Grazie alla ricerca e agli avanzamenti tecnologici e informatici, nel tempo sono stati sviluppati molti modi per effettuare la SLAM, mediante la possibilità di utilizzare molti sensori diversi e di poterli accoppiare e fondere in svariate combinazioni. Di seguito sono mostrati in generale alcuni dei metodi più comuni, distinguendoli in funzione dello strumento base a cui fanno affidamento. Si possono osservare due grandi macro categorie, suddivise in base al sensore primario utilizzato. La SLAM basata su sistemi di visione e la SLAM basata su LiDAR [2].

La SLAM basata su **sistemi di visione** sfrutta come sensore principale una o più camere. Essa può essere monoculare o mediante un sistema di stereo camere. La ricostruzione della posizione relativa si basa su estrazione di descrittori tramite determinati detector e associazione delle *feature* tra frame consecutivi. Questo metodo dipende molto dal sistema di visione in esame. Lo svantaggio principale

di questa tecnica è la determinazione del fattore di scala.

La SLAM basata su **LiDAR** utilizza il LiDAR come sensore principale di acquisizione. Il LiDAR può essere da mono a tridimensionale in base a costi e necessità. Il LiDAR consente la generazione di molti dati sotto forma di punti tridimensionali di cui si conosce la distanza relativa al sensore. La determinazione della traiettoria percorsa si basa sulla determinazione dello spostamento relativo di due nuvole di punti. Il LiDAR è adatto alla ricostruzione di ambienti 3D, ha lungo raggio di misurazione, ampio campo di vista ed è pressoché indipendente dalla luminosità ambientale.

Esistono poi numerosi strumenti che, mediante fusione delle misurazioni, consento di rendere la SLAM più precisa, robusta e veloce. Alcuni esempi sono sicuramente gli strumenti inerziali **INS** (*inertial navigation system*) o le meno costose **IMU** (*Inertial Mesurament Unit*), le quali verranno approfondite in seguito. Il principale vantaggio dell'utilizzo di strumenti inerziali è la capacità di irrobustire il sistema e renderlo meno sensibile alle rapide variazioni di assetto e alla condizione di moto rapido. Essi sono però molto sensibili alla deriva, ovvero l'inevitabile deviazione della traiettoria stimata dalla traiettoria vera, dovuta ad un accumulo degli errori di misurazione su lungo periodo.

I sistemi **GNSS** (*Global Satelite Navigation System*) sono attualmente considerati la base su cui validare e confrontare gli algoritmi, i quali, tramite un sistema di satelliti, consentono la triangolazione del target garantendo la miglior resa in termini di deriva della traiettoria su lungo periodo. Il principale svantaggio di questi strumenti è indubbiamente la loro lenta risposta alle variazioni di assetto, cosa che invece può essere compensata da sistemi inerziali.

Esistono moltissimi algoritmi che permettono la stima della traiettoria percorsa, ma sono fortemente dipendenti dal sensore utilizzato e dalla applicazione specifica per la quale sono stati sviluppati. La scelta del sensore primario, sia esso un LiDAR o una telecamera, di implementare o meno una fusione, o ancora del tipo di algoritmo non è facilmente generalizzabile e dipende molto dall'applicazione, mentre il problema matematico presenta una formulazione specifica che sarà illustrata nella parte teorica.

1.3 Scopo della tesi

In questa tesi si è scelto di approfondire e sviluppare algoritmi SLAM basati su LiDAR 3D, data la loro versatilità e capacità di mappatura. Il LiDAR consente una mappatura dettagliata del territorio scansionato, inoltre, grazie alla fusione con strumenti inerziali, permette la ricostruzione del moto in condizioni di alta velocità. I sistemi LiDAR sono quasi del tutto indipendenti dalla condizione di luminosità dell'ambiente e tolleranti ad ambienti non statici e rumorosi. Si presta di conseguenza molto bene all'implementazione su sistemi UAV, i quali possono presentare condizioni operative molto varie e condizioni di volo poco statiche.

Al fine di validare gli algoritmi presentati in questa tesi, si è scelto di servirsi di un ambiente virtuale per la generazione di dati, data la sua comodità, rapidità e soprattutto l'assenza di costi. È uno strumento che permette l'immediata possibilità di validare i propri algoritmi senza la necessità di impostare un sistema di misurazione completo. Appoggiandosi all'interazione Matlab-Simulink-Unreal Engine, è stato sviluppato un ambiente virtuale di test tramite il quale sono state estratte le misurazioni LiDAR di un drone in movimento e, in contemporanea, sono state generate le misurazioni IMU.

In seguito ad una breve introduzione teorica degli algoritmi base che consentono la SLAM e in particolare la LiDAR-SLAM, si procede illustrando i metodi LiDAR-SLAM già implementati in Matlab quali ICP e LOAM, introducendo la *pose graph optimization*, che consente la chiusura del loop nel caso di passaggio in luoghi già visitati. La chiusura del loop consente un effettivo miglioramento in termini di ricostruzione della traiettoria e della mappatura.

Si procederà confrontando l'efficacia e la velocità di esecuzione di questi algoritmi, nonché la loro capacità di ricostruzione dell'ambiente scansionato, per poi illustrare un semplice algoritmo LIO (*LiDAR Inertial Odometry*), il quale processa in modo separato (debole) i dati estratti dall'IMU.

Viene infine effettuato uno studio di sensibilità a diversi parametri, entrati in gioco nelle simulazioni qui presentate. Lo studio permetterà di osservare il comportamento dei diversi codici al variare di alcuni parametri fondamentali, per

capirne l'influenza e determinarne il corretto settaggio.

Lo **scopo della tesi** è quello di confrontare algoritmi LiDAR SLAM, partendo dal confronto di algoritmi basati esclusivamente su LiDAR, per poi procedere confrontandoli con algoritmi che fondono LiDAR e IMU. Si cercherà quindi, al fine di impostare in modo ottimale le simulazioni, di osservare l'influenza di molteplici parametri per le simulazioni qui riportare, osservando quali siano gli andamenti e se esistano punti di ottimo. Inoltre, si mostrerà come sia possibile implementare il codice affidando la generazione dei dati ad un ambiente virtuale, senza la necessità di costose apparecchiature sperimentali.

Figura 1.1: Esempio di quadricottero adoperato per LiDAR SLAM [3]

Capitolo 2

Metodi SLAM basati su LiDAR

2.1 Descrizione matematica della SLAM

2.1.1 Formulazione matematica

Come già spiegato in precedenza, il problema della SLAM è un circolo vizioso in quanto per localizzare sarebbe necessaria una mappa, mentre per mappare sarebbe necessaria una buona localizzazione. Tuttavia, esiste una formulazione matematica rigorosa del problema che sfrutta la teoria della probabilità. Quanto scritto in seguito è stato ricavato da *"Simultaneous Localization and Mapping: parte 1 e 2"* ([4] e [5]). Definiamo le variabili in gioco:

- **Input**
  - Controllo: $U_{0:k} = \{u_1, u_2, u_3, ..., u_k\}$ contiene i vettori controllo u_k applicati a $k-1$ per portare il sistema in x_k
  - Osservazione: $Z_{0:k} = \{z_1, z_2, z_3, ..., z_k\}$ contiene tutte le osservazioni dei punti di riferimento (*landmark*).
- **Output**
  - Mappa: $m = \{m_1, m2, m3, .., m_k\}$ contiene le coordinate dei punti di riferimento
  - Vettore di stato: $X_{0:k} = \{x_0, x_1, x_2, ..., u_k\}$ contiene tutti gli stati x_k

Il problema della SLAM richiede che venga calcolata ad ogni iterazione la **probabilità congiunta a posteriori** (*Joint Posterior Probability*) come mostrato nell'equazione 2.1.

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \tag{2.1}$$

La probabilità condizionata indica la probabilità che un certo evento si verifichi data la probabilità che un altro si sia già verificato. Nel caso specifico le misurazioni $Z_{0:k}$ e i controlli $U_{0:k}$ si sono già verificati e hanno associata una probabilità. La valutazione di P permette di stabilire la probabilità con cui si avrà lo stato attuale x_k e la sua associata mappa m. La probabilità congiunta a posteriori è un tipo di probabilità condizionata che risulta dall'aggiornamento della probabilità a priori tramite i teoremi di Bayes.

Solitamente il problema viene affrontato tramite soluzioni ricorsive. Partendo da una stima della distribuzione di P (equazione 2.1), la probabilità congiunta associata allo stato attuale dipenderà da controllo e osservazione, i quali richiedono quindi di essere definiti tramite dei modelli. Qui entrano in gioco due modelli fondamentali derivanti dai teoremi di Bayes (figura 2.1):

- Il **modello di moto** (o controllo) descrive il moto relativo del sistema dallo stato x_{k-1} allo stato x_k dato il comando di controllo u_k. Esso è un processo di Markov in quanto x_k dipende esclusivamente dallo stato precedente del controllo ed è indipendente dalla mappa e dalla misurazione.

$$P(x_k|x_{k-1}, u_k) \tag{2.2}$$

- Il **modello di osservazione** descrive la probabilità di ottenere l'osservazione z_k dato in input lo stato x_k e la mappa m, relazionando quindi la misurazione con lo stato.

$$P(z_k|x_k, m) \tag{2.3}$$

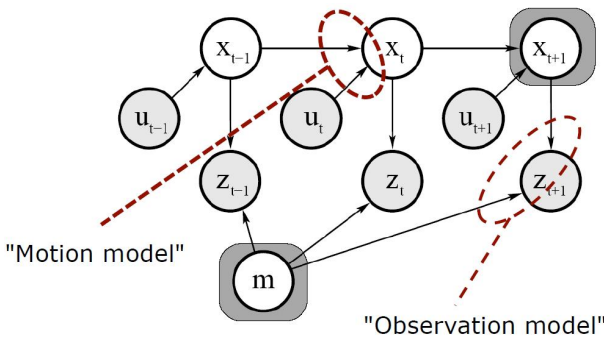


Figura 2.1: Rappresentazione grafica dei modelli di osservazione e di moto [6]

La SLAM viene solitamente implementata in un processo ricorsivo in due step del tipo predizione-correzione. In particolare:

- lo step **predittivo** sarà un aggiornamento nel tempo (*time update*). Esso viene generalizzato nell'equazione

$$P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \int P(x_k|x_{k-1}, u_k) \times P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0) dx_{k-1} \tag{2.4}$$

  Come si può notare le misurazioni in questo step non sono all'iterazione corrente. Lo step utilizza il modello di moto per dedurre lo stato corrente senza le misurazioni.

- lo step **correttivo** presenterà l'aggiornamento della misura (*measurements update*) mediante l'equazione

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|x_k, m)P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(Z_k|Z_{0:k-1}, U_{0:k})} \tag{2.5}$$

La procedura ricorsiva otterrà a convergenza la probabilità congiunta a posteriori presentata nell'equazione 2.1. La ricorsività dipende quindi da come vengono calcolati i modelli di Bayes.

Le tecniche di risoluzione del problema della SLAM possono essere suddivise in due gruppi principali [2]:

- **Filtering** (filtraggio)
  Si basano sulla stima incrementale dello stato del sistema tramite determinati filtri. Alcune delle tecniche più utilizzate sono mediante filtri di Kalman o mediate filtri particellari.

- **Smoothing** ("lisciatura")
  Si basano sulla tecnica di minimizzazione ai minimi quadrati dell'errore. Sono solitamente basati sull'ottimizzazione grafica.

Problema ai minimi quadrati

Molti degli algoritmi di ricostruzione sono basati sulla risoluzione di un problema ai minimi quadrati. Un problema basilare risolvibile mediante minimi quadrati è la regressione lineare, in cui si cerca una retta o curva che meglio rappresenti una serie di misurazioni (figura 2.2). In pratica il problema si risolve estraendo la somma degli errori tra dati e modello elevati al quadrato, ottenendo così la **fun-**

zione di costo (in inglese *cost* o *loss*). Si cercano quindi quei valori dei parametri del modello che minimizzino la funzione di costo. Il problema si può estendere a curve interpolanti generiche. Il problema di ottimizzazione si definisce lineare quando la funzione interpolante può essere descritta da una combinazione lineare delle variabili indipendenti e dai parametri non noti. L'equazione interpolante può quindi essere espressa come nell'equazione 2.6.

$$f(x; c_1, c_2, ..., c_M) = f(x; \vec{c}) = \sum_{j=1}^{M} c_j \Phi_j(x) \tag{2.6}$$

I residui (errori) verranno quindi definiti ad ogni valore dal set di dati (x_i, y_i) con $i = 1, ..., N$ come:

$$r_i = y_i - f(x_i, \vec{c}) \tag{2.7}$$

Il problema ai minimi quadrati cerca i parametri ottimali tali da minimizzare la funzione di costo S

$$S = \sum_{i=1}^{N} r_i^2 \tag{2.8}$$

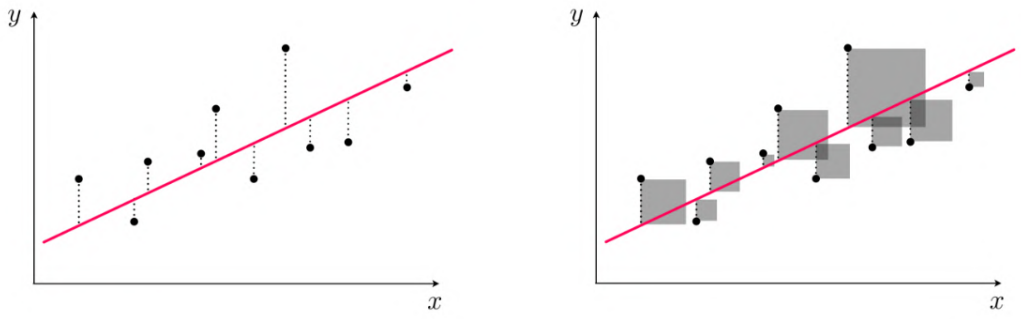


Figura 2.2: Esempio illustrativo di regressione lineare ai minimi quadrati

Dato che il modello contiene M parametri, la minimizzazione si ottiene valutando le derivate della funzione di costo rispetto ad ogni parametro c_j:

$$\frac{\partial S}{\partial c_j} = 2 \sum_{i=1}^{N} r_i \frac{\partial r_i}{\partial c_j} = -2 \sum_{i=1}^{N} r_i \frac{\partial f(x; \vec{c})}{\partial c_j} \quad con \quad j = 1, ..., M \tag{2.9}$$

Nel caso lineare, avendo descritto la funzione interpolante come nell'equazione 2.6, si avrà che $y_i \approx c_1\Phi_1(x_i) + c_2\Phi_2(x_i) + ... + c_j\Phi_j(x_i)$. Quindi denotando $X_{ij} = \Phi_j(x_i)$, si può scrivere l'equazione in forma matriciale $\vec{y} \approx X\vec{c}$. Il residuo

può essere quindi definito come $\vec{r} = \vec{y} - X\vec{c}$. Sviluppando, si può ottenere la generica funzione di costo da minimizzare.

$$S = |\vec{r}|^2 = \vec{y}^T\vec{y} - \vec{y}^T X\vec{c} - \vec{c}^T X^T\vec{y} + \vec{c}^T X^T X\vec{c} \tag{2.10}$$

Estraendo le derivate per minimizzare, tramite alcuni passaggi algebrici si ottiene

$$\frac{\partial S}{\partial \vec{c}} = 0 \rightarrow X^T X\vec{c} = X^T\vec{y} \rightarrow \vec{c} = (X^T X)^{-1} X^T Y \tag{2.11}$$

Il valore dei parametri è quindi determinabile. Il problema si può estendere a funzioni non lineari. Il problema ai minimi quadrati servirà per definire i metodi di registrazione quali ICP e LOAM, in quanto questi si basano solitamente sul minimizzare una funzione di costo. Una delle tecniche numeriche di calcolo di minimizzazione delle funzioni utilizzata nell'ICP è la SVD (*Single Value Decomposition*) [7].

Full vs Online SLAM

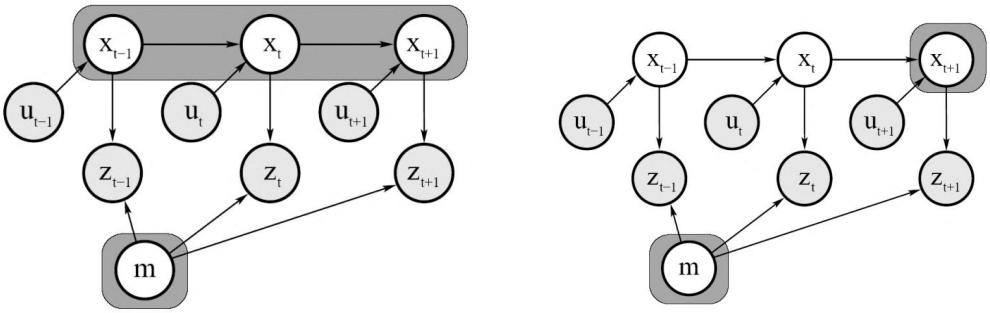
Si dice **full SLAM** quando si vuole conoscere l'intera traiettoria percorsa dal dispositivo durante il tempo di misurazione. La definizione matematica è riportata nell'equazione 2.12 [6].

$$P(X_{0:T}, m | Z_{1:T}, U_{1:T}) \tag{2.12}$$

Di contro invece, si dice **online SLAM** se si è interessati solo allo stato corrente e non all'intera traiettoria percorsa. La sua risoluzione si basa sulla marginalizzazione delle pose precedenti. La sua formulazione matematica è riportata nell'equazione 2.13.

$$P(x_t, m | Z_{1:t}, U_{1:t}) \tag{2.13}$$

In figura 2.3 sono mostrate graficamente le due tipologie di SLAM.
Il problema principale della SLAM è l'accumulo di incertezza che aumenta costantemente durante la misurazione, comportando l'inevitabile deriva della traiettoria nel tempo. Il costo computazionale necessario alla SLAM per ottenere risultati



(a) Full slam **(b)** Online slam

Figura 2.3: Rappresentazione grafica SLAM [6]

accettabili è alto, e ciò ne fa un problema difficile in quanto la potenza di calcolo dei veicoli che solitamente la effettuano è contenuta.

Mappatura

Una mappa è una rappresentazione della topologia di un ambiente. Nella SLAM si possono distinguere due gruppi di mappe:

- Le **mappe volumetriche** sono basate su nuvole di punti 3D, e creano una rappresentazione tridimensionale dell'ambiente, dando informazioni sulla geometria
- Le **mappe basate su *feature*** invece sono un accumulo di punti di riferimento (*landmark*) estratti tramite algoritmi specifici.

Il processo di mappatura consiste nel fondere conoscenze pregresse sull'ambiente con nuove misurazioni. Esso è quindi un processo iterativo di associazione di misure con una mappa già esistente. Nel caso di mappe volumetriche le misurazioni saranno i punti 3D estratti, mentre nel caso di mappe basate su *feature* esse deriveranno da un processo di estrazione.

2.1.2 Back-end e front-end

Andando ad osservare lo scheletro generale degli algoritmi SLAM si distinguono sempre due macro blocchi. Il **front-end** è responsabile dell'acquisizione e dell'elaborazione dei dati di ingresso tali da renderli utilizzabili dal **back-end**, il quale è responsabile del post processo e ottimizzazione. Il collegamento dal front-end al back-end si dice interfaccia.

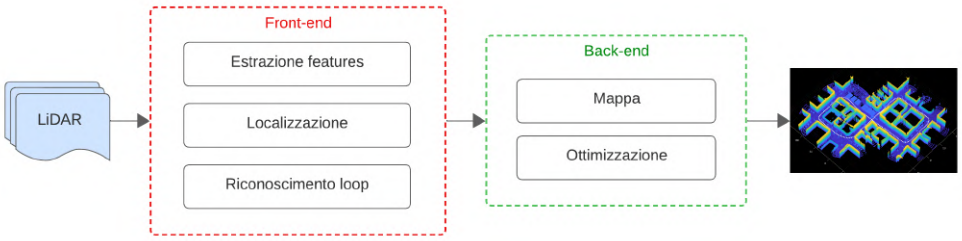


Figura 2.4: Schema della suddivisione front e back end nella LiDAR SLAM

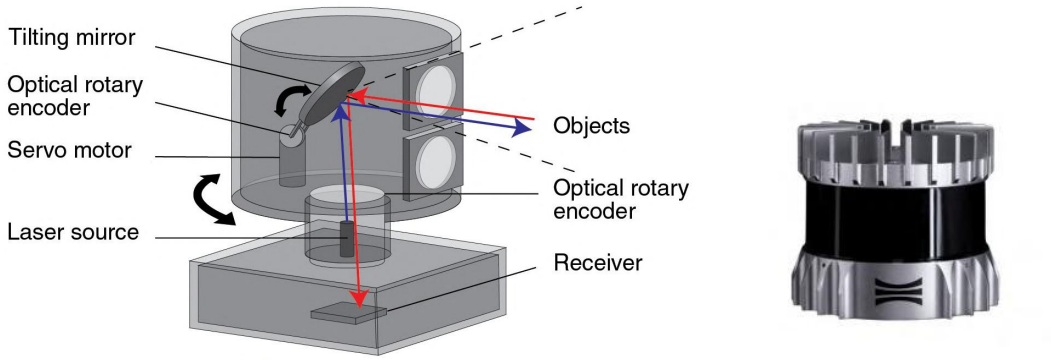
Nel caso specifico della LiDAR SLAM, tutte le operazione eseguite in seguito all'ingresso dei dati LiDAR, quali estrazione delle *feature*, registrazione e localizzazione, riconoscimento e stima della deriva per la chiusura del loop sono front-end. Mentre tutte le operazioni eseguite dopo l'immagazzinamento dei dati elaborati, allo scopo di ottimizzare, ricostruire e visualizzare la mappa e la traiettoria sono operazioni di back-end.

2.2 SLAM basata su LiDAR

Dopo aver definito meglio cos'è la SLAM in generale e come è suddiviso il *framework*, in questa sezione è descritta la SLAM basta su LiDAR, partendo proprio dal funzionamento dello strumento.

2.2.1 Funzionamento del LiDAR

L'acronimo LiDAR sta per *Light Detection And Ranging*. La sua tecnologia si basa sull'irradiazione di impulsi laser che, quando colpiscono una superficie, vengono riflessi e tornano al dispositivo, il quale raccoglie il segnale. Il LiDAR calcola la distanza dal punto di riflessione registrando il tempo di volo o, a seconda della tecnologia utilizzata, osservando lo sfasamento tra segnale di input e di output. Si possono distinguere varie parti tra cui un trasmettitore, il quale emette gli impulsi e un ricevitore che accoglie il segnale in ingresso. Il laser viene riflesso o retro-diffuso tornando al ricevitore (figura 2.5a). Nel caso di LiDAR basati su tempo di volo, un cronometro elettronico calcola il tempo trascorso tra emissione e ricezione del segnale. Un LiDAR utilizza tipicamente diversi componenti, tra cui laser, fotorilevatori e circuiti integrati di lettura (ROIC) con capacità di tempo di volo (TOF).



(a) Immagine illustrativa del funzionamento di un LiDAR a scansione meccanica

(b) Lidar Ouster OS1

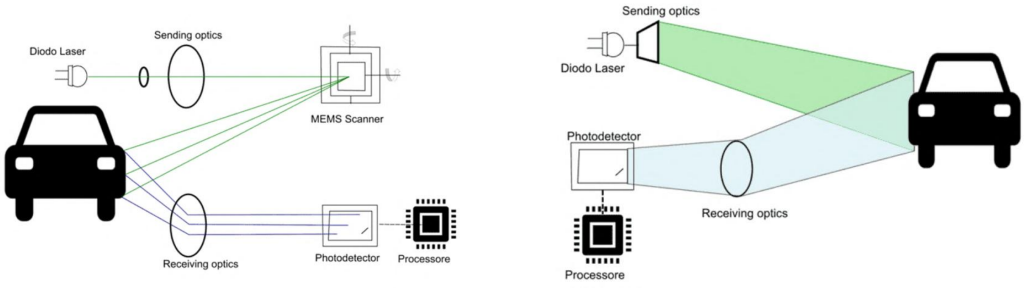
Figura 2.5: Strumento LiDAR

I LiDAR si possono distinguere in base al metodo di scansione utilizzato [8]:

- **Scansione meccanica:** un dispositivo meccanico fa ruotare fisicamente il laser e il ricevitore, ottenendo un campo di vista orizzontale a 360°. La scansione meccanica può avvenire ruotando meccanicamente l'emettitore e il rilevatore, oppure utilizzando dei microspecchi comandabili realizzati in tecnologia MEMS (*Micro Electro-Mechanical Systems*).

- **LiDAR a stato solido**: recentemente sviluppati, non prevedono alcuna parte in movimento. Questo permette di ottenere pesi contenuti in proporzione ai precedenti, mantenendo comunque alte performance. Un esempio è il Flash LiDAR che genera un fascio laser ampio tramite un diffusore, il quale ritorna ad una griglia di ricevitori su più linee.

- **LiDAR a stato semi-solido**: molto simili ai precedenti ma contengono solo una linea di ricevitori.



(a) Meccanica

(b) Flash

Figura 2.6: Tipologie di scansionamenti LiDAR

Proprietà del LiDAR

Nel *data sheet* di un LiDAR sono solitamente definiti alcuni parametri specifici. In tabella 2.1 sono riportati alcuni dati del sensore implementato in questa tesi, l'*Ouster OS1* (figura 2.5b), un LiDAR di tipo *flash* a stato solido *multi-beam*. Sebbene sia ampiamente configurabile, sono stati riportati solo i settaggi utilizzati nelle simulazioni.

Performance	Valore
Ottiche	
Range	100 m (> 90% con superficie lambertiana al 80%)
Range minimo	0,3 m
Accuratezza del range	±3 cm Lambert ±10 cm Retroriflettente
Risoluzione del range	0,1 - 0,8 cm
Risoluzione verticale	32 canali
Risoluzione orizzontale	1024 canali
Campo di vista verticale	45 °
Campo di vista orizzontale	360 °
Accuratezza angolare	$\pm 0,01°$ sia orizzontale che verticale
Frequenza di rotazione	10 Hz
Laser	
Classificazione	Class 1 eye-safe per IEC/EN 60825-1: 2014
Lunghezza d'onda	865 nm
Diametro fascio	9,5 mm
LiDAR Output	
Punti al secondo	fino a 655.360 (32 channel)

Tabella 2.1: Data sheet LiDAR Ouster OS1 [9]

In questa tesi si svilupperà un ambiente virtuale in cui simulare l'acquisizione LiDAR, di conseguenza si ometterà tutta la parte di calibrazione del sensore.

2.2.2 Workflow LiDAR SLAM

Non è semplice generalizzare l'algoritmo SLAM basato su LiDAR in quanto esistono molte varianti che ne affinano e complicano il processo. Tuttavia, si possono riscontrare dei blocchi presenti negli algoritmi SLAM con chiusura del loop. In figura 2.7 un diagramma ne raffigura la struttura.

Acquisizione

Si inizia prendendo in input la nube di punti generata dal LiDAR. Una nube di punti è un set di punti in uno spazio tridimensionale di cui si conosce la posizione

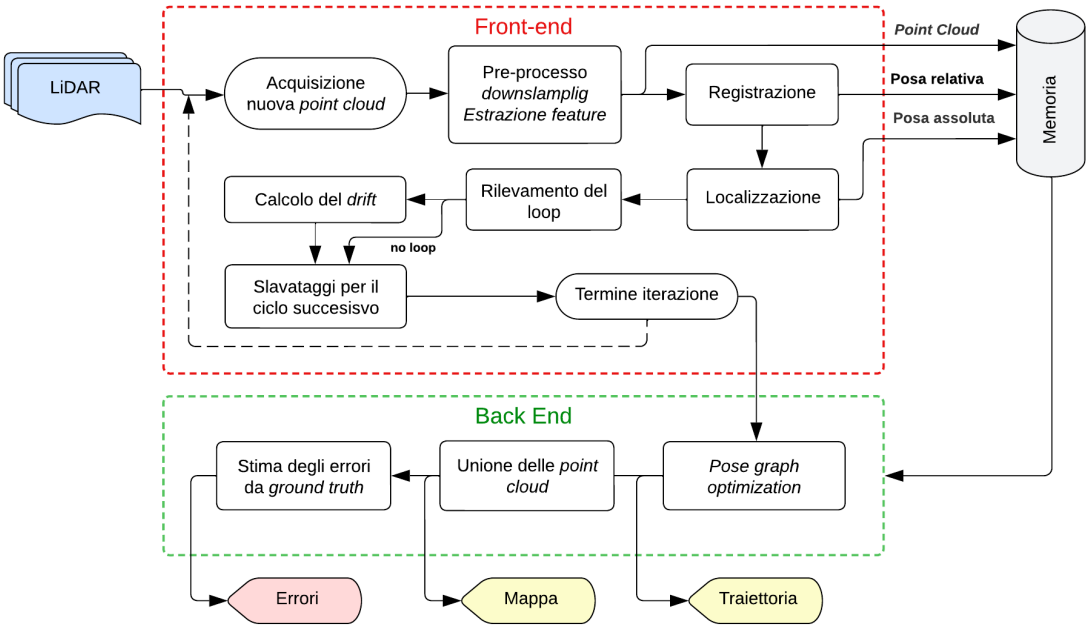


Figura 2.7: SLAM workflow

relativa al sensore, entro certi limiti di accuratezza dipendenti dallo strumento. L'acquisizione ha solitamente una frequenza di 10 Hz (ovvero 10 nuvole di punti al secondo). Il processamento di tutti i dati in uscita dal LiDAR comporterebbe un costo computazionale troppo elevato. Di conseguenza per ottenere la SLAM in tempo reale è necessario estrarre dei frame chiave, ovvero saltare un numero costante di acquisizioni LiDAR. Inoltre, in molti algoritmi avanzati, vengono saltati frame se si riconosce che il dispositivo si ferma. La scelta del passo dipende molto dalla velocità di moto del dispositivo.

Pre-processo

Si entra quindi nella fase di pre-processo della nube di punti, in cui si cerca di ridurre il numero di dati tramite opportune eliminazioni e sotto-campionamenti, necessari per ridurre il costo computazionale delle operazioni successive. A seconda dell'algoritmo utilizzato, in questa fase può essere presente l'estrazione delle *feature*. Il sotto-campionamento può essere anche effettuato direttamente su quest'ultime (vedi metodo LOAM). Questa fase è critica in quanto la bontà della successiva registrazione dipende in buona parte dalla qualità del sotto-campionamento. Se la nuvola di punti in ingresso è la prima, si passa direttamente alla memorizzazione, salvando la nube processata in modo tale da utilizzarla nell'iterazione successiva.

Registrazione

Per tutte le iterazioni successive si esegue nuovamente il pre-processo per poi passare alla fase di registrazione, nella quale viene calcolata la trasformazione che permette l'allineamento di due nubi di punti correlate, in uno stesso sistema di riferimento della nube considerata "fissa". La nube del frame precedente si considera "fissa" mentre la nube dell'iterazione corrente si considera "mobile". Successivamente si andranno a spiegare nel dettaglio gli algoritmi utilizzati da ICP e LOAM.

Localizzazione

In questa fase si calcola la posizione assoluta del dispositivo concatenando la matrice di trasformazione ottenuta nell'iterazione corrente con la matrice di trasformazione assoluta ottenuta dalle iterazione precedenti. La trasformazione assoluta così ottenuta sarà la trasformazione che porta dal sistema di riferimento iniziale al sistema di riferimento mobile dell'iterazione corrente. Si memorizza quindi la posizione assoluta per poterla concatenare all'iterazione successiva e la posizione relativa per la costruzione del grafo di posizione. Il grafo di posizione è necessario per la *pose graph optimization* in *back-end*. Eseguire la sola localizzazione, senza utilizzare una ottimizzazione in *back-end* comporta l'inevitabile deriva della stima.

Chiusura del loop

Il processo di chiusura del loop identifica il passaggio del sensore in un luogo già visitato, al fine di diminuire la deriva su lungo periodo della traiettoria. Esistono svariati modi per rilevare ed effettuare la chiusura del loop, che si basano su due fasi principali:

- **Rilevamento del loop:** in questa fase si cercano candidati in un'area contenuta nella sfera di ricerca prefissata. Se sono presenti punti interni alla sfera di ricerca si valuta la somiglianza tra i candidati interni e il frame attuale, calcolando la differenza tramite valori di *rms*. Il metodo utilizzato in questa tesi si basa sull'algoritmo *ScanContext*, che estrae delle *feature* ad hoc per rilevare la chiusura del loop.
- **Creazione e ottimizzazione del grafico di posa:** in questa fase si utilizzano i loop rilevati in front-end per minimizzare la deriva della traiettoria. Per prima cosa si crea il grafo di posa tramite le posizioni assolute e relative

memorizzate. Quindi si vanno ad utilizzare le relazioni ottenute nella rilevazione del loop per ottimizzare il grafo così ottenuto mediante algoritmi di ottimizzazione specifici.

Creazione della mappa

Per creare una mappa consistente dell'ambiente, si procede allineando le nuvole consecutive registrate, utilizzando le pose relative ottenute in seguito all'ottimizzazione. In questo modo tutte le nubi di punti verranno portate nel sistema di riferimento globale. Oltre alle mappe volumetriche, esistono altre tipologie di mappe ottenibili in seguito all'ottimizzazione. Una di queste è l'*occupancy map 3D*, nella quale viene eseguita una fusione dei punti allineati ottenuti, fornendo una risoluzione di input sotto forma di dimensione di *voxel* (controparte tridimensionale del pixel). Esistono inoltre mappe che uniscono le *feature* estratte (per esempio la LOAM map).

Stima dell'errore e tempi di simulazione

Una volta ottenuta la traiettoria la si può confrontare con la *ground truth*, valutando gli errori di *rms* definiti nell'equazione

$$rmse(\mathbf{x_{mes}}, \mathbf{x_{gt}}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x_{mes}} - \mathbf{x_{gt}}\|^2} \tag{2.14}$$

dove x_{mes} sono le coordinate estratte tramite SLAM, mentre x_{gt} sono le coordinate di *ground truth*. É inoltre possibile valutare gli errori angolari commessi durante il processo di registrazione. Essi possono essere calcolati sia osservando gli errori per componenti degli angoli di Eulero, sia valutando l'errore angolare complessivo definito mediante un solo asse di rotazione. Definendo R_{mes} la matrice di rotazione assoluta calcolata mediante registrazione e definendo R_{gt} la matrice di rotazione esatta, l'errore angolare assoluto commesso sarà definito nel modo seguente:

$$\theta = arccos\left(\frac{tr(R_{mes}R_{gt}^T) - 1}{2}\right) \tag{2.15}$$

Un altro parametro importante per valutare la capacità di un algoritmo SLAM è il tempo computazionale. Oltre che valutare il tempo di simulazione è anche importante valutare il tempo computazionale per singolo frame studiato. Come

definito nell'articolo [10] il PTPF (*Processing Time Per Frame*) non è altro che il rapporto tra il tempo computazionale totale e il numero di frame studiati. In seguito si analizzeranno anche i tempi della singola parte di codice per osservare quale di queste sia la più importante e quanto oscilli il tempo di iterazione.

2.3 Pre-processo

Nella fase di pre-processo sono solitamente tre le operazioni più importanti eseguite al fine di preparare la nube di punti per l'algoritmo di registrazione: rimozione di punti non necessari, sotto-campionamento ed estrazione delle *feature*.

2.3.1 Rimozione dei punti non necessari

Questa operazione è necessaria per rimuovere punti scansionati troppo lontani da non essere affidabili e punti troppo vicini, come per esempio le auto scansioni date da interferenze della struttura stessa del dispositivo. Di conseguenza, se si ha una nube di punti organizzata si possono estrarre gli indici associati ai punti tali che:

$$dist(P_i) < R_{min} \wedge dist(P_i) > R_{max} \tag{2.16}$$

Una volta estratti gli indici non si fa altro che rimuove dalla nube i punti associati a tali indici.In questa fase è frequente vedere anche la rimozione dei punti di suolo, molto più utilizzata nel caso di veicoli terrestri. Il procedimento è lo stesso, solo che l'estrazione degli indici è leggermente più complessa, in quanto bisogna indirizzare l'algoritmo di ricerca inserendo la direzione in cui si pensa sia presente il suolo e la distanza massima di ricerca. Per questi motivi, solitamente, questa operazione si applica a veicoli terrestri. Un dispositivo che varia la sua quota e il suo assetto in beccheggio e imbardata, come un drone ad esempio, non può associare dei parametri costanti per la rimozione dei punti di suolo.

In questa fase potrebbe essere anche effettuato il *denoysing*, ovvero la rimozione dei punti generati a causa di rumore nelle misurazioni o rumori ambientale. Matlab sconsiglia di effettuare questa operazione perché troppo onerosa computazionalmente.

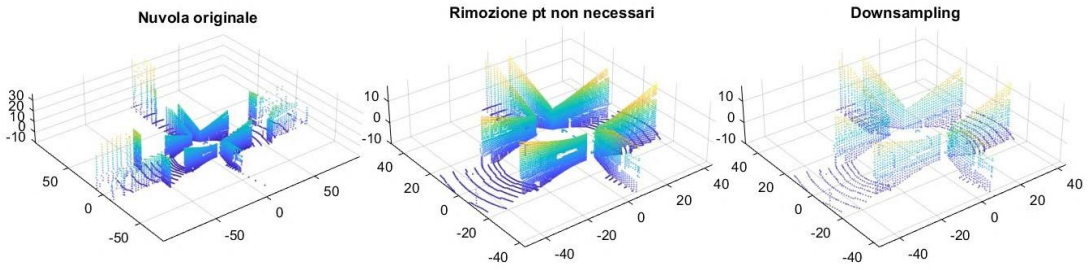


Figura 2.8: Esempio di rimozione punti superflui e sotto-campionamento

2.3.2 Downsampling

L'operazione sicuramente più delicata è il sotto-campionamento, necessario per ridurre il numero di punti da dare in input all'algoritmo di registrazione. Un sotto-campionamento troppo elevato non permetterà la corretta stima della trasformazione relativa tra nubi consecutive di punti, mentre un campionamento scarso comporterà un tempo computazionale elevato, non implicando necessariamente un miglioramento della ricostruzione. Matlab esegue il sotto-campionamento di una nuvola di punti in vari modi, due dei quali sono i più utilizzati:

- Con metodo **randomico**, fornendo una percentuale di punti che si vuole in output. Dando una percentuale minore si avranno meno punti in output.
- Con metodo ***grid average*** (a media di griglia). L'algoritmo calcola le dimensioni della cella allineata agli assi coordinati per l'intera nuvola di punti. Essa viene quindi divisa in celle di dimensioni specificate e i punti all'interno di ogni griglia sono fusi mediando la loro posizione, creando un unico punto posizionato nel baricentro.

Il metodo randomico è più efficiente in termini computazionali del metodo *grid average*, mentre il metodo *grid average* preserva meglio le caratteristiche geometriche della nube.

2.3.3 Estrazione delle feature

Alcuni algoritmi di ricostruzione (quali per esempio ICP o NDT) non necessitano di descrittori per la valutazione della trasformazione. Esistono invece algoritmi che si basano esclusivamente su *feature* di cui il precursore è LOAM. Esistono diverse tipologie di *feature* ma quelle presenti in Matlab sono due:

- **LOAM**: vengono estratti piani e spigoli a partire dalla nube di punti, basandosi sulla curvatura locale (figura 2.9)

- **FPFH** (*Fast point feature histogram*): invarianti all'assetto, rappresentano le proprietà della superficie in cui giace un punto. Sulla base di relazioni geometriche con i KNN di tale punto, calcolano la normale alla superficie, creando un istogramma dipendente da queste informazioni [11].

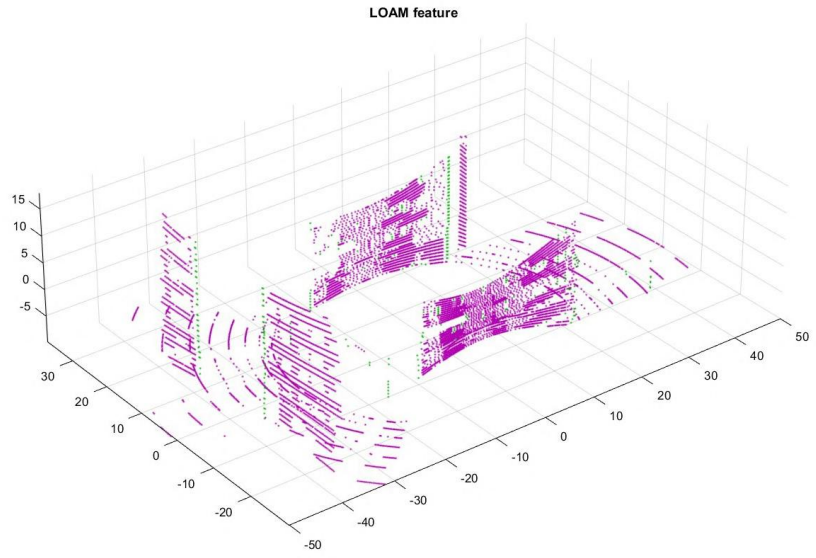


Figura 2.9: Esempio di descrittori LOAM, punti piani in magenta, punti spigolo in verde

2.4 Algoritmi di registrazione

La registrazione è il processo di calcolo della trasformazione (definita come traslazione e rotazione) di due nuvole di punti. Gli algoritmi di registrazione possono essere distinti in due grandi categorie (secondo l'articolo [10]):

- I metodi basati su punti (*point-wise*) stimano la trasformazione relativa valutando i punti grezzi in uscita dal LiDAR.

- I metodi basati su *feature* (*feature-wise*) dapprima estraggono dei descrittori dalle nuvole di punti come per esempio spigoli e piani, cercando poi corrispondenze tra scansioni successive e ricavando la trasformazione relativa basandosi su queste ultime.

In tabella 2.2 è riportato un beve riassunto dei principali metodi di registrazione sviluppati, facendo una distinzione tra punti e *feature* e indicando autore e anno di sviluppo.

Metodo	Autore	Anno	Base
ICP [12]	Besl et al.	1992	Punti
G-ICP [13]	Segal et al.	2009	Punti
VGICP [14]	Koide et al.	2020	Punti
NDT [15]	Biber et al.	2003	Punti
LOAM[16]	Zhang et al.	2014	Feature
LeGO LOAM [17]	Shan et al.	2018	Feature
LIO-mapping [18]	Ye et al.	2019	Feature
Fast LOAM [19]	Han	2020	Feature

Tabella 2.2: Elenco dei più importanti metodi di registrazione [10]

Di seguito si andranno ad illustrare nel dettaglio i metodi ICP e LOAM, i metodi cardine del *point-wise* e *feature-wise*, i quali verranno poi implementati nella parte pratica.

2.4.1 ICP

Il primo algoritmo sviluppato per la LiDAR *odometry*, e precursore di tutti i metodi *point-wise*, venne sviluppato da Besl et al. nel 1992 [12]. Questo algoritmo di registrazione relaziona direttamente due nubi di punti a livello dei punti stessi. I principali svantaggi sono il peso computazionale, la forte dipendenza dalla trasformazione iniziale di primo tentativo, dovuta alla non convessità dell'ottimizzazione, e la sensibilità a oggetti in movimento che comporta una ulteriore non convessità. L'algoritmo si basa sulla minimizzazione dell'errore del point-to-point. L'equazione 2.17 rappresenta la funzione di costo di partenza (procedura semplificata di calcolo [10]).

$$T_{k+1}^{k} = argmin \left\{ \frac{1}{2} \sum_{i=1}^{N} ||x_{(k,i)} - \left(R_{k+1}^{k} x_{(k+1,i)} + t_{k+1}^{k}\right)||^2 \right\} \tag{2.17}$$

L'algoritmo procede rimuovendo il baricentro dalle due nubi di punti. Il baricentro è calcolato con l'equazione 2.18 e poi sottratto con l'equazione 2.19.

$$\bar{x}_k = \frac{1}{N} \sum_{i=1}^{N} x_{k,i} \tag{2.18}$$

$$x'_{(k,i)} = x_{(k,i)} - \bar{x}_k \tag{2.19}$$

Inserendo la nube così sottratta nell'equazione 2.17 si ottiene

$$T_{k+1}^{k} = argmin \left\{ \frac{1}{2} \sum_{i=1}^{N} ||x'_{(k,i)} - Rx'_{(k+1,i)}||^2 + ||\bar{x}_k - R\bar{x}_{k+1} - t||^2 \right\} \tag{2.20}$$

Calcolando il minimo della derivata tramite metodo SVD (*Single Value Decomposition*) si ricava la trasformazione relativa associata alle due nubi di punti.

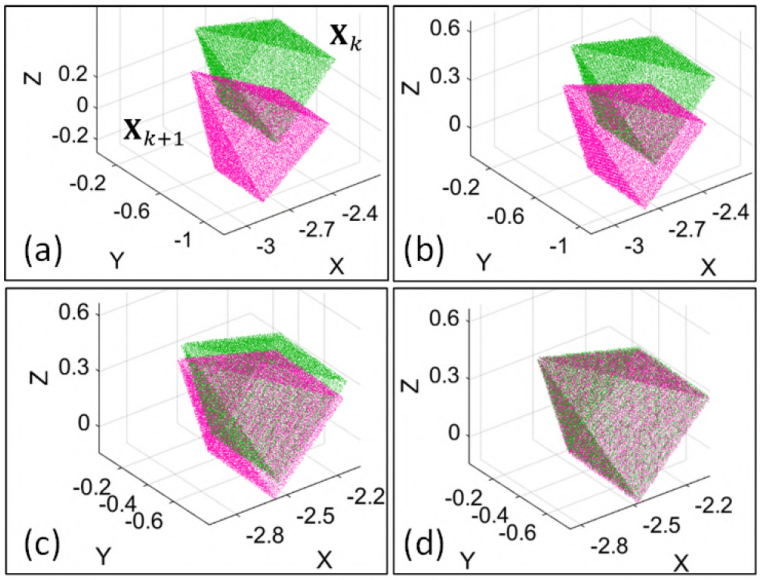


Figura 2.10: Esempio trasformazione a varie iterazioni

2.4.2 LOAM

L'algoritmo di ricostruzione LOAM è stato sviluppato da Zhang et al. nel 2014 [16], e rappresenta il precursore di molti dei metodi basati su *feature*. Il metodo si compone di tre step fondamentali (procedura semplificata di calcolo [10]):

1. **Estrazione delle *feature***
   I descrittori vengo estratti in base alla curvatura locale dei punti. L'equazione 2.21 calcola la curvatura locale.

$$c_i = \frac{1}{N_s \left\| x_{ki} \right\|} \left\| \sum_{j \in S_{k,i}^{m} \wedge j \neq i} (x_{k,i} - x_{k,j}) \right\| \tag{2.21}$$

   $S_{k,i}^{m}$ è l'anello m della nube di punti X_k in un intorno di $x_{k,i}$, mentre $x_{k,j}$ è un punto in tale anello e N_s è il numero di punti in tale anello più il punto

$x_{k,i}$. Il metodo LOAM estrae piani e spigoli. La distinzione sarà data dalla curvatura:

- curvature $c_i > c_{th}$ saranno spigoli e verranno chiamati $x^{em}_{k,i}$
- curvature $c_i < c_{th}$ saranno piani e verranno chiamati $x^{pm}_{k,i}$

Viene suddiviso lo spazio in sotto-regioni e per ogni sotto-regione vengo estratti 2 spigoli e 4 piani. Il risultato dell'estrazione sarà:

$$x^{s,m}_{k,i} = \begin{cases} x^{e}_{k,i} \in x^{e,m}_{k,i} \mid c_i > c_{th} \wedge N^{e}_{k,i} \leq 2 \\ x^{p}_{k,i} \in x^{p,m}_{k,i} \mid c_i < c_{th} \wedge N^{p}_{k,i} \leq 4 \end{cases} \tag{2.22}$$

2. **Odometria**
   Una volta estratti spigoli e piani si esegue una trasformazione di primo tentativo della nube X_k tramite $T^L_{k,k+1}$ avvicinandola alla nube X_{k+1}.

$$\tilde{X}_k = T^L_{k,k+1} X_k \tag{2.23}$$

Eseguita la trasformazione di primo tentativo le corrispondenze andranno cercate tra X_{k+1} e $\tilde{X}_k$. Estratte le corrispondenze tramite KNN (*k-nearest-neighbors*) si procede valutando le distanze (equazione 2.24).

$$\begin{cases} d^{e}_{k+1,i} = \dfrac{\left|(x^{e}_{k+1,i} - \tilde{x}^{e}_{k,j}) \times (x^{e}_{k+1,i} - \tilde{x}^{e}_{k,l})\right|}{\left|(\tilde{x}^{e}_{k,j} - \tilde{x}^{e}_{k,l})\right|} \\ d^{p}_{k+1,i} = \dfrac{\left|(x^{p}_{k+1,i} - \tilde{x}^{p}_{k,j}) \cdot (\tilde{x}^{p}_{k,j} - \tilde{x}^{p}_{k,l}) \times (\tilde{x}^{p}_{k,j} - \tilde{x}^{p}_{k,m})\right|}{\left|(\tilde{x}^{p}_{k,j} - \tilde{x}^{p}_{k,l}) \times (\tilde{x}^{p}_{k,j} - \tilde{x}^{p}_{k,m})\right|} \end{cases} \tag{2.24}$$

Valutate le distanze si possono ricavare le relazioni geometriche tra piani e spigoli

$$\begin{cases} f(x^{e}_{k+1,i}, T_{k+1}) = d^{e}_{k+1,i} \\ f(x^{p}_{k+1,i}, T_{k+1}) = d^{p}_{k+1,i} \end{cases} \tag{2.25}$$

Si ricava quindi la funzione di costo da minimizzare basata sulle relazioni geometriche precedenti.

$$T^k_{k+1} = argmin \left\{ \frac{1}{2} \sum_{i=1}^{N_e} ||\omega_{k+1,i} f(x^{e}_{k+1,i}, T_{k+1})||^2 + \frac{1}{2} \sum_{i=1}^{N_p} ||\omega_{k+1,i} f(x^{p}_{k+1,i}, T_{k+1})||^2 \right\} \tag{2.26}$$

3. **Mappatura**
   La parte di mappatura va a relazionare l'odometria appena eseguita con la mappa generale, in modo da riportare la nube nel sistema di riferimento globale.

$$X_{k+1}^{W} = T_{k,k+1}^{W} X_{k+1} \tag{2.27}$$

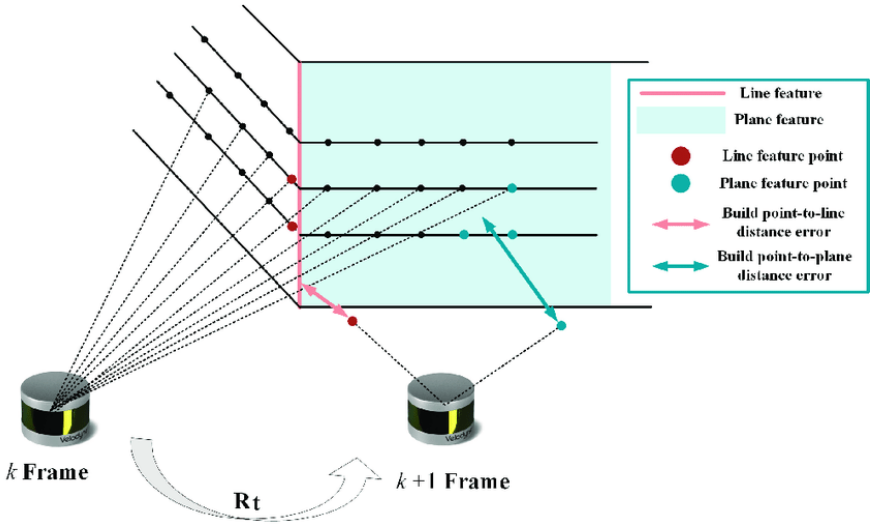


Figura 2.11: Esempio di *feature* estratte con metodo LOAM

2.5 Sistemi di riferimento

Definiamo due sistemi di riferimento:

- **Globale**: sistema inerziale rigido levogiro vincolato al terreno, con asse z rivolto verso l'alto
- **Locale** (corpo): non inerziale definito con asse x rivolto nella direzione della velocità, asse z che può essere verso l'alto nel caso di sistema ENU (Est Nord Up) o verso il basso nel sistema NED (Nord Est Down) e asse y che completa la terna levogira.

Lo stato di un dispositivo in un sistema tridimensionale può essere definito rispetto al sistema di riferimento globale nel seguente modo:

$$x_k = (X_k, Y_k, Z_k, \theta_k, \phi_k, \psi_k) \tag{2.28}$$

dove k è l'indice dell'iterazione corrente, x_k è definito rispetto al sistema di riferimento globale, (X_k, Y_k, Z_k) sono le coordinate dell'origine del sistema di riferi-

mento assi corpo (NED o ENU) associato al dispositivo, mentre θ_k, ϕ_k, ψ_k sono ripetitivamente rollio, beccheggio e imbardata del sistema assi-corpo nel sistema globale.
La registrazione consente di determinare la trasformazione relativa tra sistema di riferimento corpo iniziale e sistema di riferimento corpo finale (figura 2.12).

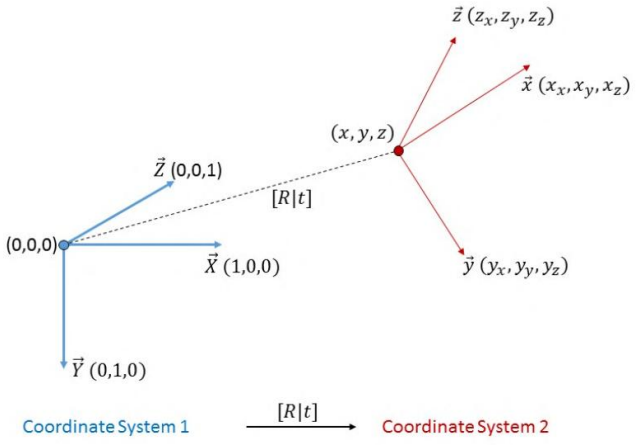


Figura 2.12: Esempio di trasformazione relativa di coordinate

Tra istanti di tempo consecutivi vi sarà una roto-traslazione del corpo. Esistono molti modi per definire matematicamente la trasformazione tra i due sistemi, ma la più comoda è mediante matrici di trasformazione, le quali contengono informazioni sulla traslazione e sulla rotazione rispetto ad un sistema di riferimento fisso. Esistono altri modi per definire la rotazione relativa tra sistemi di riferimento:

- Mendiante **angoli di Eulero**: essi sono formati da una terna di angoli che, ruotati in sequenza, trasformano un sistema in un altro. Gli angoli vengono definiti con terne che indicano l'ordine di rotazione degli assi (es. ZXZ, XYZ), la più intuitiva è la terna ZYX, molto utilizzata in quanto corrisponde a imbardata, beccheggio e rollio.

$$\begin{aligned} e^k_{k+1} &= (\theta_k, \phi_k, \psi_k) \qquad terna \quad X-Y-Z \\ e^k_{k+1} &= (\psi_k, \phi_k, \theta_k) \qquad terna \quad Z-Y-X \end{aligned} \tag{2.29}$$

- Mediante **matrici di rotazione**, le cui colonne rappresentano i versori degli assi del sistema ruotato in coordinate nel sistema di riferimento fisso. Le matrici di rotazione sono comode in quanto possono essere concatenate per ricavare la trasformazione rotazionale complessiva. Essa può essere ricavata

concatenando le matrici di rotazione associate ai corrispondenti angoli di Eulero come mostrato nell'equazione 2.30.

$$R = \begin{bmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{bmatrix} \tag{2.30}$$

- Mediante **quaternioni**: in certe situazioni risultano molto comodi perché non hanno problemi di singolarità in condizioni azimutali e sono facili da concatenare e confrontare.

$$q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k} \tag{2.31}$$

dove $(a, b, c, d) \in \mathbb{R}$, $\mathbf{i}, \mathbf{j}, \mathbf{k}$ sono le unità complesse. Esistono metodi per passare da quaternioni a matrici di rotazione e viceversa, e sono già implementati con specifiche funzioni in Matlab.

Ad ogni passo, si ricaveranno dai metodi di registrazione degli oggetti chiamati *rigidtform3d*, che contengono al loro interno le matrici di rotazione relativa e i vettori traslazione relativa sotto forma di matrice di trasformazione. Essi si possono scrivere in forma compatta nel seguente modo (equazione 2.32).

$$T_{k+1}^{k} = \begin{bmatrix} R_{k+1}^{k} & t_{k+1,k}^{k} \\ 0_{1x3} & 1 \end{bmatrix} \tag{2.32}$$

dove R_{k+1}^{k} è la matrice di rotazione relativa tra k e k+1, $t_{k+1,k}^{k}$ è il vettore traslazione relativa tra k e k+1 espresso nel sistema di riferimento k.
La loro comodità sta nel fatto che possono essere concatenati (equazione 2.33) per ottenere la matrice di trasformazione assoluta, di conseguenza passo dopo passo si concateneranno con la trasformazione assoluta precedente.

$$T_{k+1}^{W} = T_{2}^{1} T_{3}^{2} ... T_{k+1}^{k} \tag{2.33}$$

2.6 Pose Graph Optimization

L'ottimizzazione del grafico di posizione è un passaggio fondamentale per ridurre la deriva dovuta all'accumulo di incertezze durante il processo. Per quanto riguarda la LiDAR SLAM, bisogna innanzitutto suddividere la parte front-end

dell'ottimizzazione in cui avviene il rilevamento del loop e la parte matematica di ottimizzazione vera e propria del grafo in back-end.

2.6.1 Rilevamento del loop

Esistono innumerevoli modi per eseguire la *pose graph optimization*. In questa tesi si è scelto di utilizzare il metodo *Scan Context*, il quale è già implementato in Matlab e comunque fornisce ottimi risultati in termini di tempi computazionali.

Metodo Scan Context

Il metodo *ScanContext* è stato sviluppato da Kim et al. [20] e recentemente implementato in Matlab. Si fonda sull'estrazione e la correlazione di apposite *feature* per la chiusura del loop. Lo schema dell'algoritmo è riportato in figura 2.13.

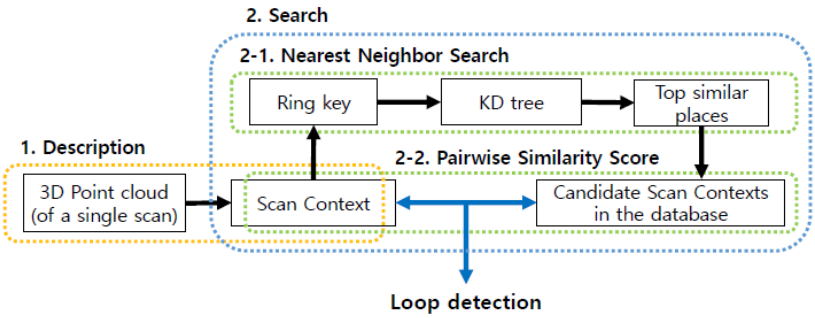


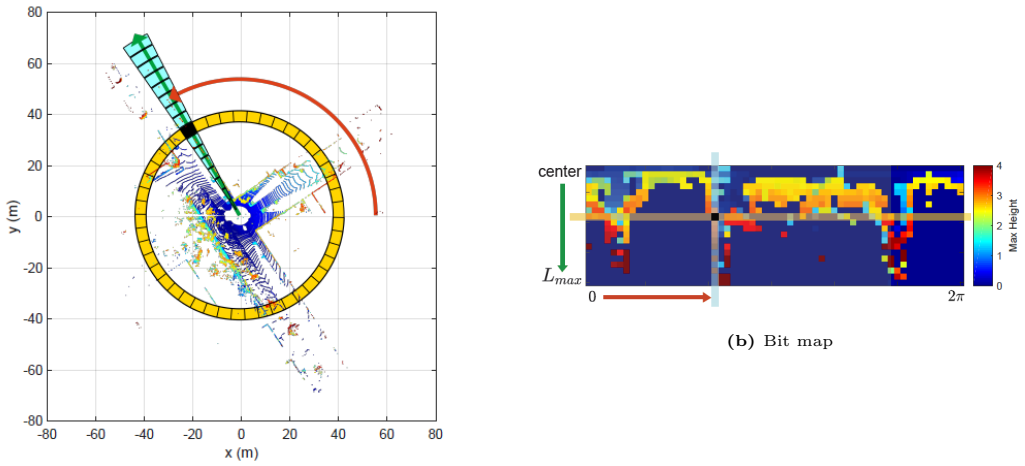
Figura 2.13: Schema dell'algoritmo *scanContext* [20]

Si possono distinguere due blocchi principali:

1. **Estrazione del descrittore**
   In questa fase viene eseguita una partizione della nuvola di punti in N_s settori angolari e N_r corone anulari (figura 2.14a). Viene preso come valore il massimo della coordinata verticale dei punti in essa contenuti per ogni cella (figura 2.14b). Nell'equazione 2.34 viene effettuata la partizione della nube, mentre nell'equazione 2.35 viene effettuata l'associazione dei valori.

$$P = \bigcup_{i \in N_r,\, j \in N_s} P_{i,j} \tag{2.34}$$

$$\Phi(P_{i,j}) = \max_{p \in P_{i,j}} z(p) \tag{2.35}$$



(a) Suddivisione in zone

(b) Bit map

Figura 2.14: Metodo *ScanContext* per la ricerca del loop [20]

In figura 2.14a è mostrata la griglia di partizione della scansione, mentre in figura 2.14b è raffigurata la corrispondente immagine estratta con l'equazione 2.35 e un determinato codice colore.

2. **Ricerca dei candidati**

   (a) *Pairwise Similarity Score*

   Dati due descrittori I^q e I^c la comparazione avviene confrontando i coseni delle distanze dei valori come nell'equazione 2.36.

$$d(I^q, I^c) = \frac{1}{N_s} \sum_{j=1}^{N_s} \left(1 - \frac{c_j^q c_j^c}{\|c_j^q\| \, \|c_j^c\|} \right) \tag{2.36}$$

   Inoltre, per avere un'indipendenza dalla rotazione dei candidati alla chiusura del loop, si calcolano le distanze delle colonne eseguendo uno *shift* di uno dei due descrittori di colonna in colonna, ed estraendo la distanza minima così trovata. Questo processo è oneroso, di conseguenza viene prima eseguito un filtraggio dei possibili candidati (punto 2.b).

$$D(I^q, I^c) = \min_{n \in N_s} d(I^q, I_n^c) \qquad n^* = argmin\{d(I^q, I_n^c)\} \tag{2.37}$$

   (b) *Nearest Neighbor Serch*

   Per una rapida esclusione di possibili candidati è necessario un descrittore che sia indipendente dalla rotazione. Di conseguenza viene

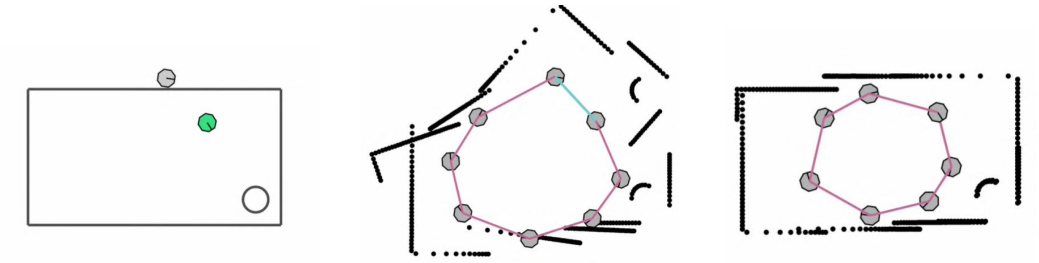
eseguita una funzione (detta *occupancy ratio*) su ogni anello in grado di ricavare un singolo valore.

$$\Psi(r_i) = \frac{\|r_i\|_0}{N_s} \quad i \in [1, N_s] \tag{2.38}$$

Si ottiene così un descrittore $K = [\Psi(r_1), ..., \Psi(r_{N_s})]$ indipendente dalla rotazione. K è utilizzato per la ricerca di corrispondenze. Trovati i candidati, in numero definito dall'utente, si andranno a eseguire le equazioni 2.36 e 2.37 per verificare nel dettaglio la corrispondenza.

2.6.2 Ottimizzazione nel back-end

Si esce dalla fase di front-end con un grafo (o un set come si vedrà in seguito) contenente nei nodi le posizioni assolute ottenute concatenando le varie trasformazioni relative, e i collegamenti tra i nodi che rappresentano la trasformazione relativa tra le varie posizioni. In esso è contenuta anche l'informazione di eventuali chiusure del *loop*, sotto forma di vincoli tra nodi del grafo.



(a) Ambiente scansionato **(b)** Senza pgo **(c)** Con pgo

Figura 2.15: Effetto dell'ottimizzazione sulla SLAM (Matlab techTalk)[21]

A grandi linee, gli algoritmi di ottimizzazione considerano ogni lato come un vincolo più o meno rigido, in base all'incertezza associata alla stima di quello specifico spostamento relativo. Il vincolo di chiusura del loop presenta una rigidezza molto grande, introducendo una sorta di stato tensionale nel grafo, il quale tenderà a riportare i nodi candidati alla chiusura molto vicini, e a spostare di conseguenza tutte le connessioni (figura 2.15). L'ottimizzazione si ferma all'equilibrio della struttura. Ottimizzando si ottengono tre vantaggi importanti:

- Una miglior stima della posizione attuale
- Una migliore stima di tutte le posizioni precedenti
- Una migliore ricostruzione della mappa

Matlab mette a disposizione due algoritmi per effettuare in modo automatico l'ottimizzazione del grafo:

- Algoritmo *builtin-trust-region* [22]
- Algoritmo *g2o-levenberg-marquardt* [23]

2.7 Metodi LIO: LiDAR Inertial Odometry

Come si è spiegato precedentemente, la SLAM basata solo su LiDAR presenta alcuni problemi:

- **Deriva** della traiettoria dovuta ad accumulo di errori
- **Sensibilità ai falsi positivi**: se l'algoritmo di ricostruzione fallisce la traiettoria devia completamente dalla *ground truth*, rendendo impossibile continuare la ricostruzione
- **Tempi di calcolo** elevati
- Sensibilità a **carenza di *feature***, soggetti in movimento e rumore ambientale.

Per tutte queste ragioni l'utilizzo di altri strumenti con cui fondere le informazioni è l'unica strada per irrobustire la LiDAR SLAM, renderla più precisa e veloce. Uno dei metodi basici, economici e leggeri per ottenere ciò è utilizzare una IMU. Alcuni degli utilizzi più comuni della fusione con IMU sono [2]:

- Dare una stima della traiettoria via pre-integrazione grazie alla maggiore frequenza di acquisizione
- Aiutare e velocizzare la convergenza
- Correggere la distorsione della scansione causata dal moto del sensore
- Correggere la registrazione nel caso di movimenti rapidi o fallimento della registrazione.

2.7.1 IMU: funzionamento

La IMU è uno strumento di misura inerziale in grado di misurare accelerazioni, velocità angolari e rotta rispetto al sistema a cui è vincolato. La tecnologia MEMS (*micro-electromechanical systems*) viene comunemente utilizzata per la produzione di sensori inerziali. I dispositivi MEMS sono solitamente di dimensioni molto

contenute e sfruttano le proprietà dei materiali semiconduttori (come il silicio) di integrare piccoli componenti con proprietà meccaniche sensibili a fenomeni inerziali e magnetici. Tipicamente un'unità IMU è formata da tre componenti [24]:

- **Giroscopio** per misurare la velocità angolare.
  Un giroscopio MEMS è formato da una struttura vibrante che, per determinare la velocità angolare, si basa sull'effetto Coriolis. La forza di Coriolis è una forza inerziale apparente, esercitata su un oggetto in rotazione che genera una deviazione apparente rispetto al sistema di riferimento in cui si trova. Conoscendo e misurando questo effetto si riesce a stimare la velocità angolare.
- **Accelerometro** per misurare l'accelerazione relativa.
  Gli accelerometri MEMS usano microstrutture sospese che permettono il moto in risposta ad un'accelerazione. Nel caso sia presente un'accelerazione, la massa comporta una risposta inerziale generando un output elettrico.
- **Magnetometro** per misurare il campo magnetico locale.
  Un magnetometro permette di calcolare la direzione del campo magnetico terrestre locale, permettendo così di stimare la rotta. Il sensore sfrutta l'effetto Hall, in cui un voltaggio proporzionale al campo magnetico viene generato su un conduttore percorso da corrente, immerso in un campo magnetico.

Data sheet

Matlab permette di simulare l'acquisizione di dati IMU. Per una corretta implementazione della IMU è necessario impostare i parametri di performance in modo corretto. Di seguito si riportano i parametri più importanti necessari a impostare l'acquisizione:

- **Measurement Range**: massimo valore misurabile
- **Resolution**: precisione dello strumento
- **ConstantBias**: valore letto dallo strumento a input nulli
- **AxesMisalignment**: errore di disallinemanto degli assi principali
- **NoiseDensity**: contenuto di rumore dell'output

Nelle simulazioni di questa tesi si è scelto di utilizzare la IMU ICM-20948 prodotta da TDK (*data sheet* [25]) già associata al LiDAR a disposizione. Essa è un dispositivo MEMS richiedente bassa potenza a 9 assi, adatto al tracciamento del moto. In tabella 2.3 sono riportate le specifiche presenti nel data sheet, e le stesse convertite nelle unità di Matlab.

Nome Data sheet	Valore	Unità	Nome Matlab	Valore	Unità
Accelerometro					
Full-scale range	2	g	MeasurementRange	19.62	m/s^2
Sensitivity Scale Factor	16384	LSB/g	Resolution	0.0005988	$(m/s^2)/LSB$
Initial Tolerance Board-level, all axes	50	mg	ConstantBias	0.4905	m/s^2
Cross-Axis Sensitivity	±2	%	AxesMisalignment	±2	%
Noise Spectral Density	230	$\mu g/\sqrt{Hz}$	NoiseDensity	0.002256	$(m/s^2)/\sqrt{Hz}$
Giroscopio					
Full-Scale Range	250	dps	MeasurementRange	4.3633	(rad/s)
Sensitivity Scale Factor	131	LSB/dps	Resolution	0.0001332	(rad/s)/LSB)
Initial ZRO Tolerance	±5	dps	ConstantBias	0.0872665	rad/s
Cross-Axis Sensitivity	2	%	AxesMisalignment	2	%
Noise Spectral Density	0.015	$dps/\sqrt{Hz}$	NoiseDensity	0.0002618	$(rad/s)/\sqrt{Hz}$

Tabella 2.3: Parametri utilizzati nella generazione di dati IMU

2.7.2 Algoritmi di fusione losely e thigtly coupled

Il modo in cui la IMU viene integrata nell'algoritmo SLAM è di fondamentale importanza. La distinzione cardine sta nel grado di accoppiamento con cui la IMU viene inserita nell'algoritmo. Per questo si possono distinguere due classi di algoritmi LIO:

- Algoritmi ***losely coupled*** (debolmente accoppiati) che solitamente utilizzano la pre-integrazione dell'IMU tra scan LiDAR successivi per aiutare l'algoritmo di ricostruzione dando una buona stima inziale della trasformazione
- Algoritmi ***thigtly coupled*** (fortemente accoppiati) i quali, non solo utilizzano la pre-integrazione dell'IMU, ma utilizzano le sue misurazioni unendo tramite filtri (come per esempio i filtri di Kalman) le misurazione IMU e LiDAR, dando quindi un peso proporzionalmente maggiore alla misurazione dell'IMU.

Sono stati sviluppati molti algoritmi che integrano la IMU nella LiDAR SLAM. Grazie agli articoli [26] e [2] in tabella 2.4 ne sono stati collezionati alcuni.

Metodo	Autore	Anno	Accoppiamento
LOAM [16]	Zhang et al.	2014	Debole
LeGO-LOAM [17]	Shan et al.	2018	Debole
RTAB-Map [27]	Labbè et al.	2018	Debole
Loam-Livox [28]	Lin et al.	2020	Debole
DLO [29]	Chen et al.	2022	Debole
LIPS [30]	Geneva et al.	2018	Forte
LIO-Mapping [18]	Ye et al.	2019	Forte
LIO-SAM [31]	Shan et al.	2020	Forte
LILO [32]	Zhang	2021	Forte
FAST-LIO [33]	Xu e Zhang	2020	Forte
FAST-LIO 2 [34]	Xu et al.	2021	Forte

Tabella 2.4: Elenco dei principali metodi debolmente e fortemente accoppiati

L'algoritmo LOAM utilizza le letture IMU per fornire una trasformazione di primo tentativo alla registrazione LOAM. Lego-LOAM procede nella stessa direzione di LOAM tramite una IMU integrata con il LiDAR, provvedendo anche alla compensazione del moto. RTAB-map procede in modo simile, dando una trasformazione di primo tentativo per evitare registrazioni sbagliate e velocizzare la convergenza. L'algoritmo DLO sviluppa un accoppiamento debole per migliorare l'accuratezza in condizioni rotazionali aggressive tramite l'utilizzo di consumer-grade IMU.

Per quanto riguarda gli algoritmi fortemente accoppiati, uno dei primi approcci è stato LIPS. Questo metodo crea una funzione di costo unica utilizzando i residui derivanti dal metodo di registrazione e i residui derivanti dalla pre-integrazione dell'IMU. Similmente LIO-Mapping ottimizza le misurazioni del LiDAR con quelle dell'IMU, utilizzando inoltre un modello *sliding-window* al fine di mantenere un certo grado di ottimizzazione. LIO-SAM, oltre all'utilizzo dell'IMU per la correzione della distorsione di acquisizione e la stima di una trasformazione di primo tentativo, ottimizza le misurazioni di LiDAR e IMU, crea un *factor-graph* dando un peso differente alle misurazioni di LiDAR e IMU e alle chiusure del loop. Il metodo presenta una marginalizzazione delle pose precedenti sfruttando il confronto con sotto-mappe locali invece che su tutta la mappa. LILO sfrutta le letture IMU per rimuovere la distorsione di acquisizione e dare una prima stima della trasformazione. Inoltre, crea una funzione di minimizzazione dei residui in modo simile a LIPS. Presenta prestazioni simili a FAST-LIO. FAST-LIO sfrutta una propagazione in avanti tramite integrazione dell'IMU per stimare la

posizione successiva e la matrice di covarianza, e una propagazione all'indietro per correggere gli errori di distorsione dell'acquisizione durante l'estrazione delle *feature*. Tramite la stima della trasformazione, sfrutta i filtri di Kalman iterativi per ottimizzare la stima della trasformazione.
L'algoritmo che si può considerare allo stato dell'arte, per quanto riguarda gli algoritmi LIO *thigtly coupled*, è indubbiamente FAST-LIO-2. Esso si basa sul predecessore FAST-LIO per quanto riguarda l'integrazione dell'IMU, rimuovendo però la parte di estrazione delle *feature* e utilizzando in toto la nube di punti in output dal LiDAR. Questo è possibile utilizzando un nuovo metodo di immagazzinamento e rimozione dei dati tramite un *ikd-tree*. Grazie al nuovo metodo di immagazzinamento sono possibili rimozione, downsampling e inserimento di punti nell'albero senza la sua intera ricostruzione. Inoltre, questa tipologia di immagazzinamento permette di velocizzare la ricerca KNN.

In letteratura sono sicuramente presenti molti altri metodi sia debolmente che fortemente accoppiati, ma si è scelto di fermarsi alla lista presentata in tabella 2.4.

Capitolo 3

Generazione dei dati in ambiente virtuale

3.1 Vantaggi di un ambiente virtuale

Il termine ambiente virtuale (in inglese *Virtual Environment*) indica quegli spazi tridimensionali navigabili ed interattivi che derivano dalla simulazione in tempo reale di un ambiente. L'ambiente virtuale è formato da un insieme di strumenti tecnologici che permettono interazione tra utente e sistema. Le altre parti di un sistema ambiente virtuale sono dispositivi di input, di output, computer, software che permettono la visualizzazione dei contenuti e l'interazione con essi.

I principali **vantaggi** di un ambiente virtuale sono:

- Possibilità di testare gli algoritmi implementati senza la necessita di affidarsi ad un processo di test reale
- Costo operativo basso rispetto a un test sul campo che necessità di comprare, tarare e implementare tutto l'apparato di test
- Rapidità di implementazione e facilità di integrazione con il software appena sviluppato
- Possibilità di eseguire moltissimi test se si presentano errori nel codice.

La scelte dell'ambiente virtuale in cui simulare dipende da molti fattori. Nel caso di questa tesi la necessità era di implementare un algoritmo LiDAR SLAM

applicabile ad un drone in un contesto urbano. Le **necessità** specifiche nel nostro caso erano quindi:

- Avere un software che si connettesse agevolmente a Matlab
- Avere un ambiente virtuale che rappresentasse un contesto urbano
- Poter inserire un drone in movimento in tale contesto
- Poter inserire uno strumento LiDAR che permettesse l'acquisizione dei dati in tale contesto
- Poter modificare le traiettorie percorse dal drone al fine di validare l'algoritmo con differenti casistiche.

Le risposte ai requisiti necessari alla simulazione si sono trovate nell'interconnessione permessa da Matlab con il software Unreal Engine (fornito da Epic Game), il quale permette la simulazione di ambienti inserendo la possibilità di generazione dati in tale ambiente.

Pacchetti Matlab necessari

I programmi utilizzati per generare ed elaborare i dati sono:

- **Matlab** versione R2023a
- **Simulink** versione 10.7
- **Unreal Engine** versione 4.27.2

L'interconnessione tra Matlab e Unreal Engine, e alcuni dei comandi che verranno sviluppati in seguito, possono avvenire solo tramite alcuni pacchetti:

- **UAV Toolbox** versione 1.5
- **UAV Toolbox Interface for Unreal Engine Projects** versione 23.1.0
- **Computer Vision Toolbox** versione 10.4
- **LiDAR toolbox** versione 2.3
- **Image Processing Toolbox** versione 11.7
- **Automated Driving Toolbox** versione 3.7
- **Navigation toolbox** versione 2.4

Hardware

Tutte le simulazioni sono state eseguite da un pc portatile le cui caratteristiche sono riportate in tabella 3.1.

Nome dispositivo	HP Pavilion Gaming Laptop
Processore	Intel® Core™ i7-10750H CPU @ 2.60GHz x 12 threads
RAM installata	16,0 GB (15,8 GB utilizzabile)
Tipo sistema	Sistema operativo a 64 bit, processore basato su x64
Scheda grafica	NVIDIA® GeForce® GTX 1650 Ti ver. 512.78
Disco rigido	512 GB PCIe® NVMe™ M.2 SSD

Tabella 3.1: Specifiche hardware utilizzato

É da specificare inoltre che, sebbene Matlab dia la possibilità di eseguire operazioni in parallelo, i codici implementati sono stati eseguiti con singolo processore. I tempi di calcolo potrebbero essere ridotti notevolmente se gli algoritmi di registrazione venissero eseguiti in parallelo.

Software Unreal Engine

É stato necessario scaricare il *launcher Epic Games* nel quale si è potuto poi scaricare Unreal Engiene, un software di modellazione di ambienti virtuali 3D. É bene specificare che Maltab 2023a non permette la connessione con la versione più recente di Unreal engine 5.2, di conseguenza si è scaricata la versione **Unreal Engine 4.27.2**.
Non si è dovuto generare un ambiente virtuale custom, in quanto il pacchetto Matlab *UAV Toolbox Interface for Unreal Engine Projects* prevedeva già l'ambiente *US City Block*, al cui interno sono presenti numerosi edifici e strade. Tramite una comoda funzione messa a disposizione da Matlab è anche possibile pianificare differenti traiettorie in questo ambiente. La creazione di ambienti custom avrebbe richiesto, oltre che una certa abilità nella creazione di ambienti tramite Unreal Engine, anche il download di ulteriori pacchetti che consentissero la connessione con tale ambiente custom.
In figura 3.1 è mostrata la vista dall'alto della mappa messa a disposizione nell'ambiente virtuale di default Matlab.

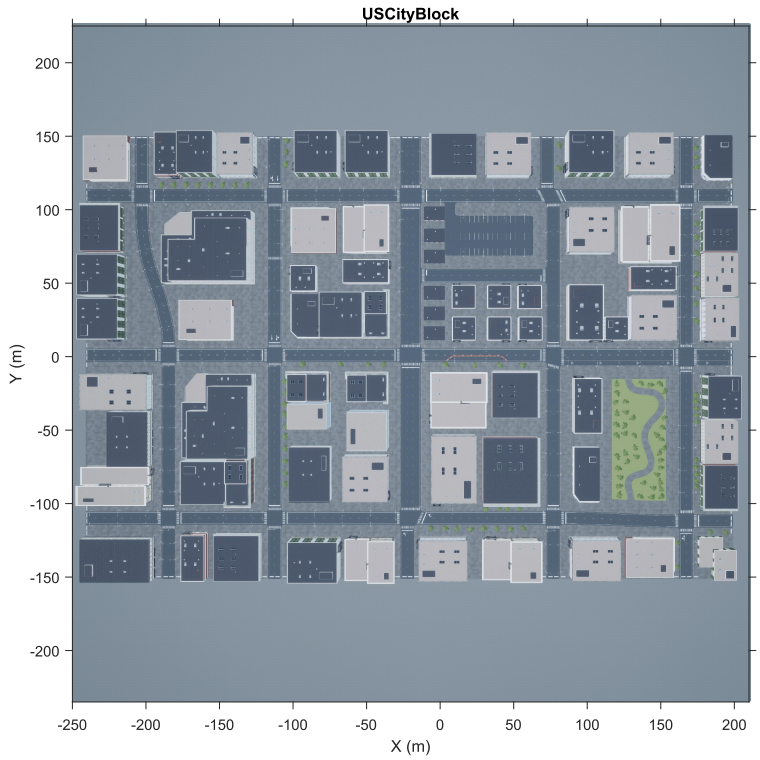


Figura 3.1: Vista dall'alto dell'ambiente virtuale US city block fornito da Matlab

3.2 Implementazione in Matlab-Simulink

L'interconnessione tra i software avviene tramite lo sviluppo di un semplice programma Simulink e uno script di input in Matlab. Di seguito è riportato il programma Simulink sviluppato.

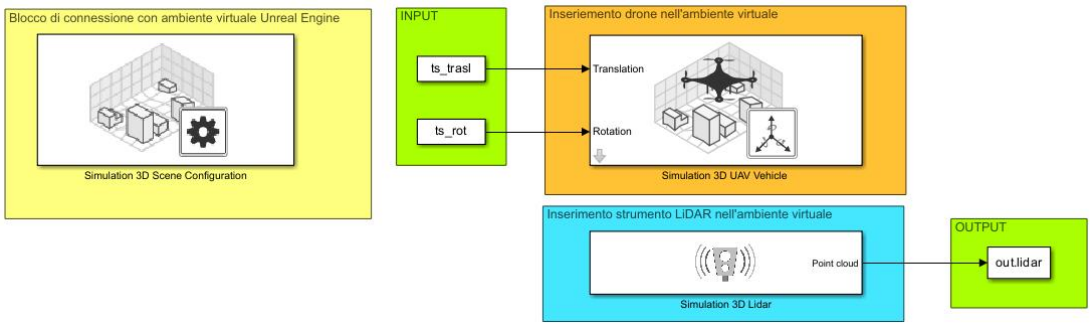


Figura 3.2: Schema Simulink per la generazione ambiente virtuale

Si possono distinguere vari blocchi:

- **Simulation 3D Scene configuration**
  Consente di connettere l'ambiente virtuale, sia default che custom, con simulink. Bisogna definire:

  - La sorgente da cui deriva la scena virtuale, nel nostro caso *Default Scene*
  - Il nome dell'ambiente virtuale, nel nostro caso *US city block*
  - Con quale veicolo è associata la vista della simulazione, nel nostro caso *SimulinkVehicle1*
  - Il *sample rate* della simulazione, impostato a 1/60 s

  In questo blocco si possono definire altri parametri, tra cui condizioni meteo, illuminazione e eventuali altri output.

- **Simulation 3D UAV Vehicle**
  Questo blocco definisce l'attore della scena, nel nostro caso il quadrirotore. Si possono impostare i parametri:
  - Tipologia, nome e colore dell'UAV
  - Valori di traslazione e rotazione iniziali (x_0 e r_0)

- **Simulation 3D LiDAR**
  Questo blocco inserisce nella scena corrente uno strumento LiDAR, permettendo l'acquisizione di dati. Si possono definire:
  - Numero che identifica il sensore
  - Veicolo a cui è associato
  - Posizione di montaggio (nel nostro caso coordinate nulle implicano la coincidenza con il centro del sistema di riferimento dell'UAV)
  - Tutti i parametri specifici del sensore LiDAR, già illustrati in tabella 2.1, e qui brevemente riportati.

Detection Range	100 m
Range resolution	0.01 m
Vertical FOV	45°
Vertical resolution	45/32 °
Horizontal FOV	360°
Horizontal resolution	360/1024 °
Sample rate	10 Hz

Tabella 3.2: Setting Simulation 3D LiDAR block

- **INPUT e OUTPUT**

– **INPUT**: vengono forniti i vettori traslazione e rotazione (angoli di Eulero ZYX) rispetto al sistema di riferimento globale in formato *timeseries*

– **OUTPUT**: si avranno K matrici 3D di dimensione $m \times n \times 3$, dove K è il numero di scansioni, m è il numero di canali verticali, n è il numero di canali orizzontali e nell'ultima dimensione saranno contenute le coordinate dei punti rispetto al sistema di riferimento locale.

3.3 Generazione delle traiettorie studiate

Per la generazione e la visualizzazione delle traiettorie, le quali rappresenteranno gli INPUT di Simulink, si è fatto affidamento su una funzione *helper* di Matlab. Di seguito è riportato lo script Matlab utilizzato per la generazione delle traiettorie (ricavato da sito Matlab [35])

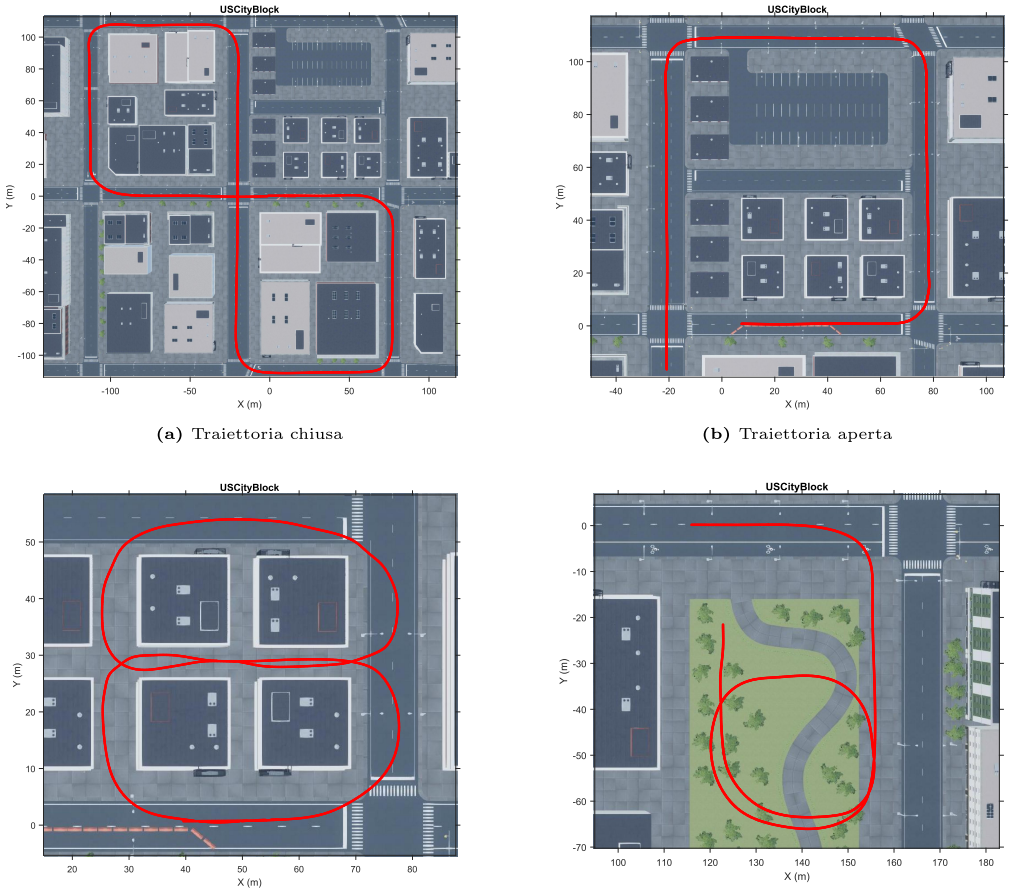
```
%% Generazione assisttia di traiettoria
sceneName = 'USCityBlock';
[sceneImage, sceneRef] = helperGetSceneImage(sceneName);
% Selezioni punti interattivo
hFig = helperSelectSceneWaypoints(sceneImage, sceneRef);

%% Fase di smoothness
numPoses = size(refPoses{1}, 1);
refDirections  = ones(numPoses,1);
numSmoothPoses = 150 * numPoses;
[smoothRefPoses,~,cumLengths] = smoothPathSpline(refPoses{1},refDirections,
    numSmoothPoses);

%% Creazione vettori input simulink
space = cumLengths(end);
v0 = 10; % modulo velocit  iniziale
h = 10; % quota
simStopTime = space/v0;     % tempo di simulazione
set_param(gcs, 'StopTime', num2str(simStopTime));
% Creazione di un profilo di velocit  costante.
timeVector = normalize(cumLengths, 'range', [0, simStopTime]);

% Creazione delle variabili richieste da Simulink
refPosesX = smoothRefPoses(:,1);
refPosesY = smoothRefPoses(:,2);
refPosesZ = h*ones(numSmoothPoses,1);
refRotYaw = deg2rad(smoothRefPoses(:,3));
refRotPitch = deg2rad(zeros(numSmoothPoses,1));
refRotRoll = deg2rad(zeros(numSmoothPoses,1));
% Vettori
t_vec = [refPosesX, refPosesY, refPosesZ];
r_vec = [refRotYaw,refRotPitch,refRotRoll];
% Valori iniziali
```

```
x0 = t_vec(1,:);
r0 = r_vec(1,:);
% Timeseries
ts_trasl = timeseries(t_vec,timeVector);
ts_rot = timeseries(r_vec,timeVector);

%% Modello simulink con unreal engine
open_system(modelName); snapnow;
sim('UAV_UnrealEngine_Lidar');

%% Salvataggio
% Generazione del vettore cella degli oggetti pointClouds
lidar = out.lidar;
k = length(lidar(1,1,1,:));
pClouds = cell(1,k);
for i = 1:1:k
     pClouds{i} = pointCloud(lidar(:,:,:,i));
end
% Struttura di raccolta dati
letture_LIDAR.time_vec =  time_vector;
letture_LIDAR.t_vec =  t_vector;
letture_LIDAR.r_vec =  r_vector;
letture_LIDAR.pClouds = pClouds;
save letture_LIDAR.m letture_LIDAR
```

Listing 3.1: Generazione traiettorie di input

La funzione `helperSelectSceneWaypoints` apre una schermata in cui si possono selezionare i punti della traiettoria cliccando sulla mappa. Una volta generati i punti la funzione `smoothPathSpline`, per mezzo di interpolazioni con spline, rende più fitta e liscia la traiettoria. Si vanno quindi a generare i vettori tempo, traslazione e rotazione e infine a creare le `timeseries` da dare in input al programma Simulink. Terminata la simulazione si esegue il salvataggio in una struttura che contenga il vettore tempo, la traslazione e la rotazione come riferimento per la *ground truth*, e che contenga le scansioni generate.
Sono state generare varie traiettorie tra cui:

- Traiettoria chiusa con media velocità
- Traiettoria aperta con media velocità
- Traiettoria chiusa a 2 loop con bassa velocità (otto)
- Traiettoria aperta con variazione di quota lineare con bassa velocità



(a) Traiettoria chiusa

(b) Traiettoria aperta

(c) Traiettoria 2 loop (otto)

(d) Traiettoria variazione quota

Figura 3.3: Traiettorie generate

3.4 Estrazione dei dati LiDAR

I dati LiDAR in output vengono forniti sotto forma di matrici 4D (concatenazione di matrici 3D). Per comodità si è scelto di trasformare queste matrici in oggetti *point cloud* e raccoglierli in vettori di tipo *cell*. In particolare l'oggetto *point cloud* è così formato:

- **Location:** è costituita dalla matrice 3D di dimensioni $m \times n \times 3$ estratta ad ogni scansione del LiDAR
- **Count:** numero di punti scansionati
- **XLimits:** rappresenta il valore massimo e minimo scansionato per la coordinata x
- **YLimits:** rappresenta il valore massimo e minimo scansionato per la coordinata y

- **ZLimits:** rappresenta il valore massimo e minimo scansionato per la coordinata z
- **Color, Normal, Intensity:** contengono altre caratteristiche che l'oggetto *point cloud* può avere ma che non sono di nostro interesse.

In figura 3.4 è raffigurata una nuvola di punti estratta da LiDAR. Si può notare come la visualizzazione associ direttamente un codice colore associato alla coordinata z di scansione.

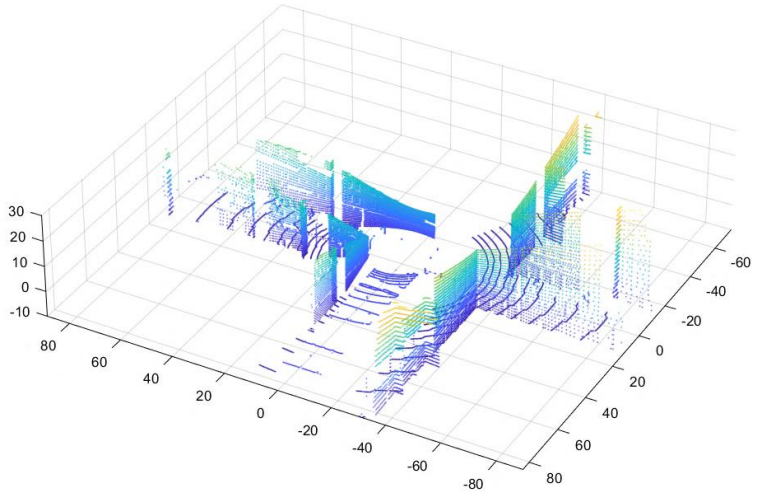


Figura 3.4: Nube di punti estratta dall'ambiente virtuale

Tutte le nuvole di punti relative ad ogni scansione verranno comodamente raccolte in oggetti cella, in cui ogni cella contiene l'oggetto *point cloud* associato a quella scansione. Si potranno quindi usare i dati così generati per le successive validazioni del codice, in quanto la generazione dei dati è iniziata inserendo nel blocco Simulation 3D UAV Vehicle la traiettoria di *ground truth* sotto forma di rotazione e traslazioni.

Visualizzazione della simulazione

Eseguendo il codice Simulink si apre in automatico la connessione con Unreal Engine, mediante una finestra di visualizzazione che mostra il nostro drone in moto nell'ambiente virtuale. Si possono inserire altre telecamere che permettano la visualizzazione da altre angolazioni. In figura 3.5 è mostrata la visualizzazione di default, creata da una telecamera che insegue la scena ad una data distanza e angolazione.

Figura 3.5: Screenshot del punto di vista defoult della simulazione

In figura 3.6 è mostrata invece una raffigurazione del drone con una angolazione differente.

Figura 3.6: Vista drone con angolazione differente

É possibile inoltre mostrare le nuvole di punti scansionate in tempo reale, tuttavia questo rallenta la simulazione e non è di particolare interesse.

3.5 Generazione dati IMU in Simulink

Per sviluppare ulteriormente il codice, e permettere l'integrazione con un sistema inerziale è stato necessario simulare anche la generazione dei dati IMU. Unreal Engine non permette la generazione diretta di dati IMU durante la simulazione virtuale, quindi ci si è dovuti appoggiare ad una opportuna funzione Matlab esterna. La funzione è l'oggetto `imuSensor`, il quale presenta la possibilità si

inserire i parametri di uno strumento IMU e di simulare la presa dati dando in input accelerazioni nel sistema locale e velocità angolari.

```
%% Prova di generazione dati IMU
% Caricamento dati LiDAR
load(folder_LiDAR+'letture_LIDAR_close.mat');
t_vec = letture_LIDAR.t_vec; r_vec = letture_LIDAR.r_vec;
time_vec = letture_LIDAR.time_vec;
% Equispaziamento vettore tempo per integrazione
simStopTime = time_vec(end);
timeVector = linspace(0, simStopTime,length(t_vec(:,1)))';
deltat = timeVector(10)-timeVector(9);

%% Calcolo dei vettori accelerazione e velocit  angolare
% Velocit  angolare
quat = eul2quat(r_vec,"ZYX");
angVel = angvel(quaternion(quat),deltat,'frame');
% Derivate della traiettoria rispetto al sistema globale
vel = diff(t_vec)/deltat;
acc = diff(vel)/deltat;
% Rotazione per riportare nel sistema corpo
acc_body = zeros(length(acc),3);
for i = 1:1:length(acc)
    R = eul2rotm(r_vec(i,:), 'ZYX');
    acc_body(i,:) = R'*acc(i,:)';
end

%% Generazione dati IMU
IMU = imuSensor('accel-gyro','ReferenceFrame','ENU');

%Setting dello strumento
IMU.Accelerometer = accelparams( ...
    'MeasurementRange',19.62, ...
    'Resolution',0.000598754, ...
    'ConstantBias',0.4905, ...
    'AxesMisalignment',2, ...
    'NoiseDensity',0.0022563);
IMU.Gyroscope = gyroparams( ...
    'MeasurementRange',4.36332313, ...
    'Resolution',0.000133231, ...
    'AxesMisalignment',2, ...
    'NoiseDensity',0.000261799, ...
    'ConstantBias',0.087266462);

% Escludo gli ultimi due valori causa metodi di differenzazione diversi
[accReadings,gyroReadings] = IMU(acc_body,angVel(1:end-2,:));

%% Salvataggio
letture_IMU.acc  = accReadings;
letture_IMU.gyro = gyroReadings;
letture_IMU.bias_g = IMU.Gyroscope.ConstantBias;
letture_IMU.bias_a = IMU.Accelerometer.ConstantBias;
letture_IMU.tempo = timeVector;
save letture_IMU_h letture_IMU
```

Listing 3.2: Generazione dati IMU

Nel codice si possono notare tre blocchi chiave.

1. Il primo è la derivazione della traiettoria al fine di ottenere l'accelerazione e la velocità angolare. Nel caso dell'accelerazione si è ottenuta la derivata numerica utilizzando le differenze finite `diff()/deltat`. Nel caso della velocità angolare, siccome il programma di generazione delle traiettorie aveva generato angoli compresi tra $[0, 2\pi]$, il profilo di angoli manifestava delle discontinuità nel caso di passaggio tra primo e quarto quadrante, di conseguenza la tecnica delle differenze finite presentava dei picchi impulsivi incontrollati. Matlab ha implementata una funzione per ricavare la velocità angolare che utilizza i quaternioni `angVel(quaternion(quat),deltat,'frame')`

2. Il secondo punto chiave è la trasformazione delle derivate della accelerazione dal sistema di riferimento globale al sistema di riferimento locale (corpo). Questo perché le letture dell'unità IMU vengono fatte nel sistema corpo, mentre le derivate della traiettoria saranno nel sistema di riferimento globale.

3. Il terzo blocco importante è la generazione delle letture IMU a partire dalle derivate appena calcolate. In seguito alla creazione dell'oggetto `imuSensor`, si associano all'oggetto le proprietà `IMU.Accelerometer.accelparams` per l'accelerometro e `IMU.Gyroscope.gyroparams` per il giroscopio. Si potrebbe anche simulare l'acquisizione di un magnetometro. I parametri dello strumento sono stati descritti nella sezione relativa all'IMU. Una volta assegnati i paramatri basta applicare la funzione che richiama l'oggetto creato per generare le letture IMU.

Nelle figure 3.7 e 3.8 vengono riportati i valori ottenuti per la traiettoria 3.3a. In particolare nella figura 3.7 sono riportati, suddivisi per coordinate, spazio, velocità e accelerazione nel sistema globale. Tramite una rotazione si è poi ricavata l'accelerazione nel sistema corpo.

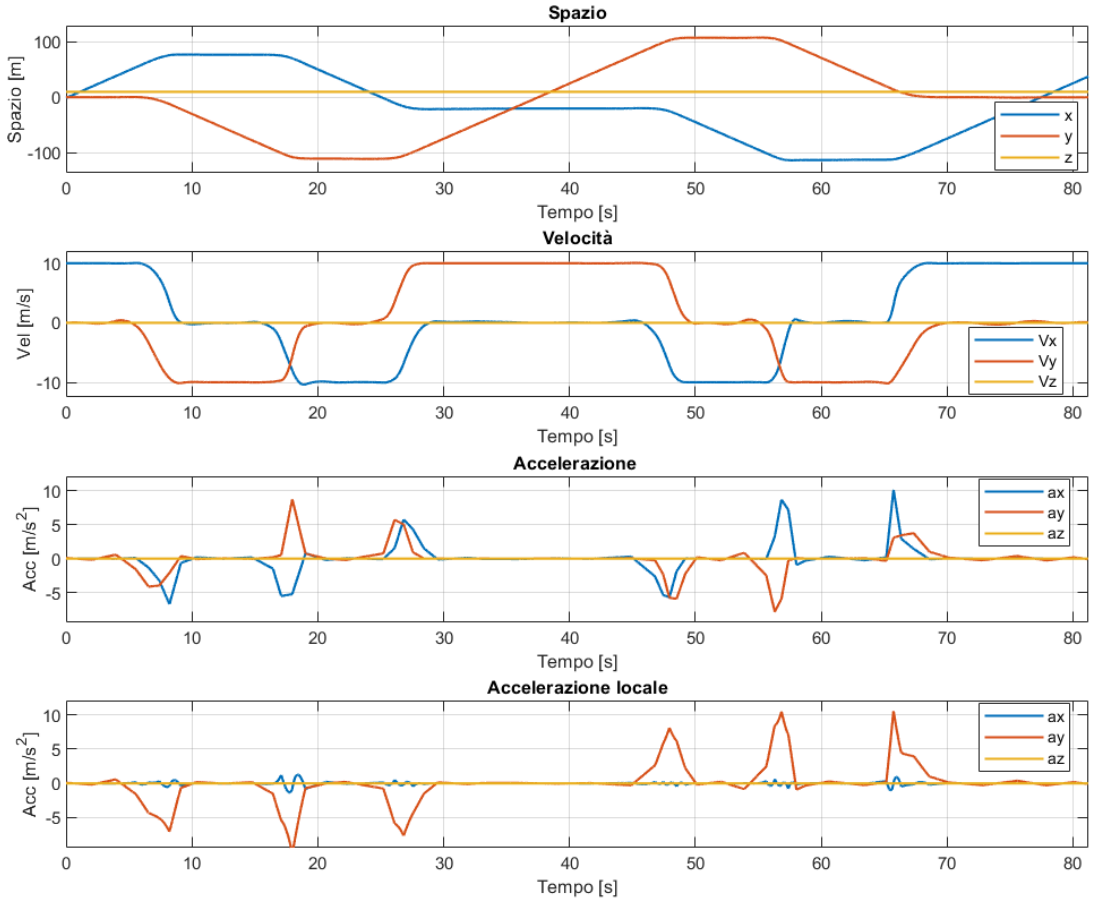


Figura 3.7: Spazio, velocità e accelerazione della traiettoria in figura 3.3a

Nella figura 3.8 sono riportati invece gli angoli e la velocità angolare. Si può notare come anche la funzione `angVel` presenti dei problemi nelle discontinuità, che sono comunque contenuti rispetto alla derivazione normale.

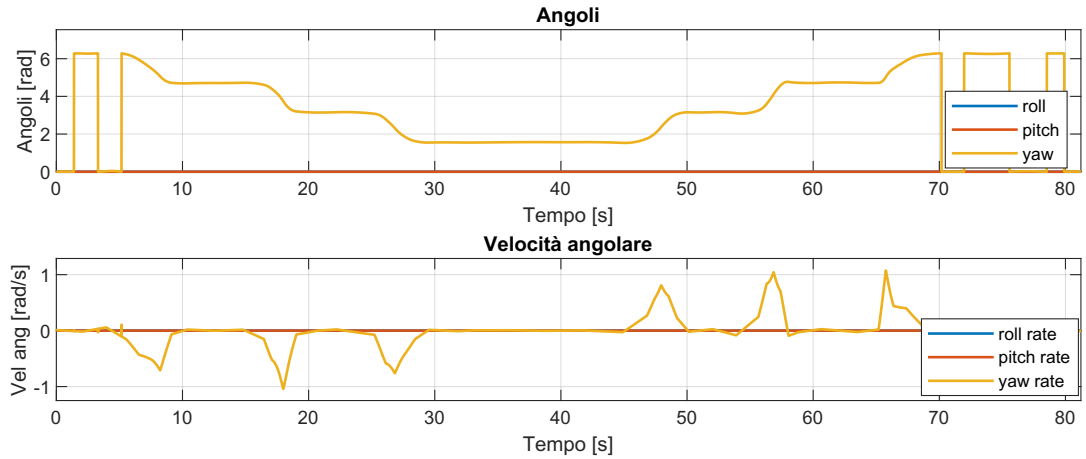


Figura 3.8: Angoli e velocità angolare della traiettoria in figura 3.3a

All'esecuzione del comando di generazione dati si ottengono gli output dell'accelerometro mostrati in figura 3.10 e l'output del giroscopio mostrato in figura 3.9.
É importante notare che le misurazioni presentano sia *bias* che rumore casuale

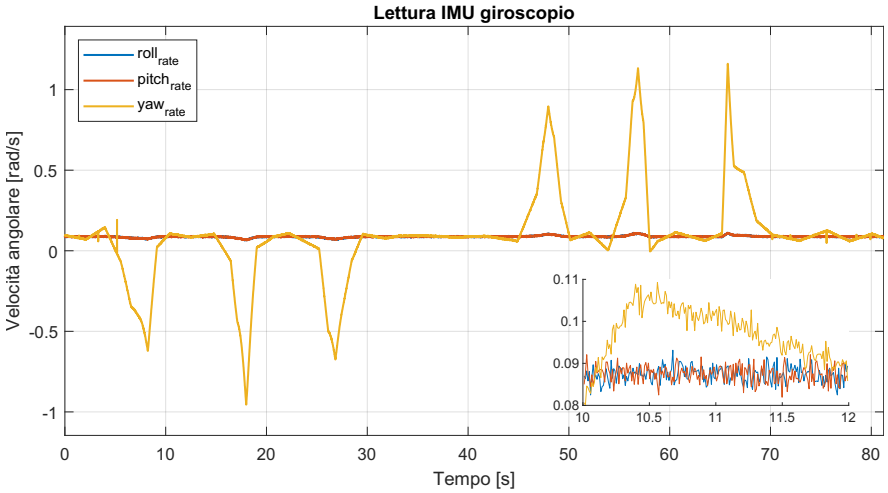


Figura 3.9: Letture IMU giroscopio

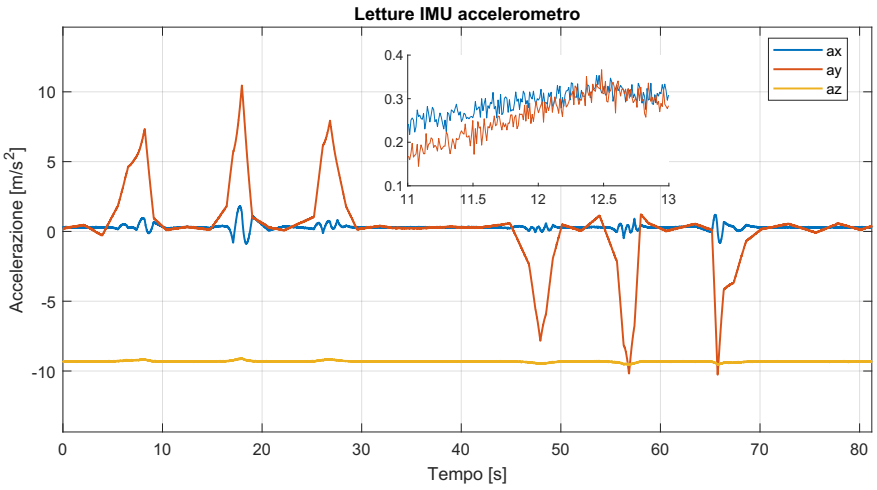


Figura 3.10: Letture IMU accelerometro

(come mostrato nei riquadri di ingrandimento). Inoltre, le letture dell'accelerometro presentano il vettore gravità verso il basso perché nell'oggetto `imuSensor` è stato scelto il sistema di riferimento *ENU*. Avendo simulato rollio e imbardata nulli, la gravità è lungo l'asse z del sistema corpo. Se si fosse simulata una generica rotazione, prima di rimuovere la gravità durante la pre-integrazione si sarebbe dovuto esprimere il vettore gravità nel sistema corpo tramite le misurazioni angolari. Le altre traiettorie hanno un andamento analogo e non verranno riportate.

Capitolo 4

Implementazione in Matlab

Terminata la generazione dei dati sono stati sviluppati i programmi LiDAR SLAM. Dapprima si è utilizzata la ricostruzione ICP, successivamente la ricostruzione LOAM, introducendo per quest'ultima la ricostruzione LOAM semplice e il suo affinamento tramite LOAM map.

4.1 Implementazione metodo ICP

Il metodo ICP illustrato in precedenza viene di seguito implementato.

4.1.1 Front-end

In seguito al caricamento dei dati generati nell'ambiente virtuale, si inizia definendo alcuni parametri utili.

```
%% Caricamento dati
t_vec = letture_LIDAR.t_vec;
r_vec = letture_LIDAR.r_vec;
pClouds = letture_LIDAR.pClouds;

%% Parametri
% # frame estratti LiDAR
numFrames   = length(pClouds);
% # frame da saltare
skipFrames  = 4;
% downsalmpling
downsamplePercent = 0.3;
% R minimo rimozione punti
Rmin = 2;
```

```
% R massimo rimozione punti
Rmax = 70;
% inlier accettati nel pcregistericp
inlierRatio = 0.2;
% Massimo RMSE  candidato loop
maxTolerableRMSE  = 0.8;
% R di ricerca candidati
Rloop = 8;
% massime Iter chiusura del loop
MaxIter = 50;
```

Listing 4.1: Parametri ICP

Si decide ogni quanto estrarre un frame chiave tramite il parametro `skipFrames`, si definiscono la percentuale di *downsampling* con metodo randomico da effettuare nel pre-processo con `downsamplePercent` e i raggi di esclusione delle letture troppo vicine o troppo lontane (`Rmin` e `Rmax`). Un parametro importante da definire è la percentuale di *inlier* da considerare valida nel metodo di registrazione ICP. In questo caso il 20 % degli *inlier* verrà considerato valido. Infine, si definiscono i parametri di chiusura del loop, in particolare l'errore di *rms* massimo tollerabile tra due candidati `maxTolerableRMSE`, il raggio di ricerca `Rloop` e le massime iterazioni del metodo di registrazione per calcolare la trasformazione di chiusura del loop. Si prosegue quindi inizializzando alcune variabili.

```
%% Inizzializzazioni
% Imposto un random seed per riproducibilit
rng(0);
% Inizzializzo il set di raccolta
vSet = pcviewset;
% Inizzializzo il set di raccolta info loop
loopDetector = scanContextLoopDetector;
% Trasformazione assoluta
absTform   = rigidtform3d;
% Trasformazione relativa
tform   = rigidtform3d;
% Contatore delle view
viewId = 1;
```

Listing 4.2: Inizializzazione ICP

Per prima cosa si imposta il *random seed* a zero per garantire la riproducibilità durante il *downsampling*. Si inizializza quindi in `pcviewset`, l'oggetto di raccolta di tutte le informazioni ricavate durante l'esecuzione. In modo analogo si inizializza l'oggetto di raccolta delle informazioni riguardanti l'investigazione della chiusura del loop `scanContextLoopDetector`. Si inizializzano quindi la trasformazione assoluta e la trasformazione relativa, generando due oggetti `rigidtform3d`. Infine

si inizializza il conteggio delle *view*.

```
for n = 1 : skipFrames : numFrames

    % Lettura e preprocesso nube di punti
    ptCloud = pClouds{n};
    % Rimozione punti non necessari
    selectedIdx = findPointsInCylinder(ptCloud,[Rmin Rmax]);
    ptCloud_EC = select(ptCloud,selectedIdx,OutputSize="full");
    % Downsampling
    ptCloud_DS = pcdownsample(ptCloud_EC,"random",downsamplePercent);

    if n == 1
        % Aggiunnta prima vista al vset
        vSet =addView(vSet,viewId,absTform,"PointCloud",ptCloud);
        % Estrazione primo descrittore per scancontext
        descriptor = scanContextDescriptor(ptCloud);
        % Aggiunta del descrittore al loop detector
        addDescriptor(loopDetector, viewId, descriptor)
        viewId = viewId + 1;
        ptCloudPrev = ptCloud;
        tform = absTform ;
        continue;
    end
```

Listing 4.3: Pre-processo ICP

Si inizia il ciclo iterativo, estraendo di volta in volta le nuvole di punti corrispondenti. Si esegue quindi il pre-processo, estraendo (`findPointsInCylinder`) e rimuovendo (selezionando solo quelli validi con `select`) i punti troppo lontani e i punti troppo vicini da non essere utilizzabili. Si passa quindi al *downsampling* della nuvola di punti con metodo randomico. Se il frame letto è il primo, si passa direttamente alla memorizzazione. La memorizzazione avviene tramite il `pcviewset` precedentemente generato tramite il comando `addView`. Prima di procedere con la nuvola di punti successiva, si estraggono i descrittori per la ricerca del loop. In particolare tramite `scanContextDescriptor` si estraggono i descrittori e tramite `addDescriptor` si memorizzano nel set di raccolta le informazioni del loop.

```
% Registrazione: calcolo della trasformazione relativa
tform = pcregistericp(ptCloud_DS, ptCloudPrev, 'Metric', 'planeToPlane', ...
    'InlierRatio', inlierRatio,'InitialTransform',tform);
% Aggiornamento posizione assoluta con concatenazione
absTform = rigidtform3d( absTform.A * tform.A );
% Aggiornamento dei vset con aggiunta connessione
vSet = addView(vSet,viewId,absTform,"PointCloud",ptCloud);
```

```
vSet = addConnection(vSet, viewId-1, viewId, tform);
```

Listing 4.4: Registrazione ICP

In queste righe si presenta la parte di codice più importante, in quanto avviene la registrazione che permette di ricavare la trasformazione relativa tra due *scan* accettati consecutivi. Il comando `pcregistericp` permette di ricavare la trasformazione relativa tra le due nuvole di punti. Si è scelto di utilizzare il metodo `planeToPlane` perché si è visto essere migliore rispetto a metodi come `pointToPoint` o `pointToPlane`. La percentuale di *inlier* accettata viene qui inserita e viene definita una trasformazione di primo tentativo per le iterazioni del metodo, utilizzando la trasformazione ricavata precedentemente. Si sta quindi facendo un'ipotesi di moto costante, che è restrittiva, ma sufficiente per un primo tentativo. Questa è la fase cruciale in quanto una trasformazione non buona causa un aumento dei tempi computazionali e, nei casi più gravi, divergenza del metodo. Una volta ricavata la trasformazione si procede concatenandola con la trasformazione assoluta ricavata nell'iterazione precedente, in modo da localizzare la posizione attuale. Si procede quindi salvando nei *vSet* la posizione attuale con `addView` e la posizione relativa con `addConnection`.

```
% Estrazione e aggiornamento descrittori scanContext
descriptor = scanContextDescriptor(ptCloud);
addDescriptor(loopDetector, viewId, descriptor)
% Ricerca candidati, variabile di igresso nella parte successiva
loopViewId = detectLoop(loopDetector, 'SearchRadius', Rloop);

if ~isempty(loopViewId)
    loopViewId = loopViewId(1);   %Candidato con rms minore

    % Estrazione del candidato dal vSet
    loopView = findView(vSet, loopViewId);
    ptCloud_loop = loopView.PointCloud;

    % Ri esecuzione del preprocesso
    selectedIdx = findPointsInCylinder(ptCloud_loop,[Rmin Rmax]);
    ptCloud_EC_loop = select(ptCloud_loop,selectedIdx);
    ptCloud_loop = pcdownsample(ptCloud_EC_loop,"random", downsamplePercent);

    % Stima posizione relativa nube attuale con la nube candidata
    [tform_loop,~, rmse] = pcregistericp(ptCloud_DS,ptCloud_loop,'Metric','
    planeToPlane','InlierRatio',inlierRatio,"MaxIterations",MaxIter);
    % verifica rms limite
    acceptLoopClosure = rmse <= maxTolerableRMSE;

    if acceptLoopClosure
    % InfoMat, per semplicita' matrice identita'
    infoMat = eye(6);
```

```
27      % Aggiunta della connessione di chiusura nel set
28      vSet=addConnection(vSet,loopViewId,viewId,tform_loop,infoMat);
29      end
30  end
31  % Aggiornamento variabili
32  viewId = viewId + 1;
33  ptCloudPrev = ptCloud_DS;
34  end
```

Listing 4.5: Chiusura del loop

Si entra quindi nell'ultima parte del blocco front-end, nel quale avviene la ricerca di candidati alla chiusura del loop. In seguito all'estrazione e memorizzazione dei descrittori *ScanContext* si utilizza il comando `detectLoop` per cercare candidati nel raggio di ricerca definito. Se la ricerca ha esito positivo si entra nelle righe successive. Si estrae il candidato con *rms* minore e si estrae la nuvola di punti associata tramite il *vSet*. Avendo salvato le nuvole di punti non pre-processate per ottenere una mappa migliore, si deve rieseguire il pre-processo. Si esegue quindi la registrazione tra la nube attuale e la nube candidata alla chiusura del loop, estraendo anche il valore di *rms*. Si esegue quindi un controllo sull'*rms*, confrontandolo con `maxTolerableRMSE` definito nei parametri iniziali. Se il confronto è positivo si passa alla memorizzazione della chiusura del loop nel *vSet*.
Si termina memorizzando la nuvola di punti per l'iterazione successiva e incrementando la `viewId`.

4.1.2 Back-end

Concluse le misurazioni LiDAR si entra nel back-end. Per prima cosa si esegue l'ottimizzazione del grafico di posizione con le informazione ricavate dalla chiusura del loop.

```
1  %% Creazione e ottimizzazione del grafo
2  G = createPoseGraph(vSet);
3  % Ottimizzazione del grafo
4  optimG = optimizePoseGraph(G, "g2o-levenberg-marquardt");
5  % Aggiornamento del vset con le posizioni ottimizzate
6  vSetOptim = updateView(vSet, optimG.Nodes);
```

Listing 4.6: Pose Graph Optimization ICP

Esiste il comodo comando `createPoseGraph` per la generazione di un grafo a partire dal *vSet* memorizzato. Si esegue quindi l'ottimizzazione del grafo con `optimizePoseGraph` tramite l'algoritmo `g2o-levenberg-marquardt`. Si esegue

l'*update* del *vSet* con le nuove informazioni del grafico ottimizzato. Verranno quindi aggiornati non solo i nodi corrispondenti alla chiusura del loop, ma tutte le posizioni registrate. Ciò che è riportato nel codice 4.7 ha solo lo scopo di visualizzazione e valutazione della bontà della ricostruzione.

```
%% Creazione della mappa
mapGridSize = 0.2;
ptClouds = vSetOptim.Views.PointCloud;
absPoses = vSetOptim.Views.AbsolutePose;
% Allineamento creazione della mappa
ptCloudMap = pcalign(ptClouds, absPoses, mapGridSize);
pcshow(ptCloudMap); hold on
plot(vSetOptim);

%% Confronto traiettorie
Traj_pgo = zeros(viewId-1,3);   % pose graph optimization
Traj_nlc = zeros(viewId-1,3);    % no loop closer
for i = 1:1:viewId-1
% Estrazione traiettoria ottimizzata
Traj_pgo(i,1:3)=optimG.Nodes.AbsolutePose(i,1).Translation+t_vec(1,:);
% Estrae la traiettoria NON ottimizzata
Traj_nlc(i,1:3)=G.Nodes.AbsolutePose(i,1).Translation+t_vec(1,:);
end
%  Plot traiettorie a confronto
plot3(Traj_nlc(:,1),Traj_nlc(:,2),Traj_nlc(:,3)); hold on
plot3(Traj_pgo(:,1),Traj_pgo(:,2),Traj_pgo(:,3)); hold on
plot3(t_vec(:,1),t_vec(:,2),t_vec(:,3)); hold off

%% Errori di root mean square
a = t_vec; b = Traj_nlc; c = Traj_pgo;
Na = length(a); Nb = length(b);
s = (Na-1)/(Nb-1);
a_mod = zeros(Nb,3);
% ricampionamento ground truth per avre stessa dimensione
for i = 1:1:Nb
    if i == 1
        a_mod(1,:) = a(1,:);
    else
        k = round(s*(i-1)+1);
        a_mod(i,:) = a(k,:);
    end
end
ERRORE_before = rmse(a_mod,b, 'all');
ERRORE_after = rmse(a_mod,c,'all');
```

Listing 4.7: Post-processo ICP

Il comando più importante è `pcalign`, con il quale si allineano e fondono le nuvole di punti del *vSet*, dando come informazione le posizioni assolute ricavate. Viene inoltre definito un `mapGridSize`, che non è altro che la definizione della mappa in dimensione di *voxel*. Si mostra la mappa con `pcshow` e la traiettoria

ricostruita. Nelle altre due sezioni si confrontano le traiettorie e si ricavano i valori di *rms* tramite il comando `rmse(a_mod, b, 'all')`, dove a-mod non è altro che il vettore della *ground truth* sotto-campionato per poter essere confrontato con le traiettorie trovate. In altri script sono stati eseguiti più controlli, come i tempi delle singole sezioni di codice e gli andamenti degli errori di traslazione e rotazione, sia in modulo che divisi per componenti. Essi verranno riportati nella parte di confronto dei risultati.

4.1.3 Risultati ICP

Si sono eseguite le simulazioni delle traiettorie illustrate in figura 3.3. Per poter eseguire un confronto tra traiettorie diverse, si sono utilizzati gli stessi parametri per tutte le simulazioni. É bene notare che i parametri utilizzati non sono necessariamente i parametri ottimali per la specifica traiettoria. Esiste sicuramente una combinazione di parametri che funziona meglio, tuttavia si è visto che questi parametri garantivano convergenza per tutte le traiettorie, dando un rapporto tempo computazionale su errore accettabile.

Parametro	Settaggio
`skipFrames`	4
`dowsamplePercent`	0.3
`inlierRatio`	0.2
`maxTolerableRMSE`	0.8
`Rloop`	8
Zona accettata	$R \in [2, 70]$
`MaxIter`	50

Tabella 4.1: Parametri metodo ICP

Traiettoria chiusa

Al fine di comprendere meglio i risultati, verranno riportati alcuni dati notevoli estratti nel post-processo. La prima traiettoria studiata è una traiettoria chiusa a singolo loop.

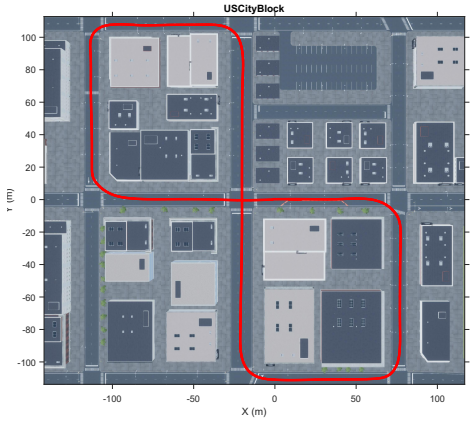


Figura 4.1: Traiettoria chiusa singolo loop

Parametro	Valore
Lunghezza	812.25 m
Tempo	81.23 s
Velocità media	10 m/s
Numero di frame	813
Numero frame chiave	204

Tabella 4.2: Parametri traiettoria chiusa singolo loop

In figura 4.2 e 4.3a viene mostrata la mappa, generata fondendo e allineando le nubi di punti con le informazioni della ricostruzione, mentre in figura 4.4 è mostrata la traiettoria ricostruita.

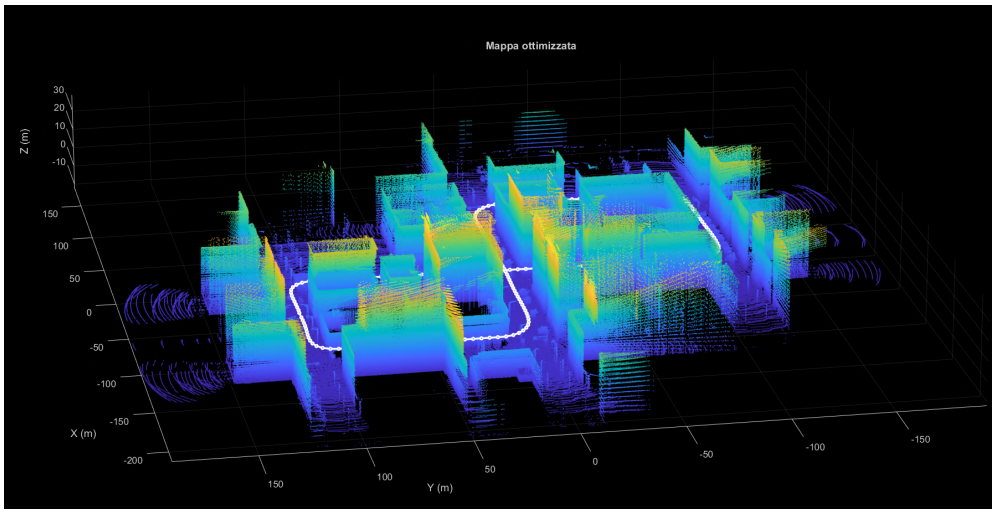
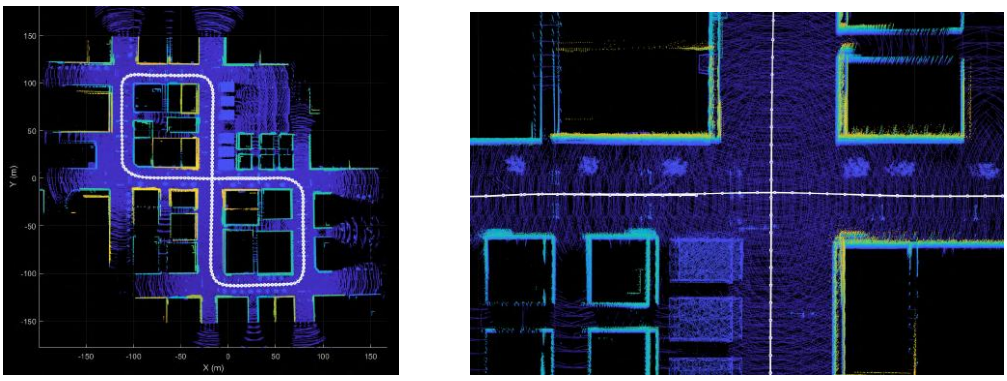


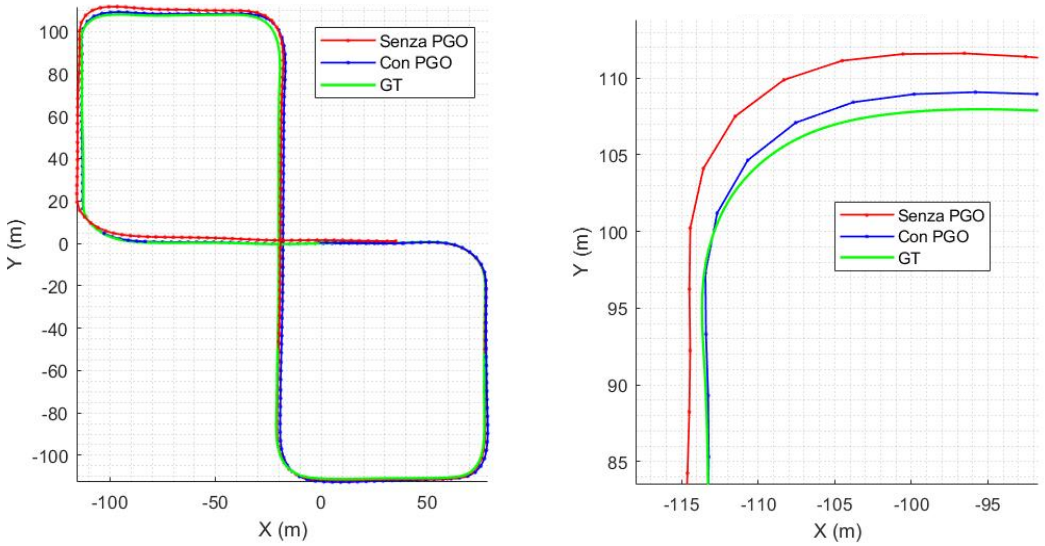
Figura 4.2: Mappa traiettoria chiusa metodo ICP

Nella figura 4.3b si può notare come la mappa presenti dei difetti di disallineamento delle nubi, anche se la ricostruzione complessiva è accettabile.



(a) Vista dall'alto mappa **(b)** Zoom vista dall'alto mappa

Figura 4.3: Mappa traiettoria chiusa metodo ICP (altre viste)



(a) Traiettoria completa **(b)** Zoom

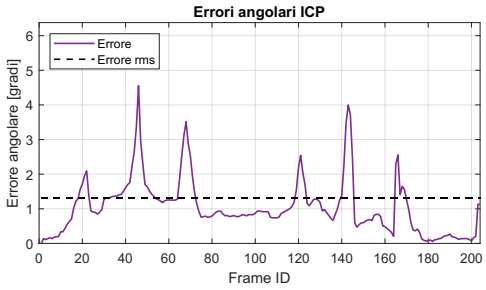
Figura 4.4: Traiettoria chiusa metodo ICP

Nella figura 4.4a vengono riportate sia la traiettoria senza ottimizzazione che la traiettoria dopo la *pose graph optimization*. Nello zoom in figura 4.4b è ben evidenziato come l'ottimizzazione abbia migliorato la ricostruzione di tutta la traiettoria e non solo della singola chiusura del loop. Sebbene la ricostruzione sia molto accettabile, essa presenta degli errori evidenziabili meglio tramite alcuni risultati numerici (tabella 4.3) e grafici (figure 4.5).

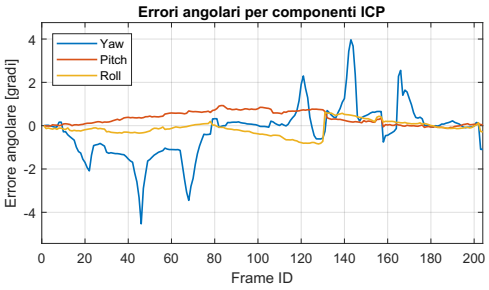
Parametro	Risultato
Errore rms pre pgo	2.046 m
Errore rms pgo	1.169 m
Errore rms %	0.14 %
Errore max	3.351 m
Errore angolare rms	1.31 °
Errore angolare max	4.56 °

Tabella 4.3: Risultati traiettoria chiusa metodo ICP

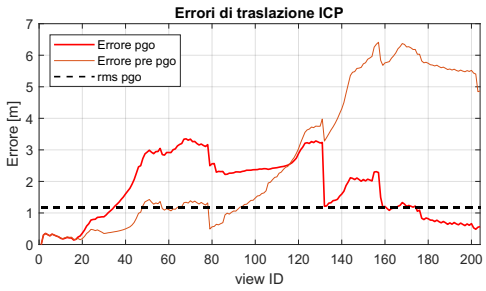
L'ottimizzazione ha ridotto l'errore di *rms* del 43% circa, la ricostruzione presenta comunque un errore di *rms* non trascurabile, che tuttavia, relativamente alla distanza percorsa, è inferiore all'1%. L'errore angolare è dell'ordine del grado. In tabella 4.3 vengono anche mostrati i valori di picco degli errori di traslazione e degli errori angolari. Per meglio comprendere da dove derivino questi valori, di seguito sono riportati alcuni grafici.



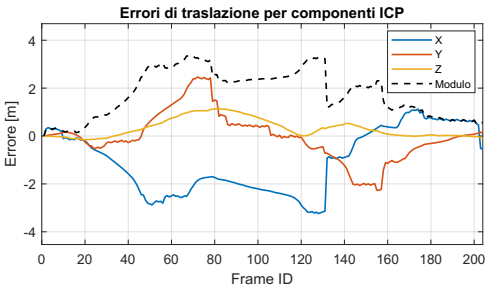
(a) Errore angolare



(b) Errore angolare per componenti



(c) Errore di traslazione



(d) Errore di traslazione per componenti

Figura 4.5: Risultati grafici traiettoria chiusa metodo ICP

In figura 4.5a viene mostrato l'andamento degli errori angolari considerando la rotazione attorno ad un unico asse (quindi l'errore angolare complessivo), mentre in figura 4.5b vengono mostrati gli errori angolari per componente. Si può subito notare come essi siano prevalentemente dovuti alla stima dell'imbardata e siano concentrati nelle ricostruzioni in fase di curva. In 4.5c viene mostrato l'andamento degli errori di traslazione prima e dopo l'ottimizzazione grafica. Si nota come l'ottimizzazione abbia complessivamente ridotto l'errore, in quanto, sebbene localmente l'errore con ottimizzazione potrebbe essere maggiore, complessivamente la deriva viene ridotta (si vedano frame finali). L'aumento locale dell'errore in seguito all'ottimizzazione è dovuto alla ridistribuzione del grafo. In simulazione di traiettorie aperte la compensazione della deriva non avviene, e l'errore di traslazione tenderà a crescere. In figura 4.5d si può notare come l'errore sia prevalentemente dovuto a scostamenti in x e y (ovvero planari). Per quanto

riguarda il tempo computazionale, in tabella 4.4 sono riportati alcuni risultati della simulazione.

Parametro	Risultato
Tempo computazionale	26.62 s
PTPF (*processing time per frame*)	0.131 s
Tempo registrazione ICP	0.119 s
PTPF massimo accettabile	0.4 s

Tabella 4.4: Tempi traiettoria chiusa metodo ICP

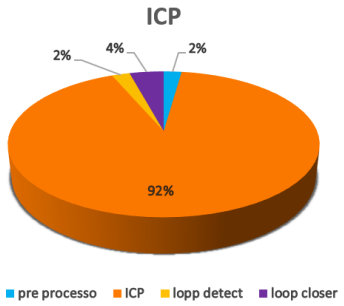


Figura 4.6: Percentuali

Il tempo computazionale è calcolato per la sola parte front-end, ovvero escludendo l'ottimizzazione e la generazione dei grafici. Esso risulta inferiore al tempo di traiettoria, facendo sì che la ricostruzione non sia in ritardo. Il PTPF è il tempo computazionale per singolo frame, mentre il tempo di registrazione è il tempo della sola riga di codice relativa al metodo ICP. Il PTPF massimo è stato calcolato con $PTPF_{max} = skipFrame/f_{lidar}$, dove f_{idar} è la frequenza di acquisizione. Se il PTPF superasse il PTPF massimo ($PTPF < PTPF_{max}$), inizierebbe un ritardo nella ricostruzione. Nel grafico a torta sono raffigurati i tempi medi delle singole parti di codice rapportati al PTPF. É evidente come la ricostruzione occupi la maggior parte del tempo computazionale (figura 4.6). In figura 4.7 viene riportato un istogramma cumulato dei tempi computazionali. I picchi viola rappresentano le chiusure del loop. I tempi per pre-processo e rilevazione del loop sono appena visibili a conferma di quanto riportato nel grafico a torta 4.6. Anche se localmente il tempo si avvicina al PTPF massimo nei punti di chiusura del loop, comunque il PTPF medio rimane inferiore, garantendo la ricostruzione in diretta.

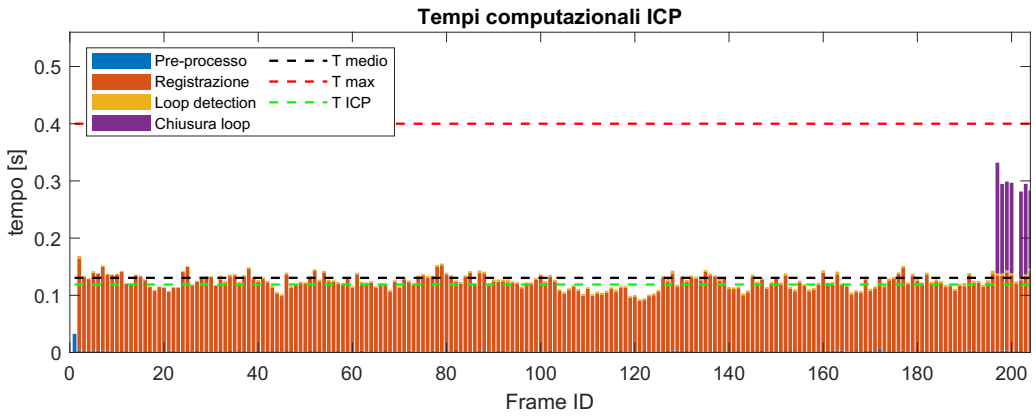


Figura 4.7: Istogramma cumulato tempi di simulazione traiettoria chiusa metodo ICP

Traiettoria aperta

Si è scelto di simulare una traiettoria aperta per vedere come l'algoritmo si comportasse senza ottimizzazione, al fine di valutare bene la deriva vera della traiettoria.

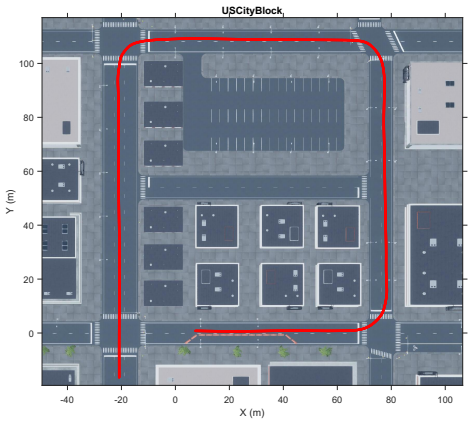
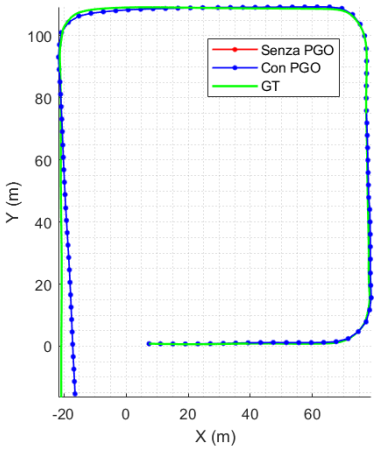


Figura 4.8: Traiettoria aperta

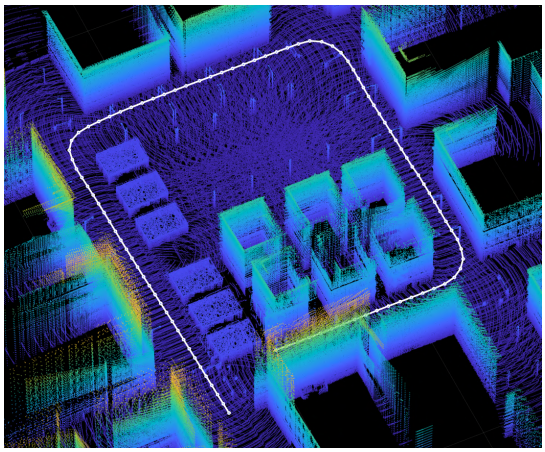
Parametro	Valore
Lunghezza	386.96 m
Tempo	38.70 s
Velocità media	10 m/s
Numero di frame	387
Numero frame chiave	97

Tabella 4.5: Parametri traiettoria aperta

La traiettoria è riportata in figura 4.9a e la mappa in figura 4.9b.



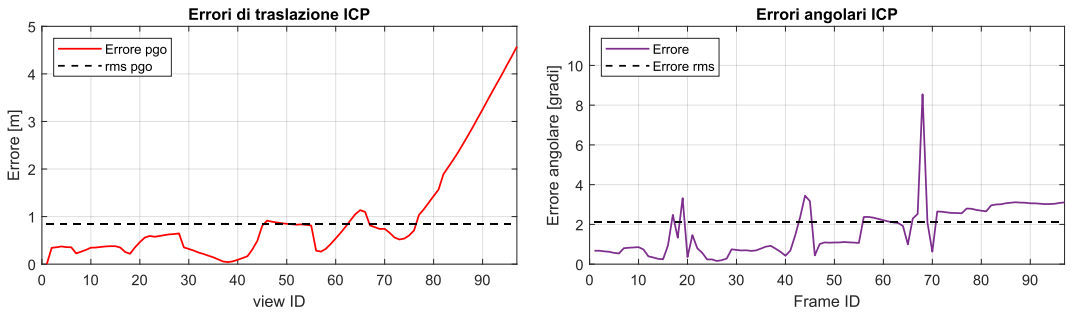
(a) Traiettoria aperta ICP



(b) Mappa traiettoria aperta ICP

Figura 4.9

In questo caso la ricostruzione presenta una deriva maggiore perché non è presente l'ottimizzazione della traiettoria, non essendo presenti loop. L'assenza di ottimizzazione crea una deriva ben evidenziata nell'andamento degli errori di traslazione (figura 4.10a). Gli errori angolari sono in corrispondenza delle zone di curva e anch'essi derivano prevalentemente dalla determinazione dell'imbardata (figura 4.10b).



(a) Errore traslazione

(b) Errori angolari

Figura 4.10: Risultati grafici traiettoria aperta metodo ICP

In tabella 4.6 sono riportati i risultati ottenuti. L'errore percentuale è comparabile a quanto ottenuto in tabella 4.3 per la traiettoria chiusa, in quanto, pur avendo una traiettoria più corta, è presente l'errore causato dalla deriva non compensata dall'ottimizzazione. I tempi PTPF sono comparabili a quanto ottenuto in precedenza.

Parametro	Risultato
Errore rms	0.843 m
Errore rms %	0.22 %
Errore max	4.574 m
Errore angolare rms	2.12 °
Errore angolare max	8.55 °
Tempo computazionale	11.15 s
PTPF	0.115 s
Tempo ICP	0.109 s
PTPF massimo	0.4 s

Tabella 4.6: Risultati traiettoria aperta metodo ICP

Traiettoria con variazione di quota

In questo caso è stato scelto di far variare la quota al drone in traiettoria quasi elicoidale.

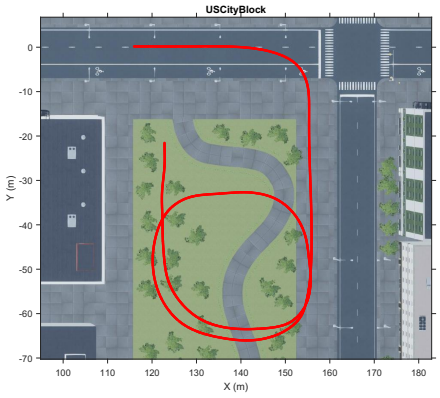


Figura 4.11: Traiettoria con variazione di quota

Parametro	Valore
Lunghezza	274.81 m
Tempo	54.69 s
Velocità media	5 m/s
Velocità verticale	0.5 m/s
Numero di frame	547
Numero frame chiave	137

Tabella 4.7: Parametri traiettoria con variazione di quota

La ricostruzione ha prodotto la traiettoria riportata in figura 4.12.

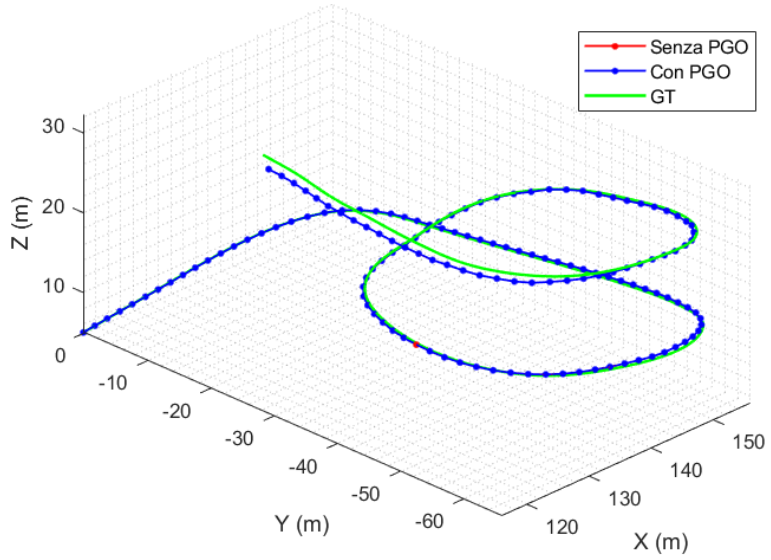
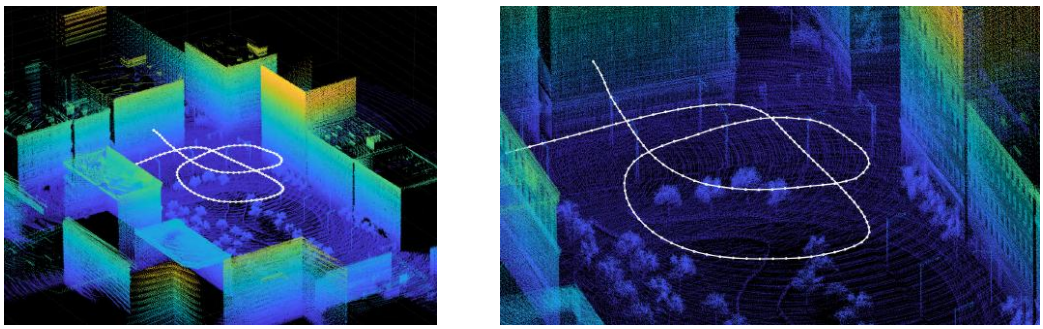


Figura 4.12: Traiettoria con variazione di quota ICP

Anche in questo caso la traiettoria non presenta loop, pur ripassando nello stesso punto ma a quote differenti. Di conseguenza, è presente deriva della traiettoria. In figura 4.13 sono riportati la mappa e un suo ingrandimento.



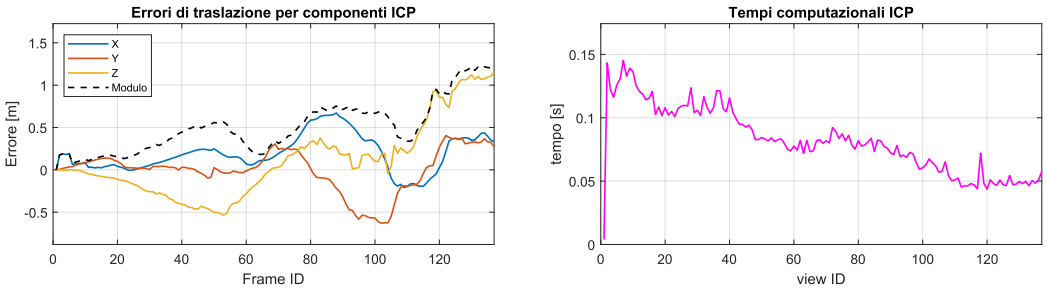
(a) Mappa **(b)** Zoom

Figura 4.13: Mappa traiettoria con variazione di quota ICP

Si nota come le caratteristiche del paesaggio siano state ben ricostruite, fino a livello di dettaglio dei palazzi e degli alberi. In tabella 4.8 sono riportati i risultati ottenuti.

Parametro	Risultato
Errore rms	0.346 m
Errore rms %	0.13 %
Errore max	1.233 m
Errore angolare rms	1.56 °
Errore angolare max	4.69 °
Tempo computazionale	11.26 s
PTPF	0.082 s
Tempo ICP	0.077 s
PTPF massimo	0.4 s

Tabella 4.8: Risultati traiettoria con variazione di quota ICP



(a) Errore per componenti **(b)** Tempi computazionali

Figura 4.14: Risultati grafici traiettoria con variazione di quota ICP

Per quanto riguarda gli errori, in figura 4.14b si può notare come sia presente anche una deriva non trascurabile in Z, a causa della variabilità della quota. Per quanto riguarda i tempi di ricostruzione (figura 4.14b), invece, si può notare come essi diminuiscano man mano che la ricostruzione procede. Questo è dovuto

al fatto che man mano che la quota aumenta, la scansione acquisisce meno punti perché la distanza dagli edifici comincia a superare il range e il FOV del LiDAR, di conseguenza la ricostruzione diventa più veloce. Inoltre, siamo entrati in una traiettoria elicoidale, di conseguenza la trasformazione relativa è quasi costante e, poiché è stata data come primo tentativo per la registrazione successiva, la convergenza sarà più rapida.

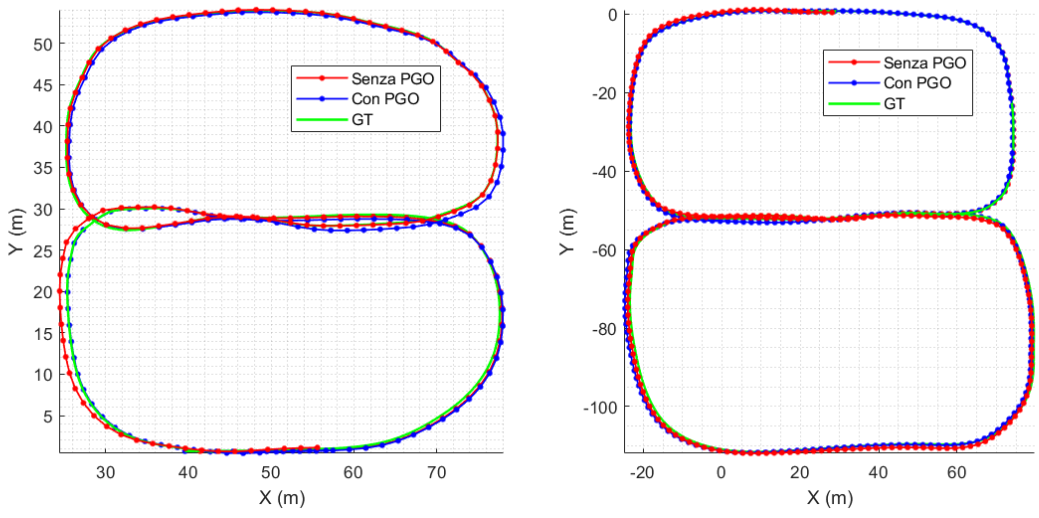
Traiettoria con due chiusure del loop

Sono state eseguite due traiettorie con doppia chiusura del loop, con distanze percorse differenti in modo da valutare quanto la chiusura del loop fosse vantaggiosa.

Parametro	Otto piccolo	Otto grande
Lunghezza	292.82 m	566.88 m
Tempo	58.56 s	113.38 s
Velocità media	5 m/s	5 m/s
Numero di frame	586	1134
Numero frame chiave	147	284

Tabella 4.9: Parametri traiettorie a otto con doppia chiusura del loop

Il risultati della ricostruzione in termini di traiettoria sono riportati in figura 4.15.



(a) Otto piccolo **(b)** Otto grande

Figura 4.15: Traiettorie a otto metodo ICP

Entrambe le ricostruzioni sono buone e, in entrambi i casi, si può vedere come l'ottimizzazione abbia migliorato ulteriormente la ricostruzione. I risultati numerici ottenuti sono riportati in tabella 4.10.

Parametro	Otto piccolo	Otto grande
Errore rms pre pgo	0.681 m	0.719 m
Errore rms pgo	0.447 m	0.533 m
Errore rms %	0.15 %	0.09 %
Errore max	1.463 m	1.718 m
Errore angolare rms	1.93 °	1.29 °
Errore angolare max	7.21 °	4.27 °
Tempo computazionale	21.43 s	40.67 s
PTPF	0.146 s	0.143 s
Tempo ICP	0.118 s	0.120 s
PTPF massimo	0.4 s	0.4 s

Tabella 4.10: Risultati traiettorie a otto ICP

Si può notare come la doppia chiusura del loop garantisca buoni risultati pressoché indipendenti dalla lunghezza della traiettoria. Lo spostamento medio percentuale, infatti, diminuisce perché l'errore è simile, ma la lunghezza della traiettoria aumenta. Per quanto riguarda i tempi computazionali, in questo caso avendo due chiusure del loop il PTPF è aumentato rispetto alle traiettorie precedenti, mentre il tempo di ricostruzione è rimasto confrontabile. Negli istogrammi 4.16 e 4.17 è ben evidenziato come il tempo computazionale aumenti molto nei punti in cui vengono localizzate le chiusure del loop, e che l'aumento sia da imputare al calcolo della trasformazione di chiusura.

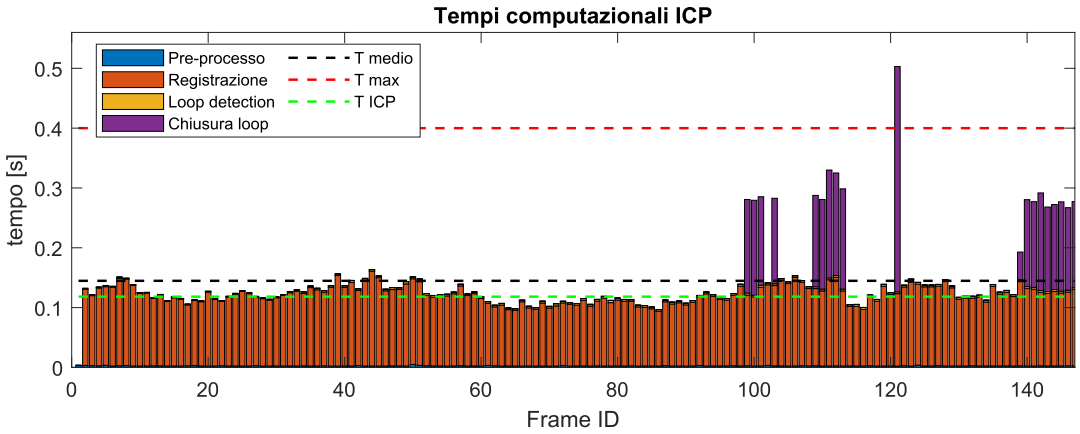


Figura 4.16: Istogramma cumulato tempi computazionali otto piccolo ICP

Nei grafici 4.18a e 4.18b si possono notare andamenti degli errori angolari comparabili, mentre nei grafici 4.18c e 4.18d si può notare come gli errori di traslazione non abbiano andamento crescente come nel caso riportato in 4.10a, grazie alle chiusure del loop. Negli stessi grafici sono riportati anche gli andamenti degli errori prima dell'ottimizzazione, in modo da evidenziare bene l'effetto positivo che essa ha avuto sull'andamento complessivo degli erroi, in particolar modo nell'ultima parte di ricostruzione.

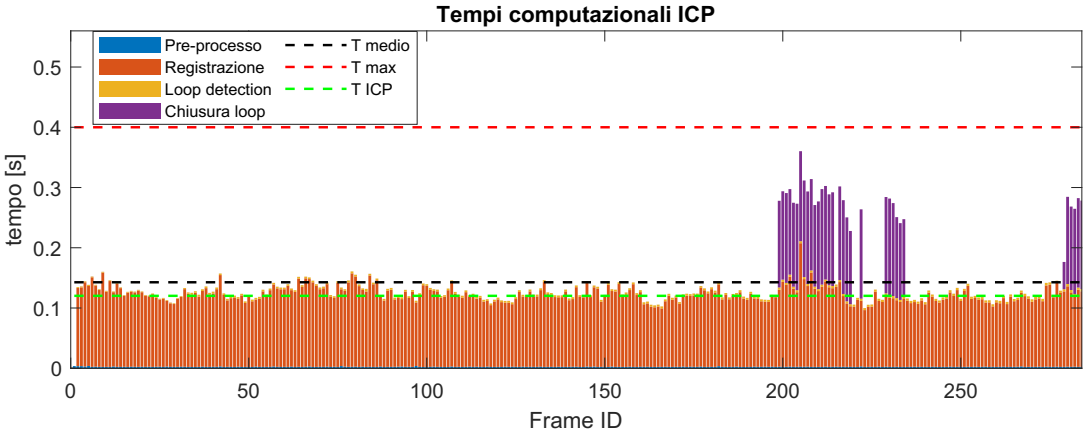
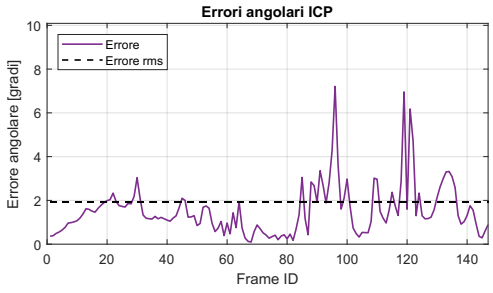
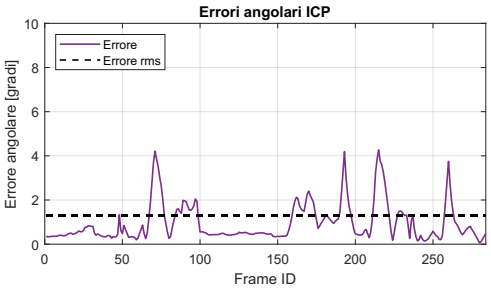


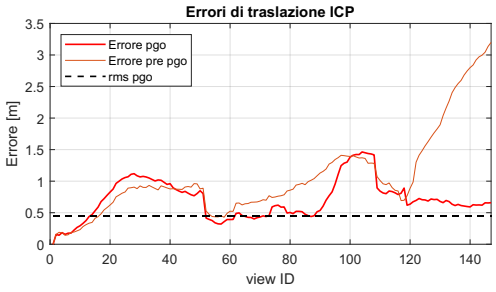
Figura 4.17: Istogramma cumulato tempi computazionali otto grande ICP



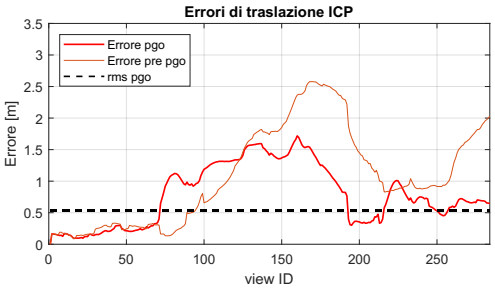
(a) Errore angolare piccola



(b) Errore angolare grande



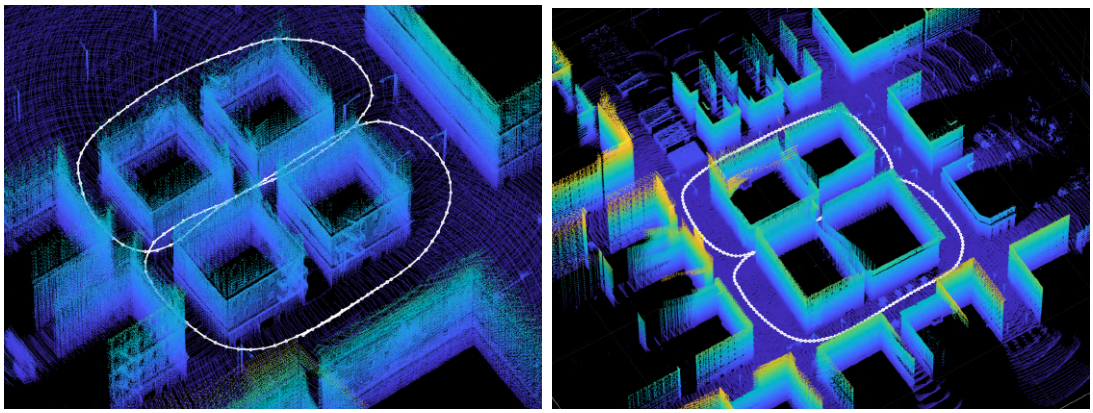
(c) Errori traslazione otto piccolo



(d) Errori traslazione otto grande

Figura 4.18: Risultati grafici traiettorie a otto ICP

Le due mappe ricostruite sono rappresentate in figura 4.19.



(a) Otto piccolo **(b)** Otto grande

Figura 4.19: Mappe traiettorie a otto metodo ICP

4.2 Implementazione metodo LOAM

Il metodo LOAM viene di seguito illustrato, spiegando solo le parti che differiscono rispetto al precedente metodo ICP.

4.2.1 Front-end

```
%% Parametri
numFrames    = length(pClouds);
skipFrames   = 4;
Rmin = 2;  Rmax = 50;
maxTolerableRMSE  = 0.6;
Rloop = 3; MaxIter = 50;
% numero massimo di piani estratti
maxPlanarSurfacePoints = 8;
% grid step per il downsampling post LOAM
gridStep = 1.5;
% dimensionen del voxel per la LOAM Map
voxelSize = 1.5;
% inizzializzazione dell LOAM map
loamMap = pcmaploam(voxelSize);
```

Listing 4.8: Parametri LOAM

Come si può notare i parametri utilizzati sono molto simili a quelli del metodo ICP, tuttavia non sono presenti né il parametro che definisce la percentuale di *inlier* da ritenere validi, né la percentuale di *downsampling* da effettuare sulla nube. Questo perché la riduzione dei punti verrà fatta sulle *feature*. Si possono

invece riscontare tre parametri importanti. Il primo è `maxPlanarSurfacePoints`, il quale rappresenta il massimo numero di punti appartenenti ad una superficie piana contenuti in una *scan region* (zona di scansione). Il numero di default è 1, il quale risulta troppo basso per una corretta determinazione delle *feature* piane. Le zone di scansione di default sono 6, ovvero la nube di punti viene divisa in 6 zone in cui verranno estratti spigoli e piani. Un altro parametro importante è il `gridStep`, ovvero la dimensione di griglia necessaria per il *downsampling* delle *feature* estratte. Infine, il parametro `voxelSize` che definisce la dimensione di griglia per la LOAM map, rappresenta la sua definizione. Tramite questo parametro si può inizializzare la `loamMap` mediante il comando `pcmaploam(voxelSize)`. L'inizializzazione delle altre variabili è analoga al metodo ICP.

```
% Lettura e preprocesso nube di punti
ptCloud = pClouds{n};
% Rimozione punti non necessari
selectedIdx = findPointsInCylinder(ptCloud,[Rmin Rmax]);
ptCloud_EC = select(ptCloud,selectedIdx,OutputSize="full");
% Estrazione e downsampling delle LOAM features
pt_LOAM = detectLOAMFeatures(ptCloud_EC,'MaxPlanarSurfacePoints',
    maxPlanarSurfacePoints);
pt_LOAM = downsampleLessPlanar(pt_LOAM,gridStep);
```

Listing 4.9: Pre-processo LOAM

Il pre-processo inizia in modo analogo al metodo ICP rimuovendo i punti non necessari, ma si differenzia in modo concettuale. Infatti, il metodo LOAM si basa su *feature*, di conseguenza, tramite il comando `detectLAOMFeatures` vengono estratti descrittori planari e lineari dalla nube di punti. Per ridurre il tempo computazionale si procede sottocampionando i descrittori così estratti tramite il comando `downsampleLessPlanar`, in cui entra in gioco il parametro `gridStep`. Si entra quindi nella fase di registrazione non più con una nuvola di punti ma con un inseme di descrittori.

```
% Registrazione: calcolo della trasformazione relativa
tform = pcregisterloam(pt_LOAM,pt_LOAM_prev,'InitialTransform',tform, '
    MatchingMethod','one-to-many');
% Ricerca della posizione assoluta nella LOAM map
absPose = findPose(loamMap,pt_LOAM,tform);
% Aggiunta dei punti features alla LOAM map
addPoints(loamMap,pt_LOAM,absPose);
```

Listing 4.10: Registrazione LOAM

Come in precedenza, anche qui la righe di codice più importanti sono sicuramente quelle relative alla registrazione. Come spiegato nella parte teorica, la registrazione comprende tre passaggi: l'estrazione delle *feature* mostrate nel codice 4.9, la registrazione LOAM e l'affinamento della registrazione tramite la mappa complessiva delle *feature* estratte (da qui in avanti semplicemente LOAM map). La registrazione LOAM viene eseguita dal comando `pcregisterloam`. In questo caso in input si avranno le *feature* estratte nell'iterazione corrente e le *feature* estratte nell'iterazione precedente. Inoltre, viene definita la trasformazione di primo tentativo `InitialTransform` come la trasformazione relativa calcolata nell'iterazione precedente, facendo quindi l'ipotesi di moto costante. Viene scelto di usare il metodo di corrispondenza `one-to-many`, ovvero il calcolo della trasformazione verrà eseguito tra un punto della nube corrente e più punti della nube precedente. Si passa quindi all'affinamento della posizione assoluta tramite il comando `findPose(loamMap, pt_LOAM, tform)`, in cui viene data in input la trasformazione appena calcolata come primo tentativo per la ricerca della trasformazione assoluta. Questo passaggio non sarebbe strettamente necessario, in quanto basterebbe concatenare la matrice di trasformazione assoluta dell'iterazione precedente con quella relativa attuale, tuttavia permette di affinare il calcolo della trasformazione assoluta. Questo passaggio verrà eseguito nelle simulazioni successive, per confrontare la ricostruzione LOAM e la ricostruzione tramite LOAM map. Viene in ultimo eseguita la memorizzazione delle *feature* tramite il comando `addPoints`. La rimanente parte del codice è pressoché analoga, eccezion fatta per la determinazione della trasformazione di chiusura del loop, la quale viene eseguita con il metodo LOAM e di conseguenza comporta una serie di comandi atti alla sua esecuzione.

Il back-and è del tutto analogo a quanto riportato nel codice 4.6 e 4.7.

4.2.2 Risultati LOAM

Sono state eseguite le simulazioni delle stesse traiettorie utilizzate per il metodo ICP, implementando sia il metodo con ricostruzione LOAM che il metodo con correzione tramite LOAM map. Di seguito verranno riportati i dati ottenuti per le varie soluzioni confrontando quindi i tre metodi sviluppati fin'ora. I parametri utilizzati per impostare le simulazioni sono riportati in tabella 4.11. Al fine di confrontare in modo più accurato i metodi presentati, si è scelto di utilizzare gli stessi frame saltati del metodo ICP. Di conseguenza i frame osservati saranno uguali e i dati specifici della traiettoria gli stessi riportati nelle tabelle

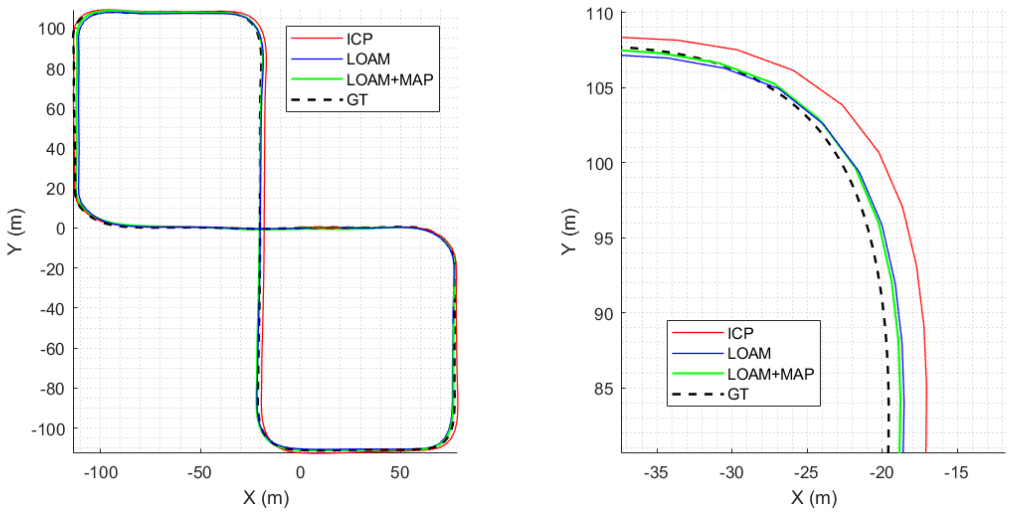
corrispondenti alle traiettorie in questione.

Parametro	LOAM map	LOAM
`skipFrames`	4	4
`gridStep`	1.5	1.5
`maxPlanarSurfacePoints`	8	8
`maxTolerableRMSE`	0.6	0.9
`Rloop`	3	5
Zona accettata	$R \in [2, 50]$	$R \in [2, 50]$
`voxelSize`	1.5	-

Tabella 4.11: Parametri metodo LOAM

Traiettoria chiusa

Si è studiata la traiettoria chiusa facendo riferimento alla tabella 4.2. Di seguito vengono riportate le traiettorie ricavate con i tre metodi al fine di confrontarne la bontà di ricostruzione.



(a) Traiettoria completa

(b) Zoom in una zona di curva

Figura 4.20: Traiettoria chiusa ricostruita con i tre metodi a confronto

Dal grafico 4.20b si può notare come la ricostruzione tramite metodo LOAM map sia superiore rispetto al metodo ICP. Questo verrà confermato anche dai risultati numerici successivi (tabella 4.12). In figura 4.21 è riportata una vista dall'alto della mappa ricostruita.

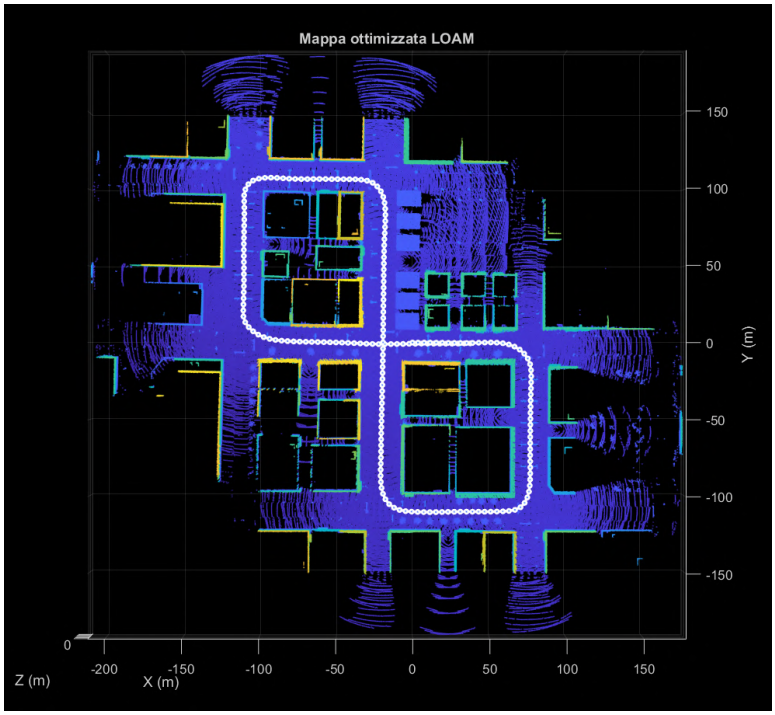
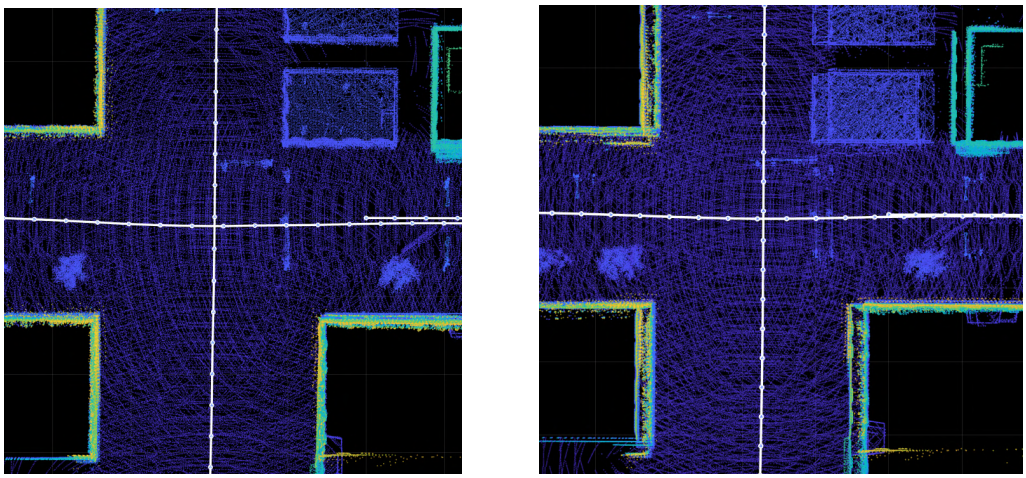


Figura 4.21: Mappa traiettoria chiusa metodo LOAM, vista dall'alto

La ricostruzione della mappa risulta migliore in termini di dettagli estratti, come si può notare confrontando le figure 4.22a e 4.22b.



(a) Metodo LOAM map **(b)** Metodo ICP

Figura 4.22: Confronto di un dettaglio della mappa traiettoria chiusa

I risultati ottenuti a confronto sono riportati in tabella 4.12. Si nota come gli errori di *rms* associati al metodo LOAM con LOAM map siano minori rispetto al metodo LOAM e al metodo ICP. Tuttavia, i tempi computazionali aumentano in quanto è presente un ulteriore passaggio di registrazione. In figura 4.23 vengono confrontati i tempi computazionali percentuali per i diversi metodi.

Parametro	LOAM map	LOAM	ICP
Errore RMS pre pgo	0.856 m	1.574 m	2.046 m
Errore rms pgo	**0.856 m**	1.004 m	1.169 m
Errore rms %	0.11 %	0.12 %	0.14 %
Errore max	2.960 m	3.626 m	3.351 m
Errore angolare rms	1.37 °	**1.28 °**	1.31 °
Errore angolare max	4.36 °	4.18 °	4.56 °
Tempo computazionale	52.06 s	34.81 s	**26.62 s**
PTPF	0.255 s	0.171	0.131 s
Tempo registrazione	0.099 s	0.130 s	0.119 s
Tempo LOAM map	0.113 s	-	-
PTPF massimo accettabile	0.4 s	0.4 s	0.4 s

Tabella 4.12: Risultati LOAM e LOAM map traiettoria chiusa

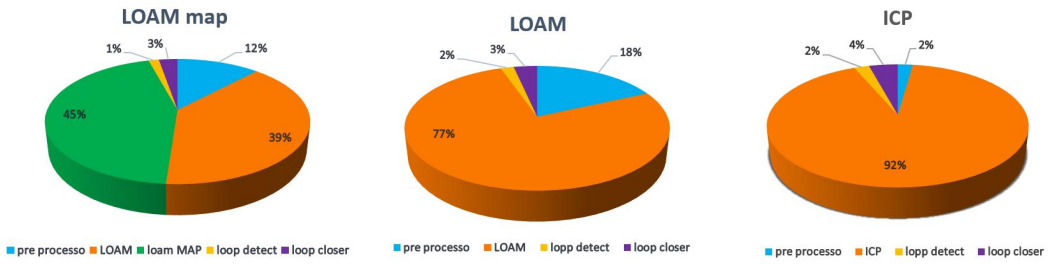


Figura 4.23: Grafici a torta dei tempi computazionali medi delle singole parti di codice

Sia LOAM che LOAM map utilizzano una maggior percentuale del tempo computazionale per il pre-processo, in quanto estraggono le *feature* dalla nube di punti. I tempi di LOAM e ICP della sola registrazione sono abbastanza confrontabili, tuttavia il tempo computazionale complessivo aumenta proprio per il maggior peso del pre-processo. Si nota come il tempo computazionale medio relativo alla sola registrazione dell'algoritmo LOAM sia inferiore nel caso in cui si abbia anche la LOAM map. Questo è dovuto al fatto che la trasformazione di primo tentativo in input all'algoritmo è migliore, in quanto la LOAM map affina il risultato del LOAM. Risulta evidente, inoltre, che il metodo LOAM con LOAM map presenti una grande parte del tempo computazionale per la registrazione nella LOAM map. Il tempo computazionale aggiuntivo apporta comunque una miglioria nella ricostruzione. Se si analizza l'andamento dei tempi, tuttavia, si notano alcuni dettagli importanti.

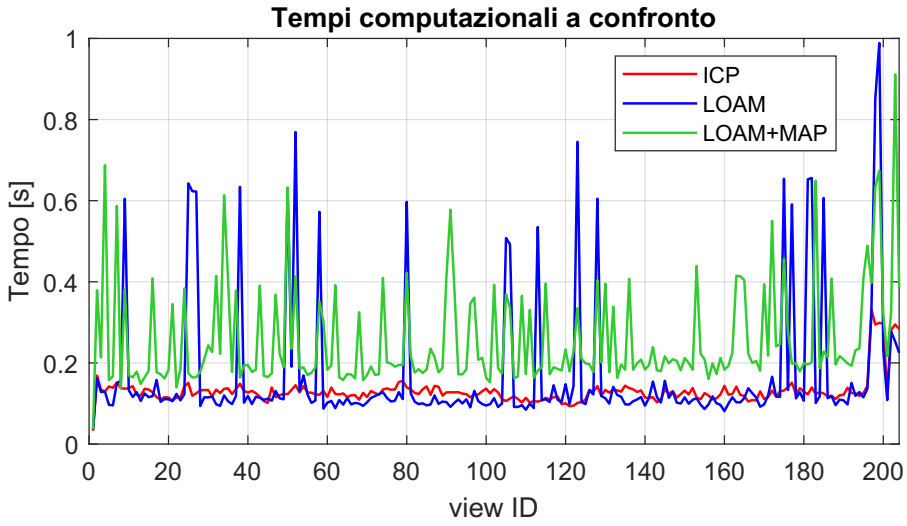


Figura 4.24: Tempi computazionali di iterazione a confronto

In figura 4.24 risulta evidente come il tempo computazionale del metodo LOAM e LOAM map sia molto discontinuo, comportando un aumento del PTPF. Vengono inoltre riportati in figura 4.25 i tempi suddivisi per singola iterazione e per singola parte di codice, in modo da evidenziare bene quale sia il contributo predominante della singola iterazione.

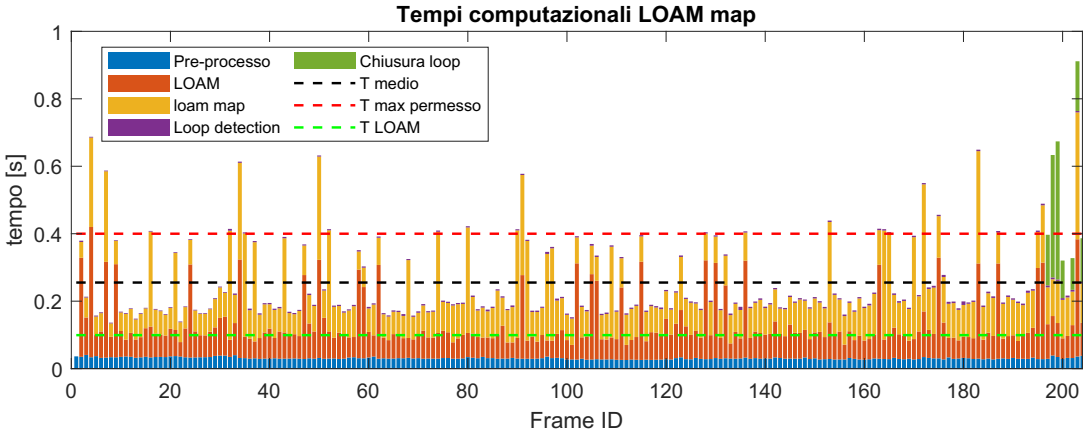
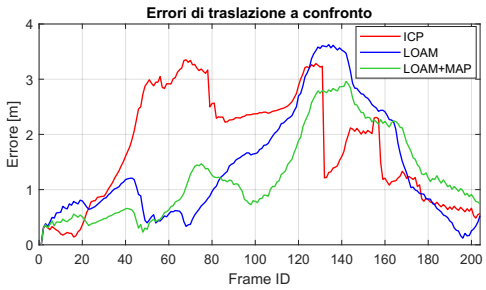
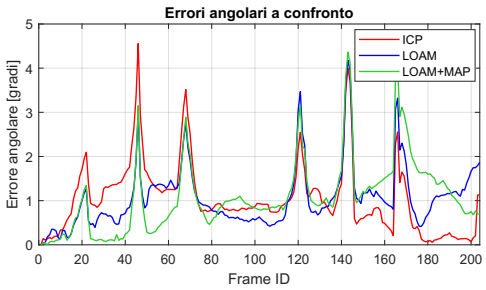


Figura 4.25: Istogramma cumulato tempi computazionali traiettoria chiusa LOAM map

Per quanto riguarda gli errori commessi dai tre metodi, in figura 4.26 vengono riportati a confronto gli errori di traslazione e gli errori angolari. In figura 4.26a si nota come gli errori di traslazione del metodo LOAM map siano quasi sempre inferiori degli altri due metodi. Di seguito vengono riportati altri risultati ottenuti nel caso di LOAM map. In figura 4.26b si può notare come gli andamenti degli errori angolari siano comparabili per i tre metodi.

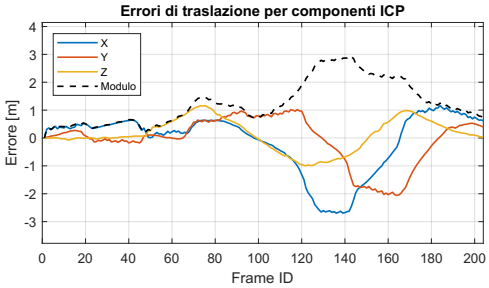


(a) Errori di traslazione a confronto

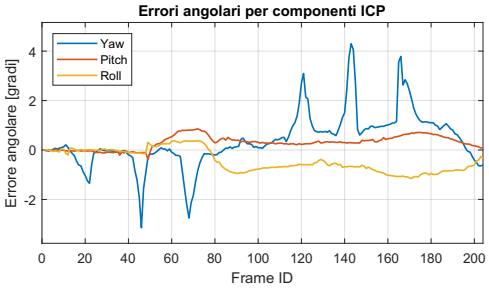


(b) Errori angolari a confronto

Figura 4.26: Risultati grafici a confronto traiettoria chiusa



(a) Errori di traslazione per componenti



(b) Errori angolari per componenti

Figura 4.27: Risultati grafici per componenti traiettoria chiusa metodo LOAM map

In figura 4.28 viene mostrata la LOAM map generata durante la ricostruzione. I punti viola rappresentano punti appartenenti a *feature* planari, mentre i punti verdi sono punti appartenenti a *feature* di tipo linea, come spigoli.

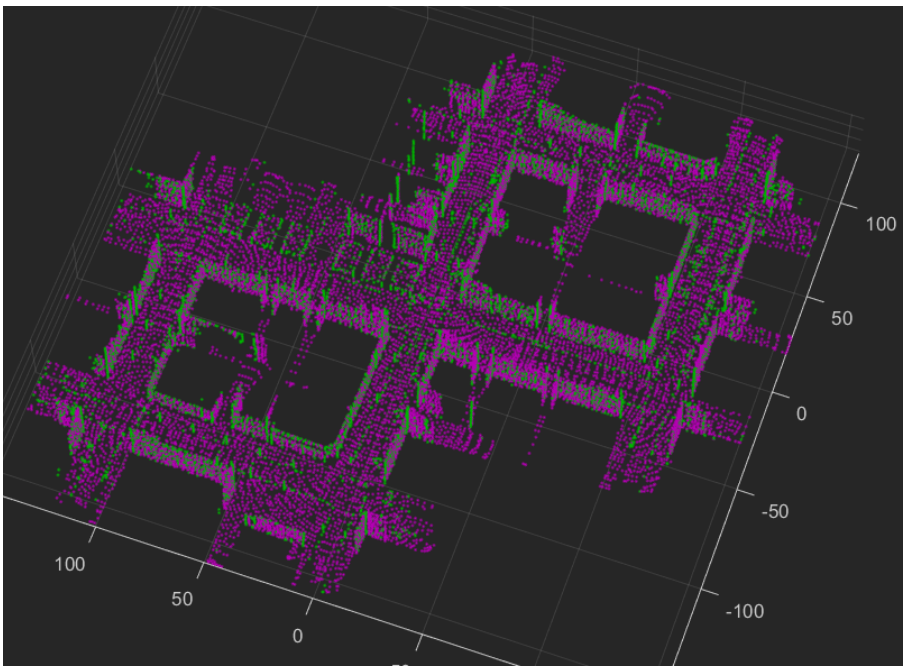
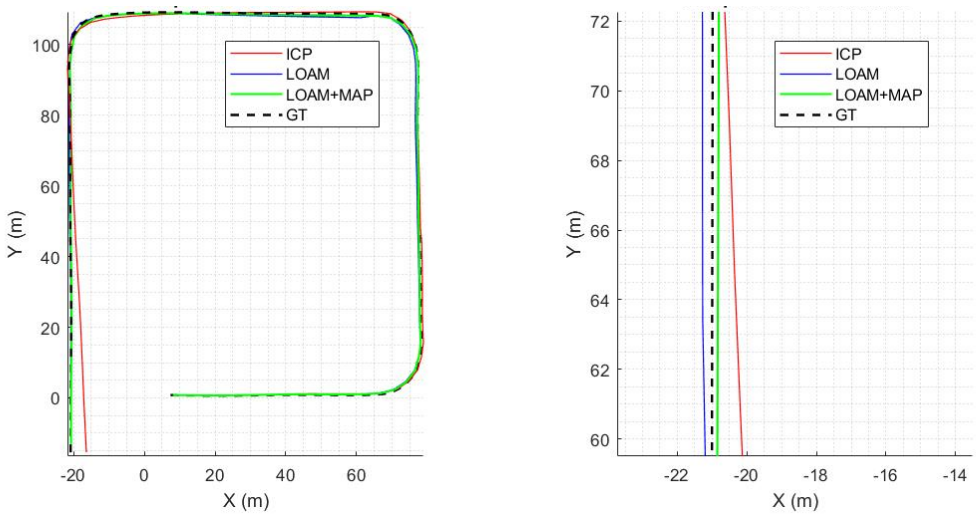


Figura 4.28: Mappa raffigurante le *feature* LOAM estratte (LOAM map)

Traiettoria aperta

La traiettoria aperta (specifiche in tabella 4.5) ci permette di studiare la bontà degli algoritmi in assenza della chiusura del loop. Vengono di seguito riportate le traiettorie per i tre metodi a confronto (figure 4.29a e 4.29b).



(a) Traiettoria aperta **(b)** Ingrandimento

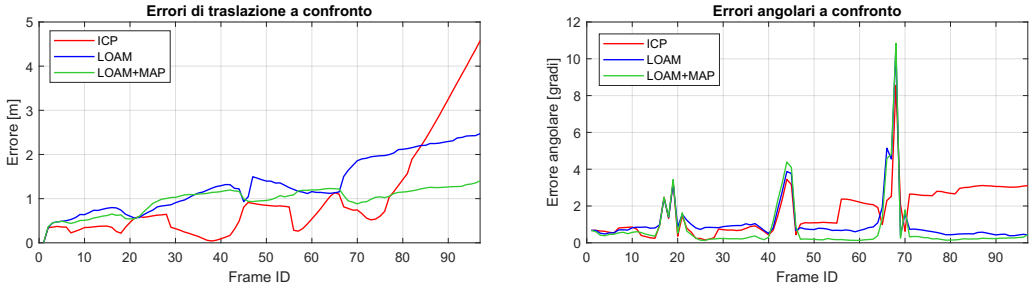
Figura 4.29: Traiettoria aperta metodi a confronto

Si nota molto bene come il metodo ICP presenti una deriva consistente che inizia a presentarsi dopo la terza curva, mentre il metodo LOAM sia più stabile. Nell'ingrandimento della zona in cui il metodo ICP presenta un inizio di deriva (4.29b), si nota come anche in questo caso i metodi LOAM e LOAM map rimangano ancora consistenti. In tabella 4.13 i risultati ottenuti.

Parametro	LOAM map	LOAM	ICP
Errore rms	**0.584 m**	0.844 m	0.843 m
Errore rms %	0.15 %	0.22 %	0.22 %
Errore max	1.404 m	2.472 m	4.574 m
Errore angolare rms	**1.61 °**	1.67 °	2.12 °
Errore angolare max	10.84 °	10.59 °	8.55 °
Tempo computazionale	21.48 s	15.97 s	**11.15 s**
PTPF	0.221 s	0.165 s	0.115 s
Tempo registrazione	0.100 s	0.129 s	0.109 s
Tempo LOAM map	0.083 s	-	-
PTPF massimo accettabile	0.4 s	0.4 s	0.4 s

Tabella 4.13: Risultati traiettoria aperta LOAM e LOAM map

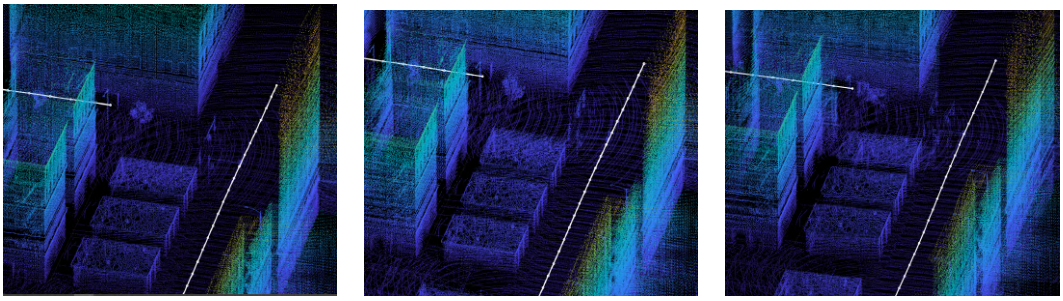
Nei grafici in figura 4.30 è ben evidenziata la deriva presente nei tre metodi, in particolar modo per ICP.



(a) Errori di traslazione **(b)** Errori angolari

Figura 4.30: Risultati grafici a confronto traiettoria aperta

Essendo la ricostruzione più precisa, la generazione della mappa sarà anch'essa più accurata. Per evidenziare ciò vengono messi a confronto tre ingrandimenti che mostrano il progressivo miglioramento della qualità della mappa (figure 4.31).



(a) LOAM map **(b)** LOAM **(c)** ICP

Figura 4.31: Ingrandimenti mappa traiettoria aperta a confronto

Osservando attentamente si potrà notare come i bordi degli edifici nel caso 4.31a siano molto più definiti, fino ad un alto livello di dettaglio. Inoltre, i pali della luce e i semafori presentano fenomeni di sdoppiamento ben visibili nel caso 4.31c. Poiché il drone ripassa per zone già scansionate ma la ricostruzione presenta fenomeni di deriva della traiettoria, essa non sarà perfettamente sovrapposta, a differenza di quella con metodo LOAM map.

Traiettoria con variazione di quota

La traiettoria con variazione di quota (specifiche in tabella 4.7) ha dato alcuni problemi all'algoritmo di ricostruzione LOAM. Successivamente verrà illustrato il motivo. In particolare l'algoritmo non è andato a convergenza con i parametri

di default utilizzati per le altre simulazione e, utilizzando altri parametri, si è ottenuta una soluzione anche se con errori maggiori di quella ottenuta con metodo ICP. In figura 4.32 vengono riportate le traiettorie ottenute con i tre metodi.

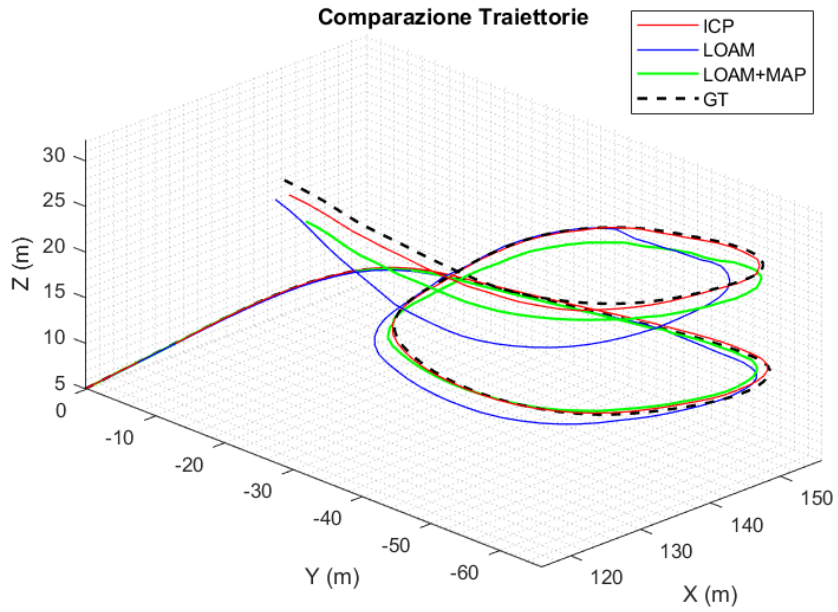
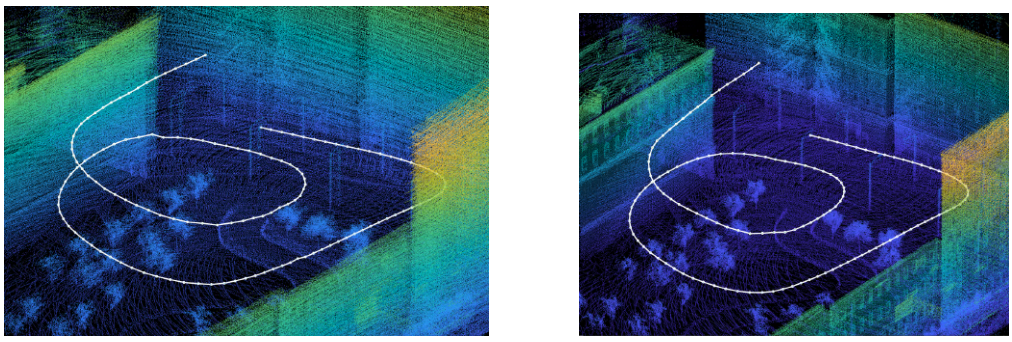


Figura 4.32: Traiettoria con variazione di quota metodo LOAM e LOAM map

É evidente come l'algoritmo non sia riuscito a ricostruire in modo adeguato la traiettoria. Di conseguenza la mappa ricostruita è inconsistente. Nelle figure 4.33a e 4.33b viene riportata una zona della mappa.



(a) Metodo LOAM map **(b)** Metodo ICP

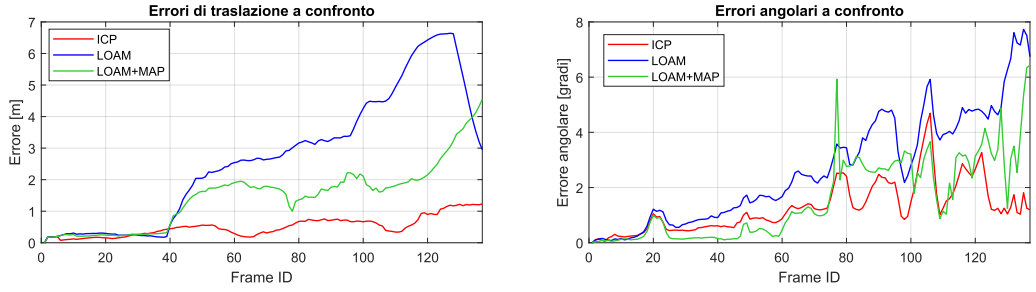
Figura 4.33: Mappe traiettoria con variazione di quota a confronto

In tabella 4.14 vengono collezionati i risultati ottenuti con i parametri di default, modificando solo il parametro del raggio massimo di acquisizione, al fine di garantire la parziale convergenza. Successivamente verrà mostrato come sia possibile ottenere risultati migliori.

Parametro	LOAM map	LOAM	ICP
Errore rms	1.037 m	1.948 m	**0.346 m**
Errore rms %	0.38 %	0.71 %	0.13 %
Errore max	4.551 m	6.641 m	1.233 m
Errore angolare rms	2.19 °	3.32 °	**1.56 °**
Errore angolare max	6.43 °	7.73 °	4.69 °
Tempo computazionale	28.13 s	20.46 s	**11.26 s**
PTPF	0.205 s	0.149 s	0.082 s
Tempo registrazione	0.075 s	0.116 s	0.077 s
Tempo LOAM map	0.091 s	-	-
PTPF massimo accettabile	0.4 s	0.4 s	0.4 s

Tabella 4.14: Risultati traiettoria con variazione di quota LOAM e LOAM map

In figura 4.34 sono riportati alcuni risultati grafici a confronto. In tutti e tre i metodi è presente deriva della traiettoria, tuttavia è evidente come in questo caso i metodi LOAM e LOAM map abbiano ottenuto risultati peggiori del metodo ICP.



(a) Errori traslazione **(b)** Errori angolari

Figura 4.34: Risultati grafici a confronto traiettoria aperta

Dai valori dei tempi computazionali si nota anche che la simulazione ha faticato. Solitamente LOAM si aggira attorno ad un +30-40% rispetto a ICP, mentre LOAM map si aggira attorno ad un +65-70%. In questo caso l'incremento è dell'80% circa per LOAM e fino ad un +150% per LOAM map. Ciò significa un numero maggiore di iterazioni per la convergenza che, dato il limite a 50 interazioni impostato, potrebbe non essere stato raggiunto in più di un'occasione. Per capire meglio perché questo è successo bisogna indagare la LOAM map (figura 4.35). Si nota subito come le *feature* estratte siano inconsistenti e insufficienti. L'algoritmo di estrazione LOAM si basa sulla ricerca di *feature* quali piani e spigoli. Man mano che la quota aumenta, le nubi di punti diventano più rade, e l'algoritmo di registrazione non riesce a convergere agevolmente.

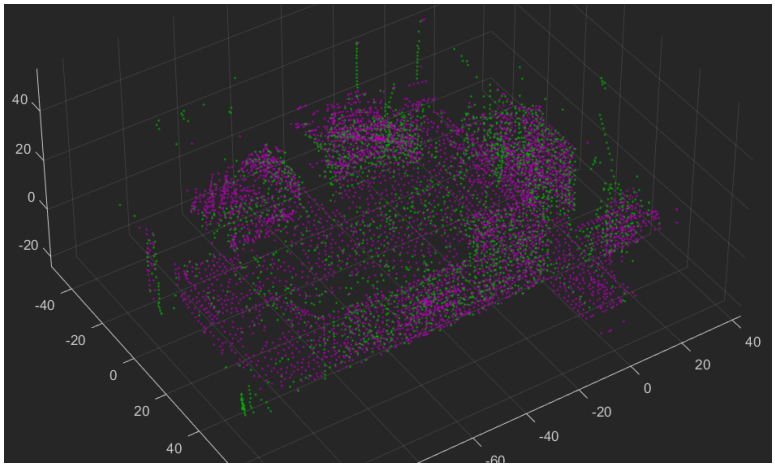


Figura 4.35: LOAM map per traiettoria con variazione di quota con parametri default

Quanto accade è comunque evidente nella ricostruzione ICP che, tuttavia, basandosi su tutti i punti a sua disposizione (a meno del sotto-campionamento) e non sull'estrazione di *feature*, riesce in qualche modo a convergere. Ovviamente se la quota aumentasse ancora fino a superare il range e il FOV del LiDAR, anche la ricostruzione ICP fallirebbe. Tuttavia, riesce a mantenere una accuratezza migliore anche con meno punti scansionati a disposizione.
In figura 4.36 sono anche rappresentati i tempi di singola iterata del metodo. Si nota come, a partire dall'iterazione 80 circa, il tempo computazionale aumenti. Questo è dovuto al fatto che siamo ancora in presenza di *feature* ma la convergenza è difficoltosa. Dall'iterazione 120 invece, le *feature* sono scarse perché siamo in quota e l'algoritmo converge velocemente data la scarsità di punti a disposizione, e accumula gli errori di stima precedenti.

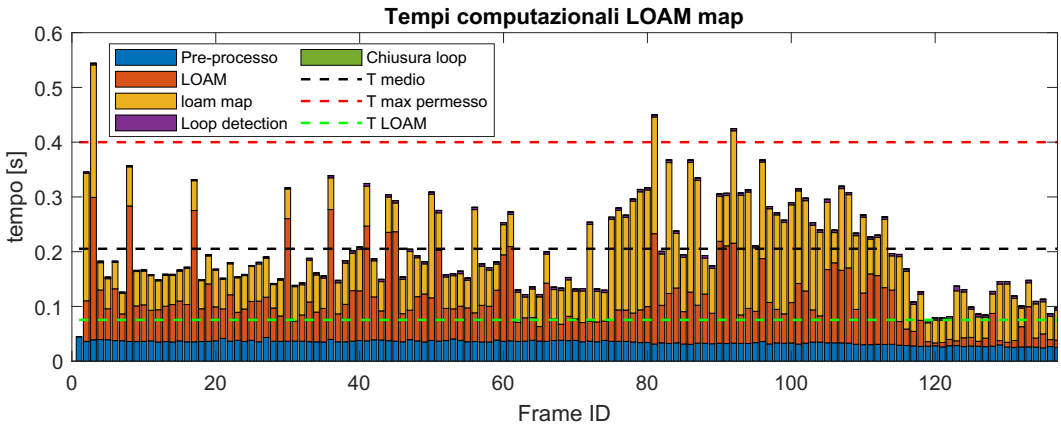
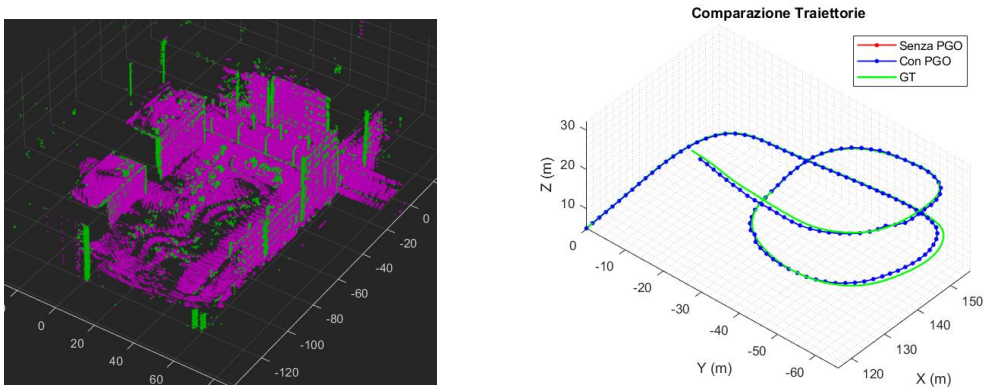


Figura 4.36: Istogramma cumulato tempi computazionali metodo LOAM map traiettoria con variazione di quota

Modificando notevolmente i parametri, come per esempio diminuendo molto il `voxelSize`, la LOAM map acquista definizione, e ciò permette di aumentare l'accuratezza anche se rimanendo a livello inferiore rispetto all'ICP. I tempi computazionali aumentano a tal punto da non essere più in simultanea. Vengono riportate la LOAM map (figura 4.37a) e la traiettoria (figura 4.37b) per una simulazione accettabile con LOAM.
Per evidenziare quanto questa ricostruzione, sebbene ad un primo sguardo possa sembrare buona, sia in realtà molto inefficiente, vengono riportati i principali risultati numerici in tabella 4.15.

Parametro	LOAM map	ICP	
Tempo computazionale	71.97 s	11.26 s	+540%
Errore rms	0.508 m	0.346 m	+45%
Errore angolare	1.60°	1.27 °	+26%

Tabella 4.15: Risultati LOAM map con voxelSize = 0.1 e ICP a confronto

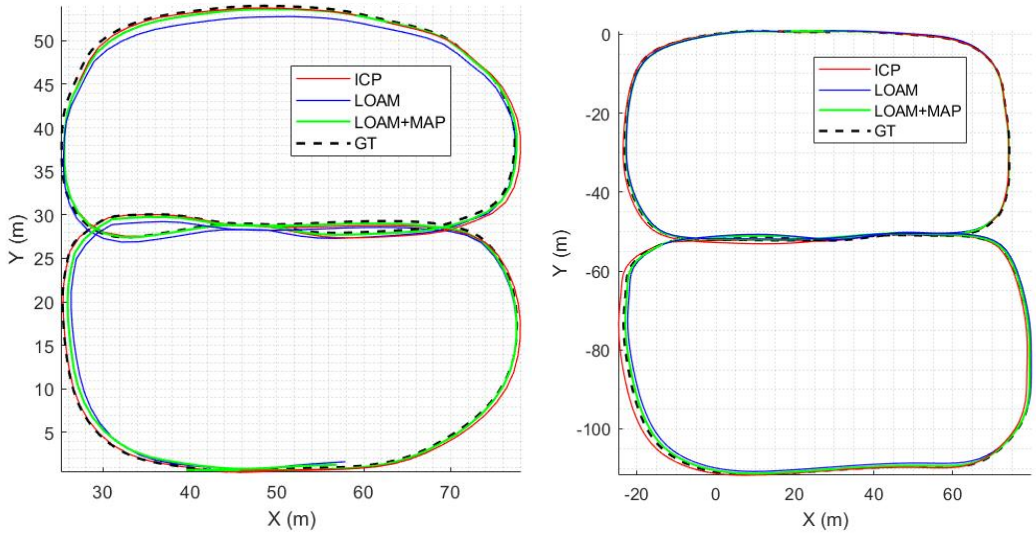


(a) LOAM map **(b)** Traiettoria

Figura 4.37: Mappa e traiettoria voxelSize = 0.1

Traiettoria con due chiusure del loop

In ultima analisi si sono studiate le due traiettorie con doppia chiusura del loop (specifiche in tabella 4.9) , confrontandole tra di loro e tra i vari metodi utilizzati. Le ricostruzioni hanno prodotto le seguenti traiettorie (figura 4.38).



(a) Traiettoria piccola **(b)** Traiettoria grande

Figura 4.38: Traiettorie ricostruite con metodi LOAM e LOAM map a confronto

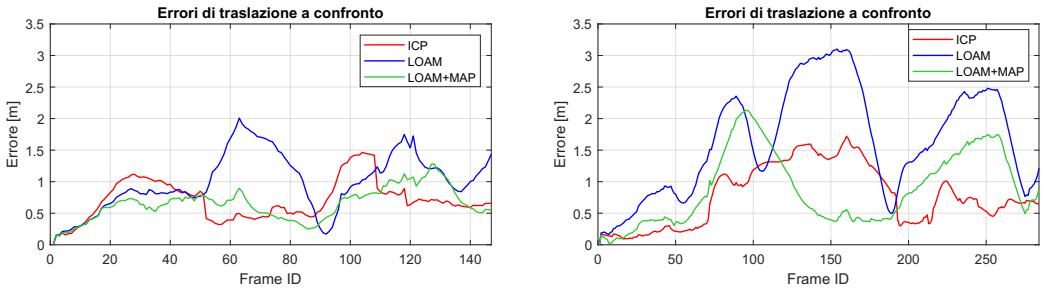
Per comprendere meglio le differenze tra le diverse simulazioni in tabella 4.16 vengono riportati i valori numerici ottenuti.

	Traiettoria piccola			Traiettoria grande		
Parametro	LOAM map	LOAM	ICP	LOAM map	LOAM	ICP
Err. rms pre pgo	0.407 m	0.701 m	0.681 m	0.635 m	2.156 m	0.719 m
Errore rms pgo	**0.402 m**	0.634 m	0.447 m	**0.611 m**	1.074 m	0.533 m
Errore rms %	0.14 %	0.22 %	0.15 %	0.11 %	0.19 %	0.09 %
Errore max	1.281 m	2.008 m	1.463 m	2.131 m	3.103 m	1.718 m
Errore ang. rms	**1.85 °**	2.81 °	1.93 °	1.84 °	3.43 °	**1.29 °**
Errore ang. max	7.04 °	7.98 °	7.21 °	5.69 °	5.33 °	4.27 °
T calcolo	44.37 s	34.50 s	**21.43 s**	71.49 s	49.15 s	**40.67 s**
PTPF	0.302 s	0.235 s	0.146 s	0.252 s	0.173 s	0.143 s
T registrazione	0.114 s	0.169 s	0.118 s	0.094 s	0.116 s	0.120 s
T LOAM map	0.117 s	-	-	0.096	-	-
PTPF massimo	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s

Tabella 4.16: Risultati traiettorie a otto metodi LOAM e LOAM map

Dai risultati si può notare che in generale il metodo LOAM map e il metodo ICP abbiano accuratezze comparabili, sebbene LOAM map impieghi più del doppio del tempo. Questo fatto è dovuto all'ottimizzazione. Se si osservano i risultati prima dell'ottimizzazione si nota subito come LOAM map sia sempre superiore a ICP. Per quanto riguarda il metodo LOAM, esso risulta avere tempi comparabili al metodo ICP ma con risultati inferiori in termini di accuratezza. É interessante notare come i tempi della sola ricostruzione risultino inferiori in LOAM map rispetto a LOAM e ICP. Questo fenomeno è dovuto al fatto che la trasforma-

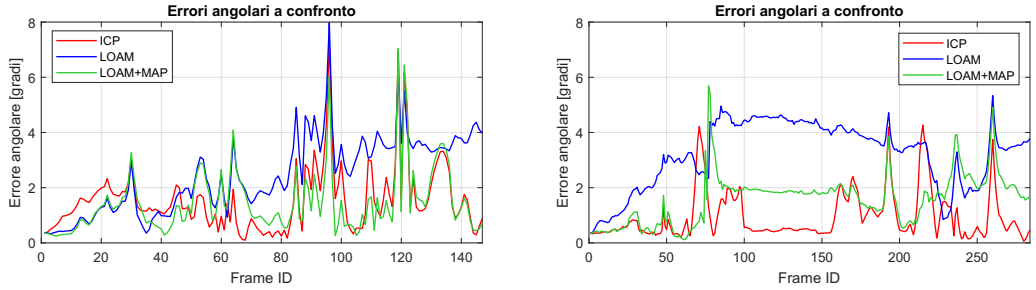
zione di primo tentativo è migliore rispetto ai due casi. Se si osserva il PTPF esso risulterà superiore per la presenza della seconda registrazione nella LOAM map.



(a) Errori traslazione otto piccolo **(b)** Errori di traslazione otto grande

Figura 4.39: Errori di traslazione per le due traiettorie a otto

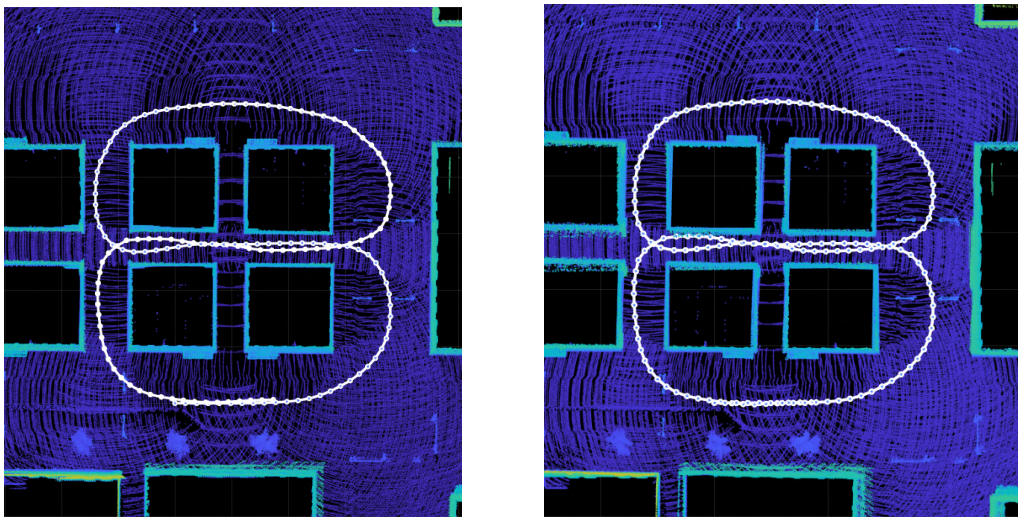
In entrambi i casi l'algoritmo LOAM map permette di migliorare l'accuratezza di ricostruzione. Si possono notare in figura 4.39 andamenti confrontabili per gli errori di traslazione, per il fatto che entrambe le traiettorie vanno incontro ad una doppia chiusura del loop. Ad esempio si veda il grafico 4.39a, dove tra i frame 70 e 100 per tutti e tre i metodi si incontra una diminuzione, andando quindi contro la tendenza alla deriva di traiettorie aperte. Nei frame 180-200 circa per la traiettoria a otto grande (figura 4.39b) si riscontra un andamento analogo. La chiusura del loop migliora non solo il punto di chiusura ma l'intera traiettoria, in particolar modo in un intorno del nodo di chiusura.



(a) Errori angolari otto piccolo **(b)** Errori angolari otto grande

Figura 4.40: Errori angolari per le due traiettorie a otto

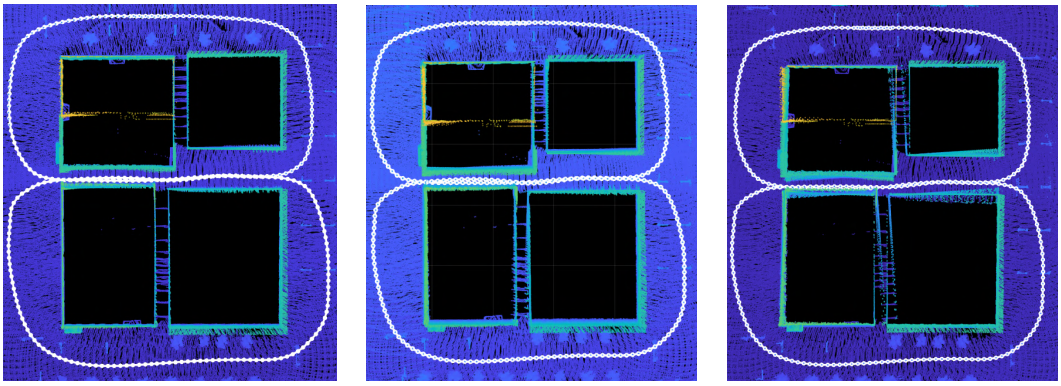
In questo caso gli errori angolari favoriscono il metodo ICP. In tutti e tre tuttavia si riscontrano gli stessi picchi, corrispondenti alle zone di curva. In figura 4.41 alcuni dettagli a confronto delle mappe ricostruite.



(a) Metodo LOAM map **(b)** Metodo ICP

Figura 4.41: Mappe ricostruite con metodi LOAM e ICP per traiettoria a otto piccolo

Osservando attentamente i muri degli edifici si nota come nel caso ICP siano più sgranati anche se non di molto. La generazione della mappa è leggermente superiore nel caso di LOAM map. Per quanto riguarda la mappa dell'otto grande, sebbene gli errori angolari e gli errori di traslazione sembrino favorire il metodo ICP, se si osservano le differenze si potrà notare come nella zona centrale siano presenti dei disallineamenti della mappa generata con metodo ICP. Complessivamente la mappa generata con metodo LOAM map è la migliore in termini di qualità.



(a) LOAM map **(b)** LOAM **(c)** ICP

Figura 4.42: Ingrandimenti mappa traiettoria a otto grande

Capitolo 5

Accoppiamento debole LiDAR-IMU SLAM

In questa sezione si mostrerà come è stata eseguita la pre-integrazione delle letture IMU a partire dai dati generati e come essa sia stata integrata nel codice LOAM. Verranno quindi riportati i risultati ottenuti dalle simulazioni. In ultima analisi si effettueranno degli studi di sensibilità sui parametri di settaggio più importanti utilizzati in questa tesi.

5.1 Fusione debole con IMU

Le letture IMU nei codici *losely coupled*, come in molti dei programmi presentati in tabella 2.4, sono solitamente utilizzate per calcolare una trasformazione di primo tentativo per l'algoritmo di integrazione. Infatti, la frequenza di acquisizione dell'unità IMU è molto più elevata della frequenza di acquisizione LiDAR (100 Hz a fronte di 10 Hz). Questo implica che molte letture IMU verranno acquisite tra una scansione accettata e la successiva (figura 5.1) .

Di conseguenza, tra ogni frame chiave accettato e il successivo avremo delle letture IMU che forniranno accelerazione e velocità angolare in quell'intervallo di tempo. Nel codice 5.1 vengono mostrate le righe di programma Matlab scritte al fine di selezionare le misurazioni IMU contenute tra due frame chiave e calcolare la trasformazione di primo tentativo.

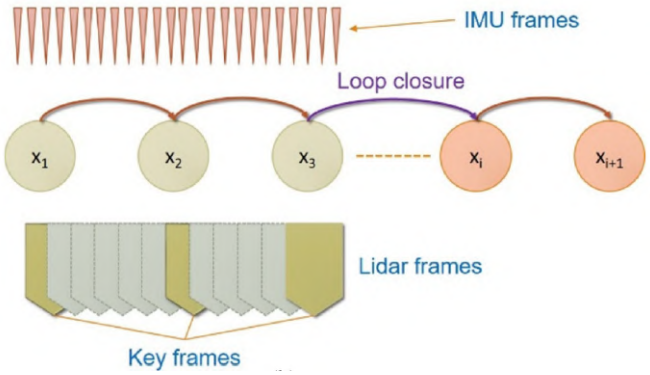


Figura 5.1: Schema della frequenza di acquisizione IMU e LiDAR [32]

```
% estraggo le misure IMU nello sweep
prevTime =time_lidar(n - skipFrames);
currTime = time_lidar(n);
index_imu = letture_IMU.tempo>=prevTime & letture_IMU.tempo<currTime;
imuAcc = letture_IMU.acc(index_imu,:);
imuGyro= letture_IMU.gyro(index_imu,:);
imuTime = letture_IMU.tempo(index_imu);
% stima dell'assetto iniziale con IMU
tform_in = preintegrationIMU(imuGyro,imuAcc,imuTime, tform, letture_IMU.bias_g,
    letture_IMU.bias_a,v0);
% Registrazione: calcolo della trasformazione relativa
tform = pcregistericp(ptCloud_DS, ptCloudPrev, 'Metric', 'planeToPlane', ...
      'InitialTransform',tform_in, 'InlierDistance',inlierDist, 'MaxIterations',
    MaxIter);
Dx_vec = tform.Translation;
Dx = norm(Dx_vec);
Dt = currTime-prevTime;
versore = [1 0 0];
v0 = versore.*(Dx/Dt);
```

Listing 5.1: Algoritmo di fusione debole con IMU

Per prima cosa si estraggono dal vettore tempo delle misurazioni LiDAR i tempi relativi alla scansione precedente e alla scansione attuale, `prevTime` e `currTime`. Si estraggono quindi gli indici del vettore tempo che corrispondo a tutti quei valori compresi tra il tempo della scansione precedente e quello della scansione attuale. Estratti gli indici si possono estrarre le misurazioni IMU effettuate in quel lasso di tempo, ricavando così `imuAcc`, `imuGyro` e `imuTime`.
Si entra quindi nella funzione `preintegrationIMU`, la quale restituisce in output una trasformazione relativa di primo tentativo, stimata tramite pre-integrazione dei dati IMU. Nel codice 5.2 viene illustrata nel dettaglio la funzione. Una volta usciti dalla funzione avremo una trasformazione di primo tentativo per la registrazione. Si può notare come in questo caso per il metodo ICP sia stato utilizzato il parametro `inlierDistance` al posto di `inlierRatio`. Questo perché, come

specificato nel *help* di Matlab, avendo una trasformazione di primo tentativo migliore, si può utilizzare *inlier distance* con un valore basso. Questo parametro cerca i punti per il calcolo della trasformazione in una sfera di raggio pari al valore del parametro. Successivamente saranno mostrati dei grafici in cui si è valutato quale valore di *inlier distance* fosse ottimale (figura 5.23a).
Per l'integrazione dell'IMU nell'iterazione successiva serve necessariamente la velocità iniziale, di conseguenza bisognerebbe salvare la velocità finale calcolata mediante integrazione e utilizzarla nell'iterazione successiva. Questo non può essere fatto perché causa problemi di deriva importanti. Si è ovviato stimando la velocità dall'iterazione precedente considerando il tempo tra i due frame e la distanza calcolata tramite la ricostruzione, allineandola poi al sistema corpo. Qui si è fatta una ipotesi importate, ovvero che non vi siano sbandate laterali. L'ipotesi è restrittiva ma va bene per le simulazioni di nostro interesse. Calcolare la velocità in questo modo abbassa i problemi di deriva dovuti all'integrazione, tuttavia, se la ricostruzione fallisce, anche l'integrazione dell'IMU non è più utilizzabile. Nello schema 5.2 è raffigurato il codice in modo intuitivo.

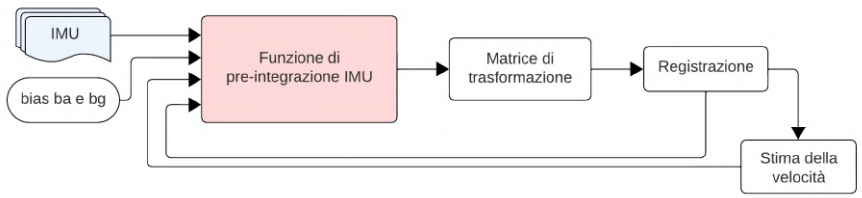


Figura 5.2: Schema del codice aggiuntivo IMU

5.1.1 Pre-integrazione

Per semplificare la scrittura del codice si è implementata una funzione Matlab esterna che preintegrasse le letture IMU.

```
%% latex
function tform_in = preintegrationIMU(imuAngVel, imuAcc, imuTime, tfrom_prev, bg
    , ba,v0)
% ipotesi di moto uniforme se non ho letture
tform_in = tfrom_prev;
g = [0,0,-9.81];
% controllo sulle letture
if isempty(imuAngVel) == false || isempty(imuAcc) == false
    % azzero vettore tempo
    imuTime = imuTime-min(imuTime);
    % angoli: rimozione bias e integrazione
    imuAngVel = imuAngVel-bg;
    cum_ang =  cumtrapz(imuTime, imuAngVel);
    % calcolo della matrice di rotazione
```

```
    R_i = eul2rotm(cum_ang, 'XYZ');
    R2 = R_i(:,:,end);

    % accelerazione: rimozione bias
    imuAcc = -(imuAcc - ba);
    % riporto tutto nel sistema locale iniziale e tolgo gravit
    acc_k = zeros(size(imuAcc));
    for j = 1:1:length(imuAcc(:,1))
        acc_k(j,:) = R_i(:,:,j)*imuAcc(j,:)' + g';
    end

    % spostamento: integrazione due volte
    vel_integral =  v0 + cumtrapz(imuTime,acc_k);
    cumLength = cumtrapz(imuTime,vel_integral);
    trasl =cumLength(end,:);

    % creazione matrice di trasformazione
    tform_in = rigidtform3d(R2,trasl);
end
end
```

Listing 5.2: Funzione di pre-integrazione letture IMU

La funzione è così concepita:

- **Input**
  - `imuAngVel`: letture giroscopio IMU
  - `imuAcc`: letture accelerometro IMU
  - `imuTime`: vettore tempo letture IMU
  - `tform_prev`: matrice di trasformazione iterazione precedente
  - `bg`: bias del giroscopio
  - `ba`: bias dell'accelerometro
  - `v0`: velocità all'inizio delle letture IMU
- **Output**
  - `tform_in`: trasformazione di primo tentativo

Se le letture IMU sono vuote, la funzione restituisce in output gli stessi valori di input, per non creare problemi al proseguimento del codice. Se invece le letture sono presenti inizia la pre-integrazione, suddivisa in 4 parti.

1. Si rimuove il *bias* del giroscopio alle letture della velocità angolare. Si procede quindi integrando la velocità angolare con il metodo dei trapezi, tramite la funzione Matlab `cumtrapez`. Si calcolano le matrici di rotazione

a partire dagli angoli di Eulero. L'ultima matrice di rotazione sarà quella di nostro interesse per la creazione della trasformazione.

2. Si rimuove il *bias* dell'accelerometro alle letture di accelerazione. Tramite le matrici di rotazione calcolate si ruota l'accelerazione nel sistema di riferimento globale e si sottrae la gravità. É bene notare che le variazioni in rollio e beccheggio sono molto piccole, quindi non è necessaria la stima dell'angolo iniziale in quanto il vettore gravità nel sistema di riferimento globale coincide con il sistema di riferimento locale. Se le variazioni di assetto fossero più pronunciate si dovrebbe stimare l'assetto per altre vie, quali per esempio il magnetometro. Le misurazioni IMU sono state fatte in un sistema ENU, quindi il vettore gravità è $\vec{g} = [0, 0, -9.81]$.

3. Si integra due volte l'accelerazione per ricavare lo spostamento. Per la velocità è necessaria la velocità iniziale, mentre per lo spostamento non serve lo spostamento iniziale in quanto si sta cercando la traslazione relativa.

4. L'ultimo passaggio è quello di costruire l'oggetto `rigidtform3d(R2,trasl)`, il quale sarà formato dall'ultima matrice di rotazione valutata `R2` e il vettore `trasl`, valutati come ultimi valori degli integrali cumulati.

In figura 5.3 un breve schema che rappresenta quanto appena detto a parole.

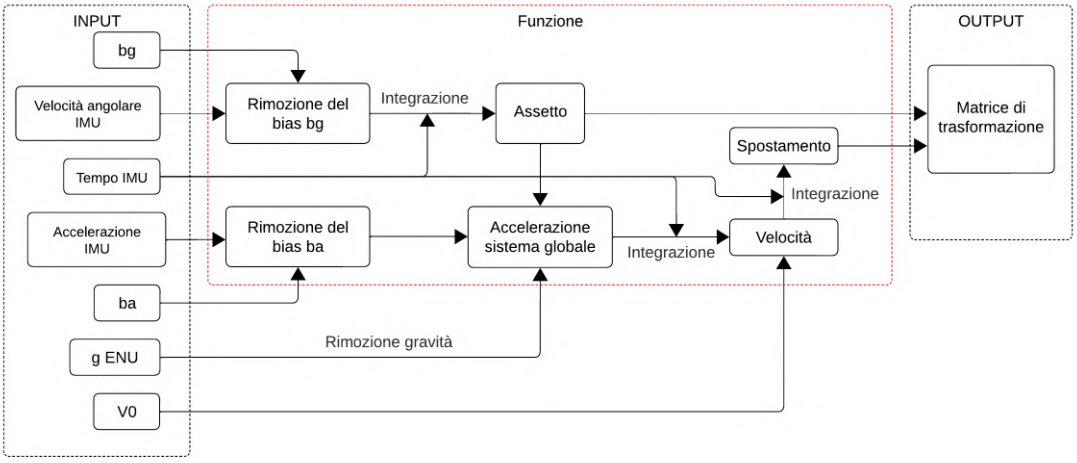


Figura 5.3: Schema della funzione di pre-integrazione letture IMU

5.1.2 Confronto metodi di pre-integrazione

La scelta del metodo adottato per la pre-integrazione dei dati IMU è stata dettata da risultati numerici e considerazioni pratiche che verranno qui spiegate. In particolare, il problema principale riscontrato durante la fase di pre-integrazione è stato quello della stima della velocità iniziale da utilizzare per il processo di integrazione della accelerazione, al fine di stimare la traslazione relativa. Sono stati provati due differenti metodi:

1. **Memorizzando la velocità in output dal processo di integrazione** e utilizzandola all'iterazione successiva. Il processo di integrazione prevede appunto il calcolo della velocità a partire dall'andamento delle accelerazioni.
2. **Stimando la velocità** a partire dai risultati del processo di registrazione.

Si sono eseguite simulazioni sulla traiettoria aperta in modo da non avere migliorie apportate dal processo di *pose graph optimization*. Di seguito verranno riportati i risultati ottenuti a confronto, in modo da avvallare la scelta fatta.

Si può subito notare dal grafico 5.4 riportante le traiettorie, che la ricostruzione con il metodo tramite integrazione e memorizzazione (da qui in poi metodo 1) ha avuto qualche problema nella ricostruzione. Nel primi due tratti la ricostruzione è tutto sommato accettabile. Tuttavia, l'accumulo degli errori di integrazione dovuti alla memorizzazione della velocità da iterazione a iterazione, causa ad un certo istante una ricostruzione errata del metodo. Ciò è da imputare ad una non adeguata trasformazione di primo tentativo. Per quanto riguarda il metodo con stima della velocità da ricostruzione (da qui in poi metodo 2), esso rimane consistente per tutta la traiettoria, ottenendo risultati migliori (si veda anche tabella 5.7).

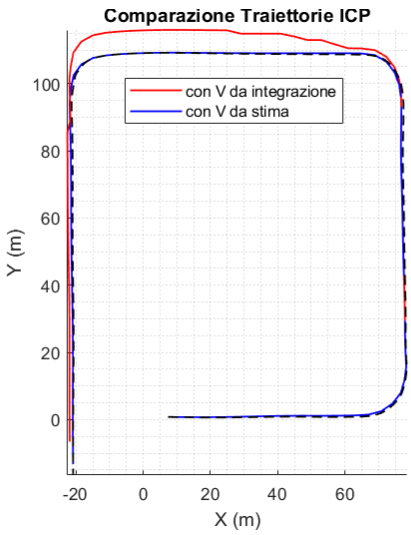
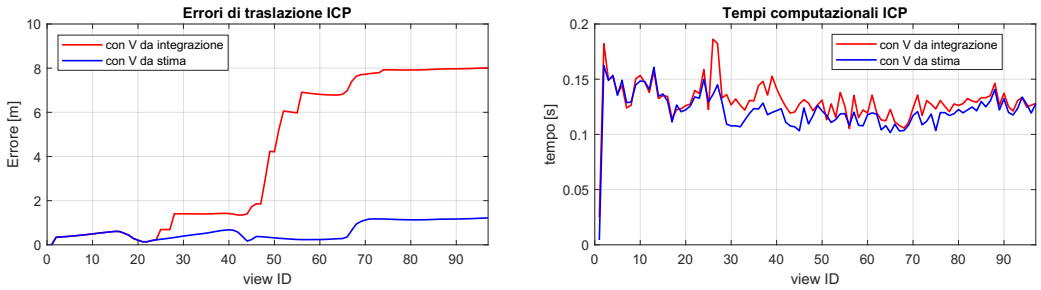


Figura 5.4: Traiettoria aperta metodo ICP assistito da IMU con i due metodi di calcolo della velocità iniziale



(a) Andamento degli errori di traslazione per i due metodi sviluppati

(b) Andamento dei tempi computazionali PTPF per i due metodi sviluppati

Figura 5.5: Risultati metodi di stima velocità per integrazione IMU (traiettoria aperta metodi ICP)

Se si osservano gli andamenti degli errori di traslazione in figura 5.5a si potranno notare nette differenze tra metodo 1 e metodo 2 associate ai frame in cui la registrazione non è andata a buon fine. Inoltre, per quanto riguarda i tempi computazionali del metodo 1 si può osservare in figura 5.5b come essi siano tendenzialmente maggiori rispetto al metodo 2 (evidenziato anche in tabella 5.7). Anche questo è causato da una peggior trasformazione di primo tentativo, la quale comporta un maggior tempo di registrazione del metodo ICP.
Tuttavia, se si osservano solo i risultati dopo l'algoritmo di registrazione, non si potrà essere ragionevolmente certi della bontà del metodo. Per confermare definitivamente il metodo 2 si sono osservate le differenze tra matrice di trasformazione stimata tramite IMU e matrice di trasformazione ottenuta in seguito alla registrazione. Il parametro RPE (*Relative Pose Error*) è stato calcolato in modo simile all'articolo [10]:

$$RPE_k = \left\| \left(T_{ICP,k}^{-1} T_{IMU,k} - I^{4\times4} \right) \right\| \tag{5.1}$$

Dove $T_{ICP,k}$ è la matrice di trasformazione relativa calcolata mediante registrazione, $T_{IMU,k}$ è la matrice di trasformazione stimata con la pre-integrazione dell'IMU e $I^{4\times4}$ è la matrice identità. La formula è così costituita perché, ipotizzando che le due trasformazioni siano identiche, il risultato di $T^{-1}T$ sarebbe la matrice identità. Sottraendo la matrice identità ed estraendone la norma si può ottenere una stima della differenza tra le due trasformazioni. Una differenza bassa implica che la pre-integrazione dell'IMU ha prodotto una buona trasformazione di primo tentativo. In figura 5.6 sono riportati gli andamenti delle differenze tra le matrici di trasformazione stimate e le matrici di trasformazione calcolate con la registrazione.

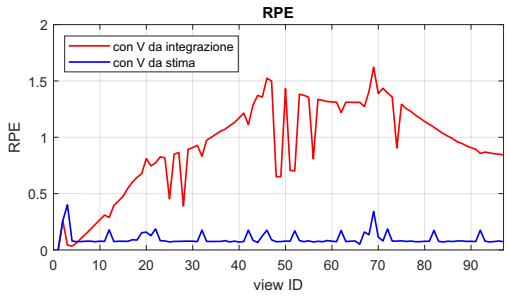


Figura 5.6: Andamenti RPE a confronto

Param.	Metodo 1	Metodo 2
Err rms	3.052 m	0.418 m
Tempi	12.62 s	11.91 s
Tempi ICP	0.122 s	0.116 s
Tempi IMU	6.2E-04 s	5.2E-04 s
RPE medio	0.927	0.092

Figura 5.7: Risultati numerici

Il grafico 5.6 evidenzia bene come la stima fatta con il metodo 1 sia notevolmente peggiore rispetto alla stima fatta con il metodo 2. Ciò conferma la scelta fatta in questa tesi di appoggiarsi ad una stima a posteriori della velocità da utilizzare all'iterazione successiva, invece che memorizzare la velocità dall'integrazione, avendo così problemi di deriva della stima. Un'ulteriore conferma viene dai risultati numerici in tabella 5.7.

5.2 Risultati ICP e LOAM con IMU a confronto

Si sono eseguite le simulazioni presentate precedentemente introducendo la preintegrazione dell'IMU. In questa sezione verranno presentati i risultati, mostrando le soluzioni tramite metodo ICP e LOAM map. Si è scelto di mostrare solo la traiettoria chiusa a singolo loop, la traiettoria aperta e la traiettoria a otto piccolo. Nella traiettoria con variazione di quota è subentrato il medesimo problema del metodo LOAM dovuto a carenza di *feature*. Ricordiamo che il metodo qui illustrato non è un metodo fortemente accoppiato, di conseguenza si affida ancora completamente agli algoritmi di ricostruzione, senza dare un peso differente ai due metodi come potrebbe fare un filtro di Kalman. Di conseguenza il fallimento della ricostruzione comporta in ogni caso la perdita dell'intera simulazione. Per prima cosa verranno riportati i risultati ottenuti e in seguito verranno riportati degli studi di sensibilità ai vari parametri per mostrare come l'introduzione dell'IMU abbia modificato la risposta a quest'ultimi. Per semplicità, di seguito si farà riferimento a LOAM intendendo LOAM map.
I parametri utilizzati sono i medesimi delle rispettive simulazioni, eccetto i parametri riportati in tabella 5.1.

Parametro	Settaggio
`inlierDist`	0.4
`ba`	[0.4905,0.4905,0.4905]
`bg`	[0.0873,0.0873,0.0873]

Tabella 5.1: Parametri aggiuntivi metodi assistiti da IMU

5.2.1 Traiettoria chiusa

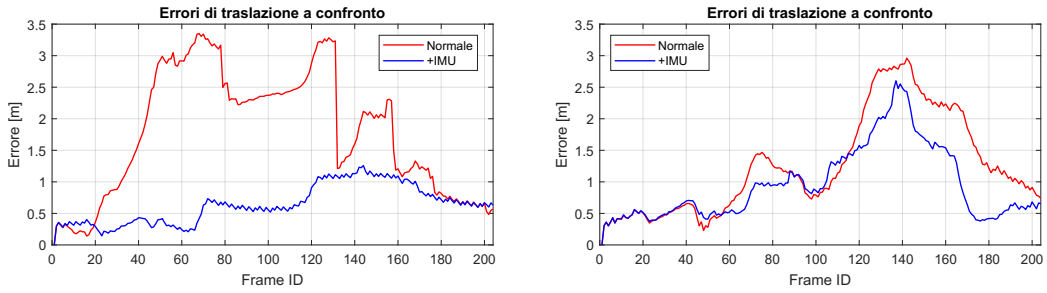
La simulazione è stata effettuata sulla traiettoria chiusa con specifiche riportate in tabella 4.2. In tabella 5.2 vengono riportati i risultati numerici ottenuti. Il tempo computazionale dell'IMU è dell'ordine di 10^{-4} ed è stato incluso nel tempo della registrazione.

Parametro	ICP IMU	ICP	%	LOAM IMU	LOAM	%
Err rms pre pgo [m]	0.426	2.046	-79 %	0.645	0.856	-25 %
Err rms pgo [m]	0.409	1.169	-65 %	0.645	0.856	-25 %
Err rms %	0.05	0.14	-65 %	0.08	0.11	-25 %
Err max	1.258	3.351	-62 %	2.602	2.960	-12 %
Err ang rms [°]	0.83	1.31	-37 %	0.96	1.37	-30 %
Err ang max [°]	3.47	4.56	-24 %	3.63	4.36	-17 %
Tempo [s]	28.05	26.62	5 %	42.76	52.06	-18 %
PTPF [s]	0.138	0.131	5 %	0.210	0.255	-18 %
T ICP/LOAM [s]	0.125	0.119	5 %	0.085	0.099	-14 %
T LOAM map [s]	-	-	-	0.089	0.113	-21 %

Tabella 5.2: Risultati ICP e LOAM assistiti da IMU per traiettoria chiusa

Si può notare come per il caso ICP gli errori siano diminuiti, ma i tempi computazionali aumentati. Gli errori sono diminuiti probabilmente per il fatto che, avendo la possibilità di garantire una trasformazione di primo tentativo più fedele, si è potuto utilizzare il parametro *inlier distance* con una sfera di ricerca più piccola. Di conseguenza, a parità di tempi computazionali è stato raggiunto un miglior risultato. Come si vedrà in seguito, avendo utilizzato algoritmi senza IMU con parametri molto prossimi ai valori ottimali, i risultati in termini di tempi ed errori erano già piuttosto buoni. Per quanto riguarda l'algoritmo LOAM, sia i tempi computazionali che gli errori sono diminuiti notevolmente, a dimostrazione del fatto che l'algoritmo LOAM necessita maggiormente di una trasformazione di primo tentativo migliore. In figura 5.8 sono riportati gli errori di traslazione a confronto per i due metodi. In figura 5.8a si nota bene come gli errori di trasla-

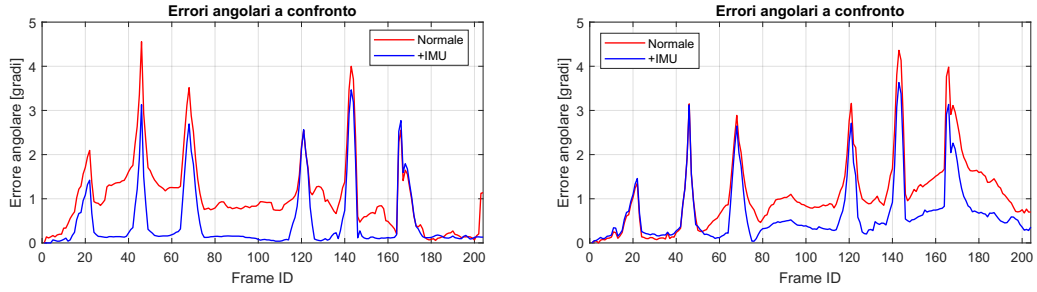
zione del metodo ICP assistito da IMU siano inferiori. Questa miglioria risulta meno evidente ma comunque presente in figura 5.8b per il metodo LOAM.



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.8: Confronto errori di traslazione con e senza IMU traiettoria chiusa

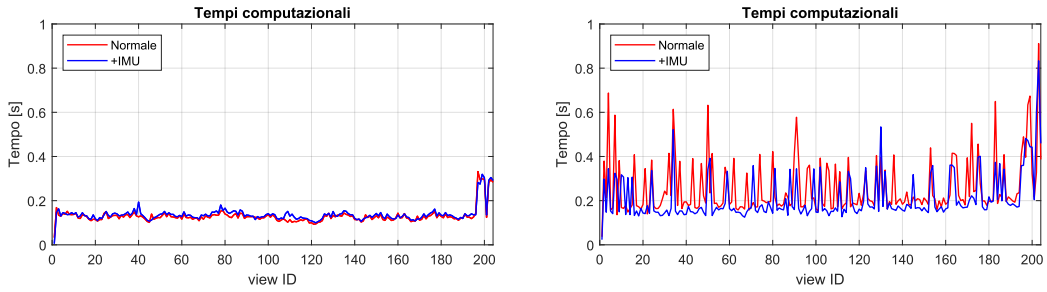
Andamenti analoghi possono essere riscontrarti per gli errori angolari. In figura 5.9 si nota come gli errori presentino i medesimi picchi, corrispondenti alle zone di curva, ma che la media complessiva sia inferiore, a prova del fatto che l'IMU ha migliorato la soluzione.



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.9: Confronto errori angolari con e senza IMU traiettoria chiusa

In ultima analisi, si possono osservare i tempi computazionali per il metodo ICP in figura 5.10a, dove gli andamenti per il metodo IMU sono pressoché analoghi. Se si osserva attentamente in figura 5.10b si potrà notare come i picchi associati a iterazioni in cui l'algoritmo LOAM ha faticato sono stati in parte ridotti. Complessivamente quindi, come mostrato in tabella 5.2, i tempi computazionali di simulazione diminuiscono.



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.10: Confronto tempi computazionali con e senza IMU traiettoria chiusa

5.2.2 Traiettoria aperta

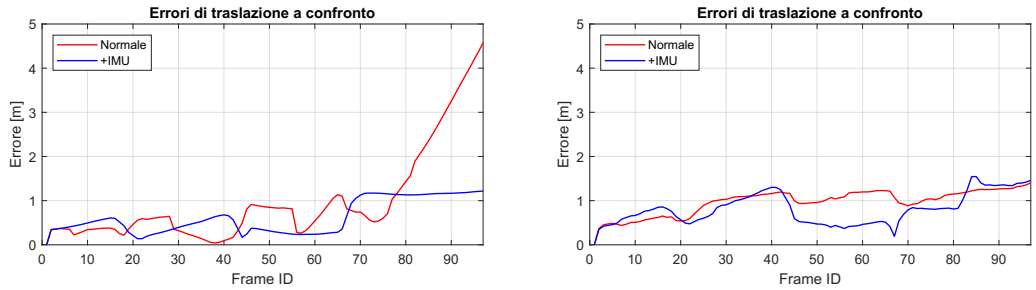
Si sono effettuate le simulazioni sulla traiettoria aperta con riferimento alle specifiche in tabella 4.5. In tabella 5.3 vengono riportati i risultati ottenuti a confronto.

Parametro	ICP IMU	ICP	%	LOAM IMU	LOAM	%
Err rms [m]	0.418	0.843	-50 %	0.507	0.584	-13 %
Err rms %	0.11	0.22	-50 %	0.13	0.15	-13 %
Err max	1.219	4.574	-73 %	1.546	1.404	10 %
Err ang rms [°]	1.55	2.12	-27 %	1.57	1.61	-3 %
Err ang max [°]	10.45	8.55	22 %	10.31	10.84	-5 %
Tempo [s]	11.53	11.15	3 %	16.71	21.48	-22 %
PTPF [s]	0.119	0.115	3 %	0.172	0.221	-22 %
T ICP/LOAM [s]	0.112	0.109	3 %	0.074	0.100	-26 %
T LOAM map [s]	-	-	-	0.068	0.083	-18 %

Tabella 5.3: Risultati ICP e LOAM assistiti da IMU per traiettoria aperta

Similmente a quanto accaduto per la traiettoria chiusa mostrata precedentemente, anche in questo caso per il metodo ICP si ha un miglioramento della accuratezza perdendo però in tempo computazionale, anche se in minima percentuale. Anche il metodo LOAM si comporta in modo simile a quanto effettuato nella simulazione della traiettoria chiusa. In questo caso i valori di *rms* di traslazione e angolari vedono una diminuzione, sebbene alcuni valori di picco subiscano degli incrementi. In figura 5.11 si nota come anche in questo caso, ad ulteriore conferma dei dati riportati in tabella 5.3, i valori di traslazione risultino complessivamente inferiori ai valori con il metodo senza pre-integrazione dell'IMU.

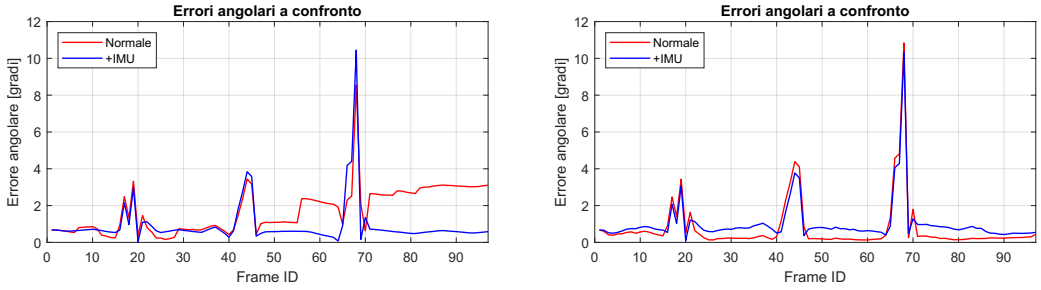
É interessante notare nel grafico 5.11a come la forte deriva presente nel metodo ICP senza IMU sia stata ridotta notevolmente dall'introduzione dell'IMU. Anche



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.11: Confronto errori di traslazione con e senza IMU traiettoria aperta

gli errori di traslazione nel caso di LOAM con IMU sono stati ridotti, anche se in modo meno importate. Bisogna considerare che il metodo LOAM con l'utilizzo di LOAM map presenta il passaggio ulteriore di registrazione nella mappa complessiva proprio per avere un ulteriore controllo sulla deriva. L'introduzione dell'IMU di conseguenza non apporta grandi cambiamenti in termini di deriva ma favorisce i tempi computazionali.

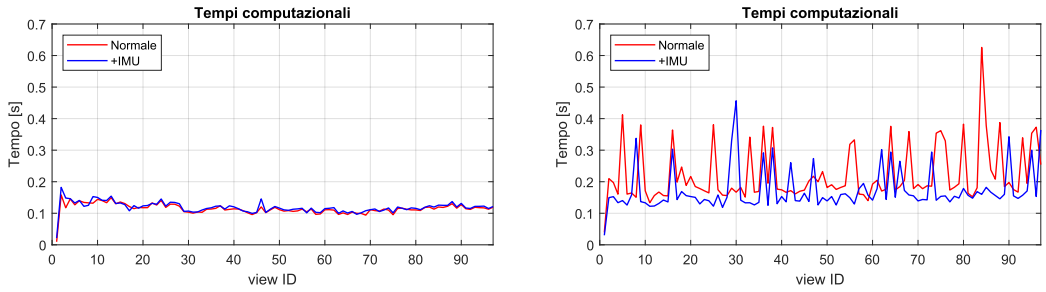


(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.12: Confronto errori angolari con e senza IMU traiettoria aperta

In figura 5.12 vengono riportati gli andamenti degli errori angolari. Per quanto riguarda il metodo ICP si nota bene come esso sia riuscito a contenere l'errore angolare all'iterazione 70. Come si nota nel grafico 5.12a infatti, l'errore angolare relativo al metodo IMU presenta un picco non compensato da iterazioni successive, come invece avviene per altri picchi precedenti. Il metodo LOAM con IMU in questo caso risulta essere leggermente peggiorativo in termini di errori angolari, anche se gli andamenti sono confrontabili e i picchi hanno ridotto la loro intensità.

Per quanto riguarda i tempi computazionali, vale quanto mostrato per la traiettoria chiusa. I metodi ICP presentano andamenti del tutto confrontabili tra loro (figura 5.13a), mentre i metodi LOAM presentano una riduzione dei picchi associati a iterazioni molto onerose computazionalmente, con conseguente riduzione



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.13: Confronto tempi computazionali con e senza IMU traiettoria aperta

del tempo computazionale complessivo. A favorirne maggiormente, come si osserva nella tabella 5.3, è il tempo computazionale relativo alla ricostruzione LOAM. Questo significa che la ricostruzione LOAM è favorita da una trasformazione di primo tentativo migliore, mentre la registrazione nella LOAM map dipende meno dal primo tentativo, quanto più invece dalla bontà della ricostruzione LOAM stessa.

5.2.3 Traiettoria doppio loop piccolo

Si è simulata la traiettoria a doppio loop (otto) piccolo, le cui specifiche sono riportate in tabella 4.9. I risultati ottenuti sono riportati in tabella 5.4.

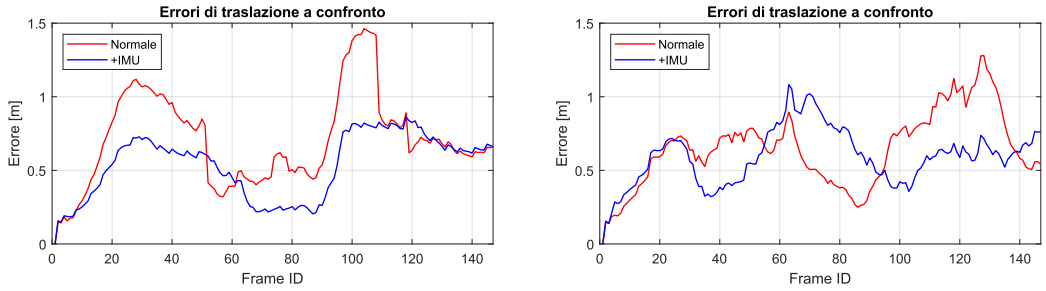
Parametro	ICP IMU	ICP	%	LOAM IMU	LOAM	%
Err rms pre pgo [m]	0.335	0.681	-51 %	0.377	0.407	-7 %
Err rms pgo [m]	0.334	0.447	-25 %	0.360	0.402	-10 %
Err rms %	0.11	0.15	-25 %	0.12	0.14	-10 %
Err max	0.863	1.463	-41 %	1.082	1.281	-15 %
Err ang rms [°]	1.70	1.93	-12 %	1.96	1.85	6 %
Err ang max [°]	7.19	7.21	0 %	6.67	7.04	-5 %
Tempo [s]	21.76	21.43	2 %	36.10	44.37	-19 %
PTPF [s]	0.148	0.146	1 %	0.246	0.302	-19 %
T ICP/LOAM [s]	0.12	0.118	2 %	0.093	0.114	-18 %
T LOAM map [s]	-	-	-	0.087	0.117	-26 %

Tabella 5.4: Risultati ICP e LOAM assistiti da IMU per traiettoria otto piccolo

Se si confrontano i risultati in tabella 5.2 e 5.3 con quanto ottenuto in questo caso (tabella 5.4), si potranno notare delle somiglianze. Questo suggerisce un andamento complessivo delle simulazioni che indica come in generale l'accuratezza in termini di *rms* tendenzialmente aumenti, mentre l'accuratezza angolare non

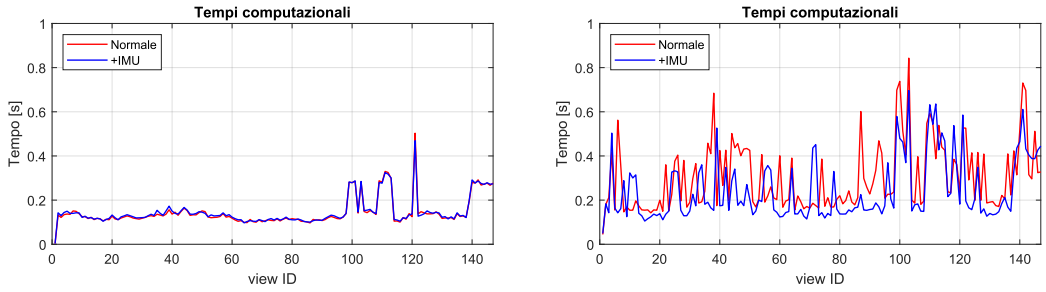
apporti migliorie sostanziali. Per quanto riguarda invece i tempi computazionali, nel metodo ICP con IMU essi tenderanno ad aumentare leggermente, mentre nel caso LOAM con IMU tenderanno a diminuire.

Per completezza vengono riportati di seguito i risultati grafici ottenuti per gli errori di errori di traslazione (figura 5.14) e per i tempi computazionali (figura 5.15). Gli andamenti sono confrontabili con quanto ottenuto precedentemente.



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.14: Confronto errori di traslazione con e senza IMU traiettoria otto piccolo



(a) Metodo ICP **(b)** Metodo LOAM

Figura 5.15: Confronto tempi computazionali con e senza IMU traiettoria otto piccolo

5.3 Studio di sensibilità sui vari parametri

Sono state effettuate un gran numero di simulazioni variando alcuni parametri, in modo da confrontare simulazioni in diverse condizioni e capire l'influenza di tali parametri. In particolar modo si andranno ad osservare:

- La risposta ai **frame saltati**, per capire se esista o meno un punto di ottimo, e capire la risposta dei vari metodi ad un alto numero di frame saltati. Le simulazioni sono state eseguite per:
  - ICP con e senza IMU
  - LOAM map con e senza IMU

- L'influenza dell'***inlier distance*** per quanto riguarda il metodo ICP assistito da IMU, al fine di spiegare il perché della scelta del parametro riportato in tabella 5.1.
- L'influenza dell'***inlier ratio***, in questo caso solo per il metodo ICP puro, in quanto questo parametro è il corrispettivo dell'*inlier distance* nel metodo ICP con IMU.
- L'influenza della ***downsample percentage*** per quanto riguarda i metodi che presentano sotto-campionamento, ovvero ICP con e senza IMU, al fine di capirne la risposta anche in caso di presenza dell'IMU.
- Influenza del ***grid step*** nel caso LOAM map e LOAM map assistito da IMU. Questo parametro è il corrispettivo del *downsample percentage* per i metodi ICP.
- L'influenza della dimensione di voxel (***voxel size***) sulla ricostruzione nel caso di LOAM map e LOAM map assistita da IMU.
- Il parametro ***maxPlanarSurfacePoint***, in modo da completare i parametri più importanti necessari al *setting* delle simulazioni.

Per le seguenti simulazioni si è scelto di studiare la sola traiettoria aperta, in quanto non presenta chiusure del loop che possano non evidenziare correttamente la capacità e i tempi computazionali dei vari metodi. Inoltre, vengono utilizzati i parametri settati in precedenza per tale traiettoria eccetto il parametro studiato. Al fine di comprendere meglio quale simulazione sia la più efficiente si è introdotto un parametro di efficienza fittizio, definito nel modo seguente

$$E = \frac{1}{err_{trasl} \cdot err_{ang} \cdot \Delta t_{icp}} \quad , \quad E = \frac{1}{err_{trasl} \cdot err_{ang} \cdot (\Delta t_{loam} + \Delta t_{map})} \tag{5.2}$$

$$E_{tot} = \frac{1}{err_{trasl} \cdot err_{ang} \cdot \Delta T_{tot}} \tag{5.3}$$

dove:

- err_{trasl} è l'errore di traslazione (di *rms*)
- err_{ang} è l'errore angolare (di *rms*)
- Δt_{icp}, Δt_{loam}, Δt_{map} sono rispettivamente i tempi medi associati alla registrazione ICP, registrazione LOAM e LOAM map.
- ΔT_{tot} è il tempo computazionale totale.

I valori così calcolati verranno divisi poi per il valore massimo, ottenendo quindi valori adimensionali confrontabili. Questo parametro è stato scelto in quanto l'obiettivo della SLAM è ottenere una simulazione con basso errore di traslazione, con errore angolare contenuto e al contempo una simulazione veloce. Di conseguenza, si avrà un'efficienza alta nel caso in cui si abbiano sia errori angolari, che errori di traslazione, che tempi computazionali bassi. L'efficienza è solo un parametro che facilita la lettura di questi tre parametri in contemporanea. Da notare che, nei valori di nostro interesse, i tre parametri hanno ordine di grandezza comparabile (metro, secondo e grado), altrimenti la formula avrebbe dato un peso maggiore alla quantità con ordine di grandezza inferiore. Il valore del parametro di ottimo di uno dei tre non è detto che coincida con il valore di ottimo complessivo. Questo parametro è da considerarsi solo a scopo illustrativo.

La differenza tra E (equazione 5.2) ed E_{tot} (equazione 5.3) sta nel fatto di considerare come parametro importante il tempo medio di singola iterazione, e quindi osservare se l'algoritmo di ricostruzione sia più o meno veloce, o considerare il tempo totale di simulazione, indipendentemente da quanto la singola ricostruzione sia efficiente. E_{tot} dipenderà quindi dai frame saltati, permettendo di confrontare il tempo di simulazione totale.

5.3.1 Risposta ai frame saltati

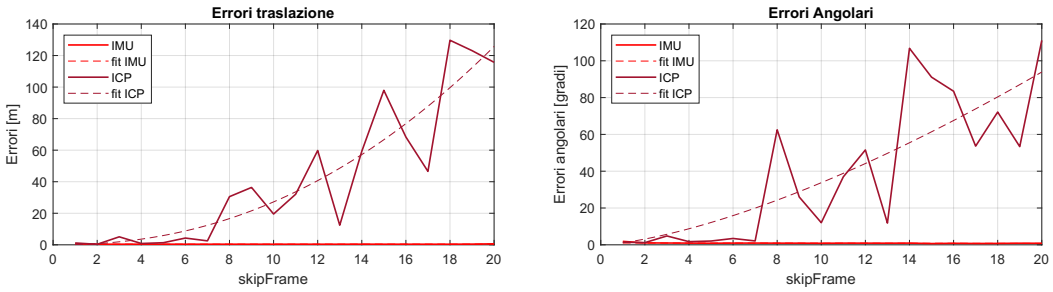
Si è scelto di far variare il parametro `skipFrames` in un intervallo tale:

$$skipFrames = [1, 2, 3, ..., 20] \tag{5.4}$$

Si consideri che 20 frame saltati implicano, con una frequenza di acquisizione di 10 Hz e una velocità media di 10 m/s, circa 20 metri di differenza tra un frame accettato e il successivo. Di conseguenza ci si aspetta che la registrazione non sia del tutto ottimale.

ICP

Per quanto riguarda gli errori di traslazione e gli errori angolari, i risultati ottenuti eseguendo 20 simulazioni per ICP e ICP assistita da IMU sono riportati in figura 5.16.



(a) Errori di traslazione (di *rms*) **(b)** Errori angolari (di *rms*)

Figura 5.16: Errori in funzione dei frame saltati

Si può notare come dopo 7 frame saltati si incorre in divergenza del metodo ICP puro. Gli errori del metodo ICP assistito da IMU sono appena visibili, a dimostrazione del fatto che il metodo garantisce ancora la convergenza. Questi risultati ci fanno capire bene quanto l'IMU stabilizzi il metodo, permettendo di utilizzare molti freme saltati. Questo è possibile in quanto l'IMU garantisce una buona trasformazione di primo tentativo, la quale permette di utilizzare il parametro *inlier distance* per la ricerca dei punti.

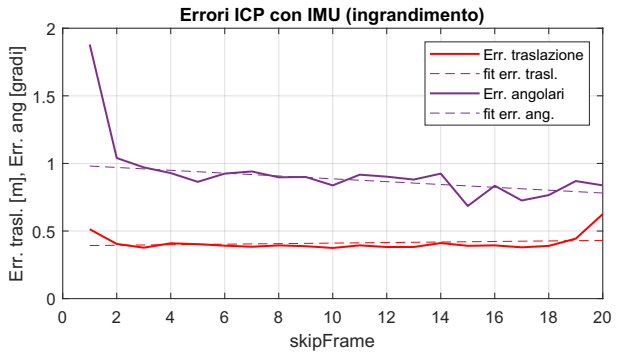
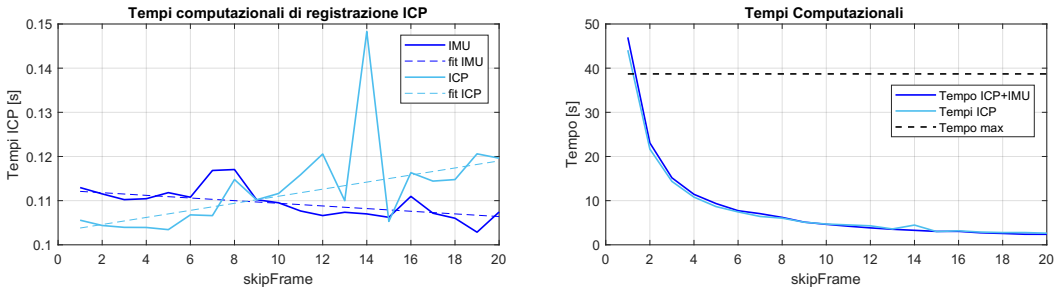


Figura 5.17: Zoom errori traslazione e angolari metodo ICP con IMU

In figura 5.17 sono riportati gli andamenti ingranditi degli errori angolari e degli errori di traslazione per il solo ICP con IMU. Si può notare come negli ultimi valori di frame saltati l'errore di traslazione aumenti, di conseguenza ci si aspetta nei frame successivi una possibile divergenza. É inoltre interessante notare che avere un solo frame saltato non necessariamente migliora la ricostruzione.

Risulta molto interessante osservare come nel grafico 5.18a i tempi computazionali del metodo ICP tendano ad essere discontinui o comunque aumentare nel caso di incremento dei frame saltati, mentre in controtendenza, i tempi computazionali del metodo ICP assistito da IMU tendano a diminuire. Nel grafico 5.18b invece, come ci si aspettava, i tempi computazionali complessivi diminuiscono



(a) Tempi medi per la sola registrazione

(b) Tempi totali di simulazione

Figura 5.18: Tempi computazionali in funzione dei frame saltati

all'aumentare dei frame saltati. La cosa interessante sta tuttavia nell'andamento inversamente proporzionale. In figura 5.19 è mostrata l'interpolazione dei dati di tempo di simulazione in funzione dei frame saltati, con intervallo di confidenza al 99%.

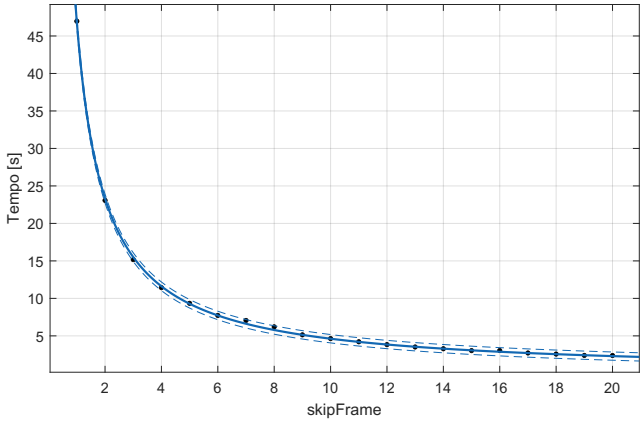


Figura 5.19: Interpolazione dei valori di tempo computazionale totali con intervalli di confidenza al 99%

La funzione interpolante è riportata nell'equazione 5.5, dove 46.82 è il tempo in secondi nel caso di skipFrames = 1, ovvero studiando tutti i frame a disposizione.

$$T_{comput}(SF) = 46.82 \cdot \frac{1}{SF} \tag{5.5}$$

Complessivamente si possono mostrare gli andamenti dell'efficienza definita come nelle equazioni 5.2 e 5.3. In figura 5.20 vengono riportati gli andamenti ottenuti.

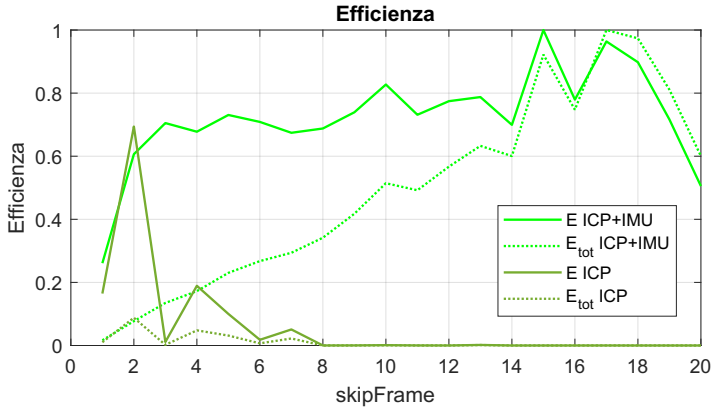
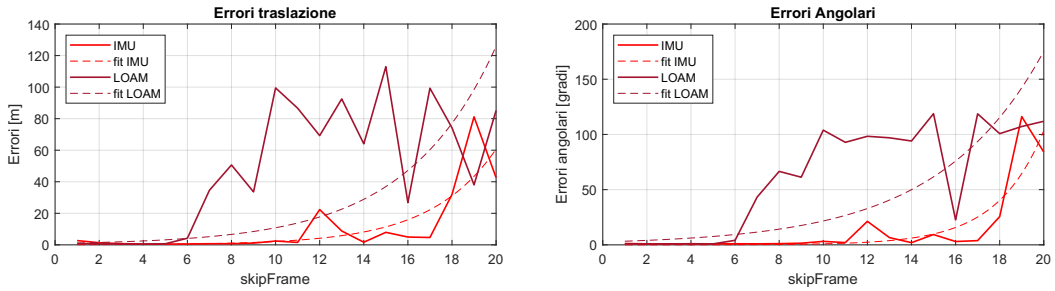


Figura 5.20: Efficienza in funzione dei frame saltati

Se si osservano i grafici 5.16 e 5.18 si potrà notare nel caso ICP puro come a 2 frame saltati si abbiano bassi errori angolari e di traslazione e tempi computazionali contenuti. A causa della divergenza del metodo dopo 7 frame saltati l'efficienza è nulla. Questo non è vero se si considera invece ICP assistito da IMU. In questo caso l'ottimo sarà spostato più avanti attorno a 15 frame saltati. Questo perché, come si nota dai singoli grafici 5.16 e 5.18, in questo caso gli errori angolari e di traslazione sono bassi anche ad alti frame. Se si osserva l'efficienza totale si vedrà come in realtà il tempo computazionale a bassi frame è molto alto e di conseguenza, anche se il tempo ICP è più basso, complessivamente la simulazione impiegherà più tempo.

LOAM

Per quanto riguarda le simulazioni con algoritmo LOAM le cose cambiano leggermente.



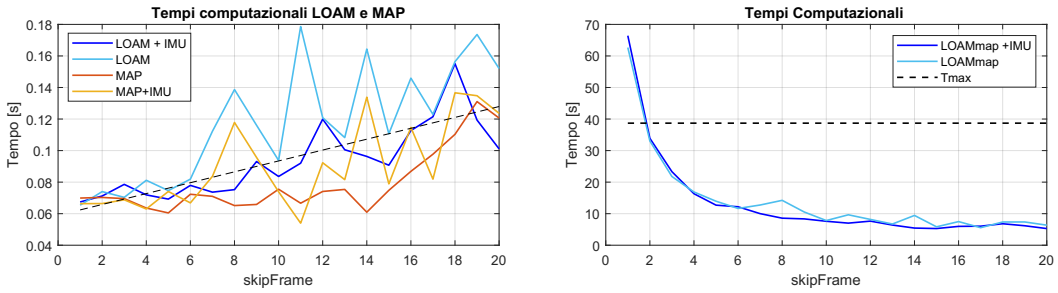
(a) Errori di traslazione (di *rms*)

(b) Errori angolari (di *rms*)

Figura 5.21: Errori in funzione dei frame saltati

Dalle figure 5.21 si può notare come gli errori complessivi del metodo LOAM siano tendenzialmente maggiori del metodo LOAM assistito da IMU. Inoltre, se

si osserva il punto di divergenza, si potrà notare come esso si sia spostato a frame saltati maggiori per il metodo con IMU (11 a fronte di 6). Questo significa che utilizzando l'IMU a bassi frame si potrebbero avere errori comparabili all'LOAM puro, ma che aumentando i frame saltati l'IMU garantirà maggior stabilità al metodo.



(a) Tempi di registrazione LOAM e map

(b) Tempi computazionali totali

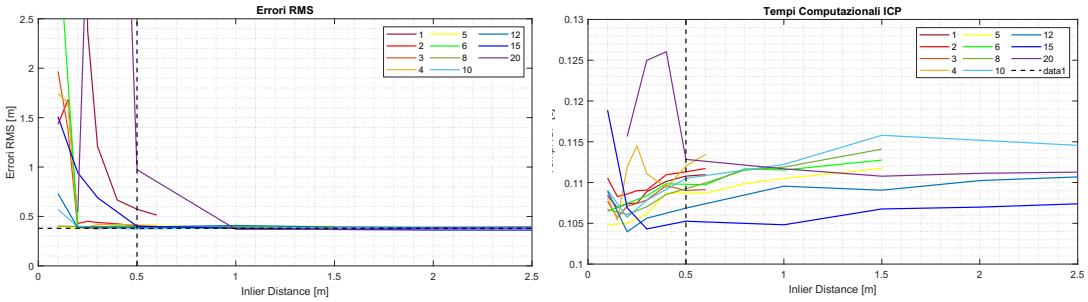
Figura 5.22: Tempi computazionali in funzione dei frame saltati

Dai tempi computazionali si può notare come il metodo LOAM assistito da IMU sia tendenzialmente migliore nella parte di registrazione LOAM. Globalmente, aumentando i frame saltati, il tempo computazionale per iterazione aumenta, ma il tempo computazionale complessivo diminuisce. Si potrà inoltre notare come essi siano più stabili (si vedano 8, 11 e 14 frame saltati).

5.3.2 Inlier distance e inlier ratio

Risposta a inlier distance per ICP IMU

Si sono eseguite un gran numero di simulazioni facendo variare sia i *frame* saltati che l'*inlier distance*. Questo parametro definisce la massima distanza tra due punti corrispondenti delle nuvole da registrare per essere considerati *inlier* dall'algoritmo.
Si sono quindi generati i grafici 5.23a e 5.23b. In figura 5.23a si nota come, oltre un certo valore (pari a 0.5 m), tutte le simulazioni mantengano errori di *rms* contenuti. Per le simulazioni effettuate precedentemente si è scelto un *inlier distance* di 0.4 m, questo perché, se si osservano le curve con $skipFrame = 4$, si potrà notare come si sia lontani dal punto in cui le simulazioni ottengono risultati scarsi ($inlier\ distance = 0.2$).



(a) Errori di *rms* (di traslazione)

(b) Tempi computazionali

Figura 5.23: Tolleranza alla variazione di *inlier distance*, parametrizzato con i frame saltati

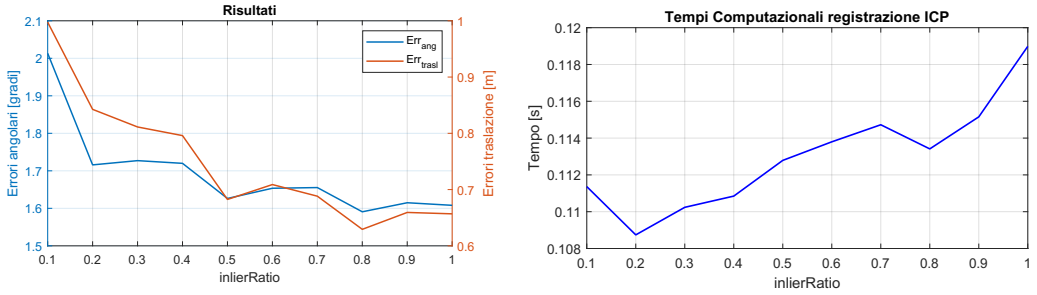
Si può inoltre notare come ad alti frame saltati, l'*inlier distance* che garantisce una buona ricostruzione aumenta (ad esempio per 20 frame saltati l'*inlier distance* aumenta a 1). Questo è dovuto al fatto che la trasformazione di primo tentativo peggiora a causa di errori di integrazione, quindi per la convergenza è necessario aumentare la sfera di ricerca dei punti. In ogni caso, aumentando l'*inlier distance*, tutte le simulazioni convergono allo stesso valore di *rms*.
Non si è scelto di tenere *inlier distance* alto (pari a 1 per esempio) perché se si osservano i tempi computazionali in figura 5.23b, si potrà notare come la tendenza sia quella di aumentare il tempo computazionale all'aumentare del parametro. Questo è tutto sommato intuitivo in quanto, se il parametro aumenta, aumenta anche la sfera di ricerca dei punti, e quindi il tempo computazionale di registrazione. Si è scelto quindi un valore che garantisse buoni risultati sia in termini di tempi che di *rms*, mantenendo una certa distanza dal punto di gomito in 0.2, che garantisse tempi computazionali inferiori ma anche stabilità.

Risposta a inlier ratio per ICP

Le simulazioni effettuate in questo caso sono servite per definire il parametro di *inlier ratio* nelle simulazioni ICP. Questo parametro è il corrispettivo del parametro *inlier distance* studiato in precedenza. In particolare, mentre prima l'esclusione veniva fatta in termini di sfera di ricerca, in questo caso la funzione considera una coppia di punti *inlier* (quindi accettata) se la distanza euclidea tra questi due punti è inferiore rispetto alla percentuale specificata della distanza euclidea massima tra due punti correlati. Le simulazioni sono state fatte variando il parametro di *inlier ratio* nel range riportato in equazione 5.6 e mantenendo tutti gli altri parametri come nelle simulazioni ICP pure (quindi $skipFrame = 4$).

$$inlierRatio = [0.1, 0.2, 0.3, ..., 1] \tag{5.6}$$

I risultati ottenuti sono presentati in figura 5.24.



(a) Errori di traslazione e angolari (di *rms*)

(b) Tempi computazionali ICP

Figura 5.24: Risultati in funzione dell'*inlier ratio*

Si nota subito come se si accettano tutti gli *inlier* (quindi con $inlierRatio = 1$) gli errori di traslazione e angolari diminuiscono, facendo aumentare i tempi computazionali. Se invece si accettano pochi *inlier* (ad esempio $inlierRatio = 0.2$) i tempi computazionali diminuisco, ma aumentano gli errori. Nelle simulazioni di questa tesi per il metodo ICP si è scelto di privilegiare il tempo computazionale, scegliendo un *inlier ratio* di 0.2. É da notare comunque che la variazione dei valori è davvero lieve, e che quindi questo parametro è meno decisivo rispetto ad altri. Se si osservano i valori dei tempi computazionali di registrazione, e si considera che i frame saltati sono sempre gli stessi, si capisce bene che i tempi computazionali totali sono pressoché costanti.

5.3.3 Parametri di sotto-campionamento

Una delle operazioni più importanti nel pre-processo è quella di ridurre la nuvola di punti prima di effettuare la registrazione. Di seguito vengono osservati quattro parametri importanti che riducono, in un modo o nell'altro i punti studiati.

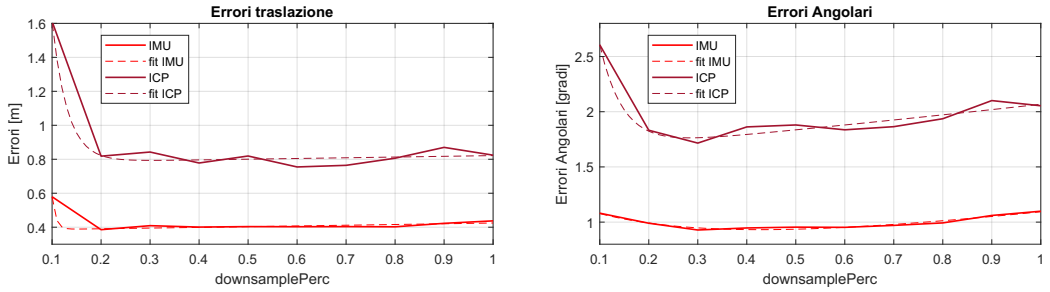
Risposta al downsampling per ICP

Viene qui studiata l'influenza della percentuale di *downsampling* sulle simulazioni che concernono il metodo di registrazione ICP. Si ricorda che in questa tesi si è scelto di utilizzare un sotto-campionamento di tipo random perché garantisce tempi computazionali migliori. Il sotto-campionamento randomico prevede la riduzione della nuvola di punti tramite eliminazione casuale degli stessi. Si sarebbe

potuto effettuare anche un sotto-campionamento di *grid step*, ovvero utilizzando un parametro di griglia che fondesse in un unico punto i punti in essa contenuti, posizionandolo nel suo baricentro. Questa operazione è più onerosa dal punto di vista computazionale, ma mantiene meglio la forma della nube.
Si sono effettuate delle simulazioni mantenendo tutti i parametri utilizzati per le simulazioni ICP e ICP assistito da IMU, eccezion fatta del parametro di *downsamplig*, fatto variare in:

$$DownsamplePerc = [0.1, 0.2, 0.3, ..., 1] \tag{5.7}$$

I risultati ottenuti sono riportati nella figura 5.25.

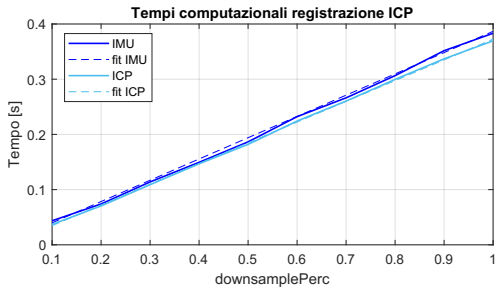


(a) Errori di traslazione (di *rms*)

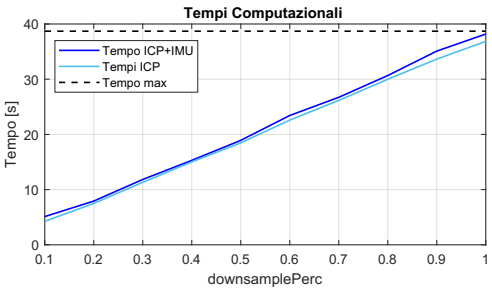
(b) Errori angolari (di *rms*)

Figura 5.25: Errori in funzione della percentuale di downsample

Dalle figure 5.25a e 5.25b si può notare come l'aumento della percentuale di *downsamplig* non apporti significativi miglioramenti alla ricostruzione. Al contrario invece, una riduzione drastica dei punti osservati (quindi con sotto-campionamento elevato) comporta un incremento degli errori di ricostruzione. Si nota inoltre come l'andamento complessivo di errori di traslazione ed errori angolari siano confrontabili. L'errore con il metodo IMU rimane sempre inferiore all'errore con ICP puro, ma gli andamenti sono comparabili, a prova del fatto che l'utilizzo di un metodo piuttosto che un altro non è influenzato particolarmente dal sotto-campionamento (tralasciando l'*offset* dovuto al miglioramento generale del metodo).
Molto interessante è notare come l'andamento dei tempi computazionali associati ad una variazione del *downsampling* siano quasi perfettamente lineari. Se si osserva attentamente in figura 5.26a si potrà notare come i tempi computazionali associati al metodo ICP con IMU abbiano pendenza complessiva leggermente inferiore.



(a) Tempi computazionali LOAM e MAP



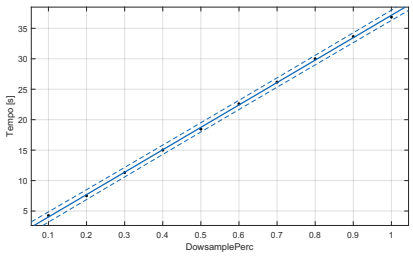
(b) Tempi computazionali complessivi

Figura 5.26: Tempi computazionali in funzione della percentuale di *downsamplig* delle *feature*

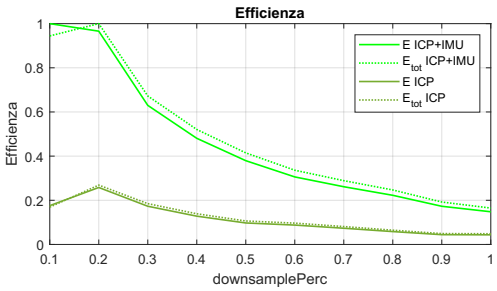
Interpolando i dati dei tempi complessivi in figura 5.26b, come mostrato in figura 5.27a, si ottiene un'equazione che permette di stimare i tempi computazionali in funzione della percentuale di *downsampling* (a $skipFrame = 4$):

$$T_{comput}(DS) = 36.81DS + 0.3373 \tag{5.8}$$

dove DS e la percentuale di *downsampling* (tra 0 e 1) e 36.81 rappresenta circa il tempo computazionale considerando tutta la nube di punti.



(a) Interpolazione tempi computazionali con intervallo di confidenza al 99%



(b) Efficienza

Figura 5.27: Interpolazione ed efficienza in funzione della percentuale di *downsampling*

Complessivamente, se si osserva la figura 5.27b in cui viene riportata l'efficienza, si potrà notare che essa tende solo a calare. Questo è logico se si pensa che l'errore rimane quasi costante ma che il tempo computazionale aumenta. Inoltre, si potrà notare come si abbia un picco dell'efficienza intorno a 0.2, mentre per valori inferiori essa sia inferiore. Questo è dovuto a quanto detto prima in merito agli errori commessi (si veda figura 5.25). Nelle simulazioni di questa tesi si è scelta una percentuale di sotto-campionamento di 0.3, questo perché si è visto che rispondeva bene anche in altre simulazioni. Inoltre, essere leggermente lontani dal punto di picco garantisce maggior stabilità.

Unendo le simulazioni con variazione di frame saltati e le simulazioni con variazione di sotto-campionamento si ottiene quanto presentato nella figura 5.28.

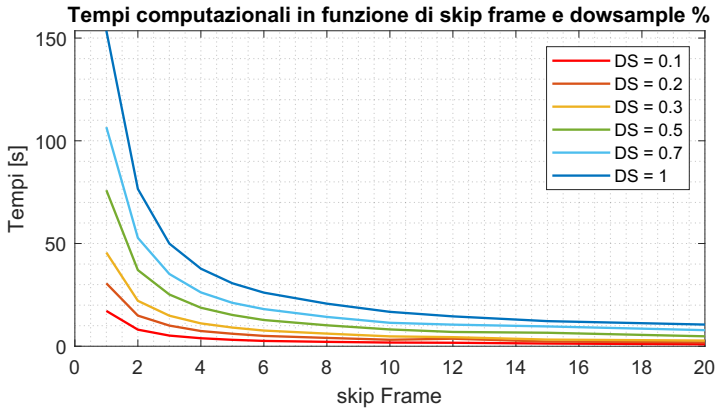


Figura 5.28: Tempi computazionali al variare di frame saltati e parametrizzato con la percentuale di *downsample*

Il grafico presenta una combinazione di andamento iperbolico dato dagli skip frame e andamento lineare dato dalla percentuale di sotto-campionamento. La funzione complessiva è ottenuta interpolando il tempo computazionale a *skipFrames* = 1 (equazione 5.9) e inserendola nell'equazione 5.5.

$$T_{comput}(DS) = 152.175DS + 0.5781 \approx 152.2DS \tag{5.9}$$

$$T_{comput}(SF, DS) = 152.2 \cdot \frac{DS}{SF} \tag{5.10}$$

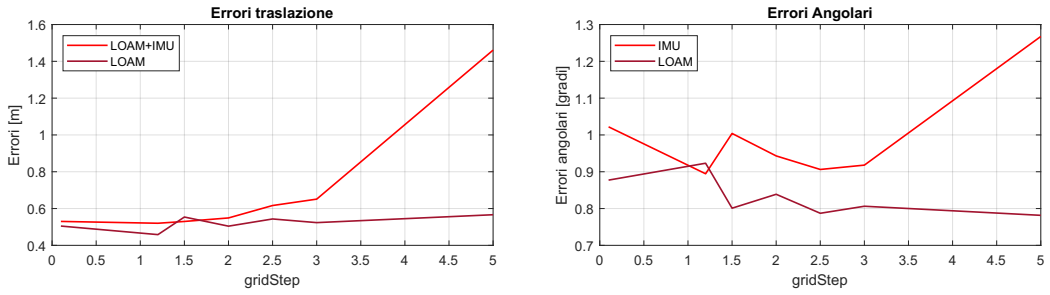
Ovviante rimane sempre un parametro da determinare, ovvero il tempo computazionale a *downsample* = 1 e *skipFrames* = 1, il quale dipenderà da moltissimi fattori.

Risposta a grid step per LOAM

Il sotto-campionamento nel metodo LOAM è stato effettuato sulle *feature*, tramite il comando `pt_LOAM = downsampleLessPlanar(pt_LOAM,gridStep)`, dove in input vengono dati i punti *feature* estratti. In questo caso entra in gioco il parametro *grid step*, che rappresenta la dimensione di griglia all'interno della quale viene effettuata la fusione dei punti estratti. Quanto segue servirà per capire meglio il comportamento delle simulazioni LOAM con e senza IMU al variare di

questo parametro con andamento tale:

$$gridStep = [0.1, 0.2, ..., 1, 1.2, 1.5, 2, 3, 5] \tag{5.11}$$

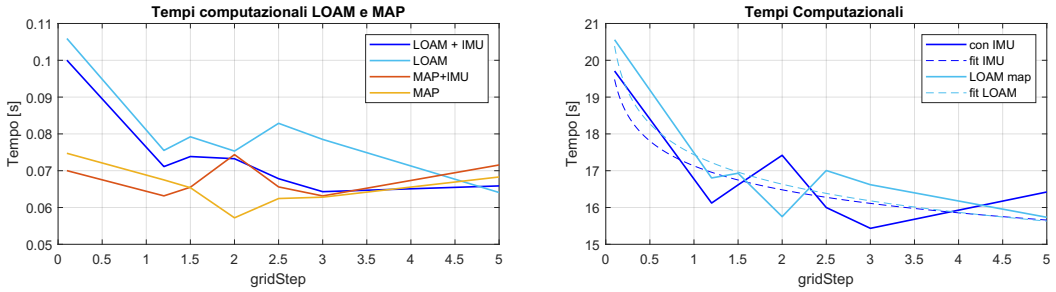


(a) Errori di traslazione (di *rms*)

(b) Errori angolari (di *rms*)

Figura 5.29: Errori in funzione del grid step di *downsamplig* delle *feature*

In figura 5.29 vengono riportati gli errori commessi in funzione del *grid step*. Come ci si poteva aspettare gli errori tendono complessivamente ad aumentare. In questo caso il metodo LOAM assistito da IMU è risultato essere più sensibile.



(a) Tempi computazionali LOAM e MAP

(b) Tempi computazionali complessivi

Figura 5.30: Tempi computazionali in funzione del grid step di downsamplig delle feature

In figura 5.30a sono riportati i tempi computazionali associati alla registrazione LOAM e MAP. Si può notare come il tempo maggiormente influenzato sia quello relativo alla registrazione LOAM. Esso tende a diminuire perché il numero di punti a disposizione dell'algoritmo diminuisce. Complessivamente in figura 5.30b i tempi computazionali diminuiscono.

Se si osserva la figura 5.31 si potrà notare come l'efficienza massima si abbia per valori di *grid step* compresi tra 1 e 2. Nelle simulazioni di questa tesi, per mantenere lo stesso *grid step* tra simulazioni con e senza IMU si è scelto un valore intermedio di 1.5.

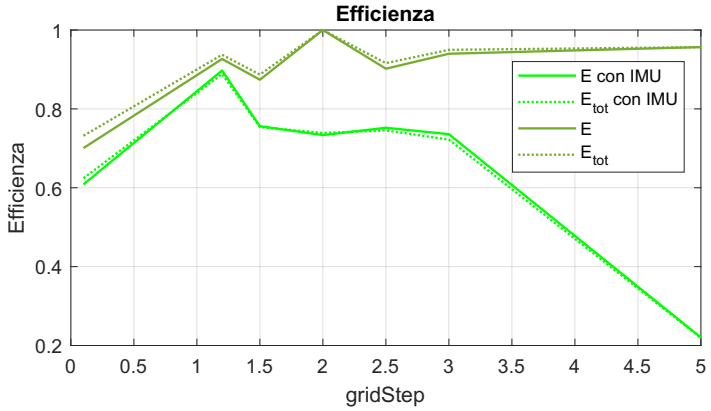


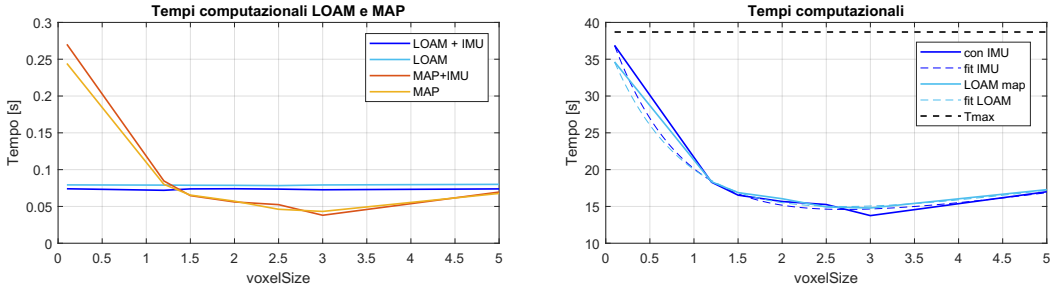
Figura 5.31: Efficienza in funzione del *grid step* di sotto-campionamento delle *feature*

Risposta a voxel size per LOAM map

Un altro parametro importante è la definizione della LOAM map, ovvero la dimensione di *voxel* ad essa associata. Questo parametro ha la stessa funzione del *grid step* per quanto riguarda il sotto-campionamento delle *feature*, ma considerando la LOAM map. Il range di variazione studiato è il medesimo del range del *grid step*:

$$voxelSize = [0.1, 0.2, ..., 1, 1.2, 1.5, 2, 3, 5] \tag{5.12}$$

In figura 5.32 sono riportati i risutati in termini di tempi computazionali.



(a) Tempi computazionali LOAM e MAP

(b) Tempi computazionali complessivi

Figura 5.32: Tempi computazionali in funzione del voxel size della LOAM map

Si nota subito dalla figura 5.32a che ad essere influenzata è solo la componente di tempo associata alla registrazione nella mappa. Ciò era prevedibile in quanto questo parametro entra in gioco dopo la registrazione LOAM, di conseguenza non ne influenza il calcolo. Se si osserva l'efficienza in figura 5.33 si potrà notare come il picco si abbia attorno a valori di *voxel* pari a 1-1.5, a confermare la scelta di 1.5 fatta nelle simulazioni di questa tesi.

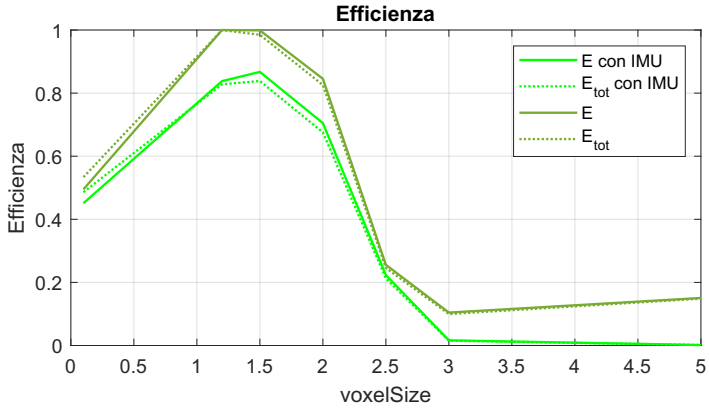
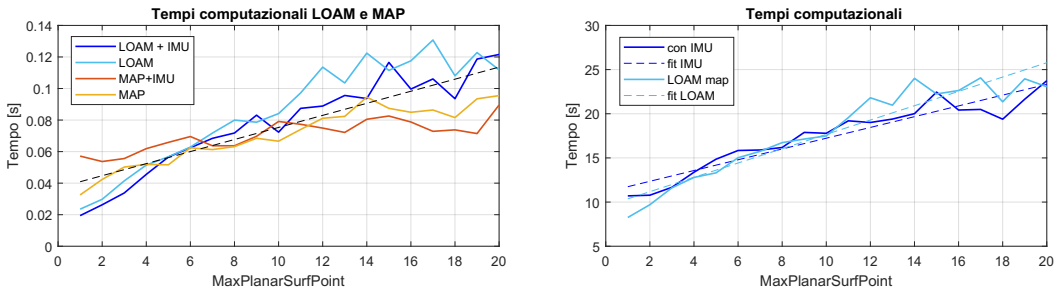


Figura 5.33: Efficienza in funzione della dimensione di *voxel* della LOAM map

5.3.4 Risposta a max surface planar points per LOAM

Si è voluto studiare questo parametro perché si è visto avere un forte impatto su tempi computazionali ed errori, e si è voluto capire meglio su quali valori settarlo. Questo parametro rappresenta il numero massimo di punti di superficie piana per regione di scansione. Come specificato nell'*help* di Matlab, LOAM suddivide ogni nuvola di punti in 6 regioni (default) che contengono un egual numero di punti. L'algoritmo cerca quindi un numero di punti pari a quelli specificati da vari parametri per ogni tipo di *feature* per ogni regione, tra cui i punti planari. Il numero di punti planari di default è 1, risultato essere troppo basso. Si sarebbero potuti modificare anche i punti massimi per le *feature* per gli spigoli, ma in questo caso si è scelto di modificare solo un parametro. Il range di variazione è:

$$MaxSurfPlanrPoint = [1, 2, ..., 20] \tag{5.13}$$



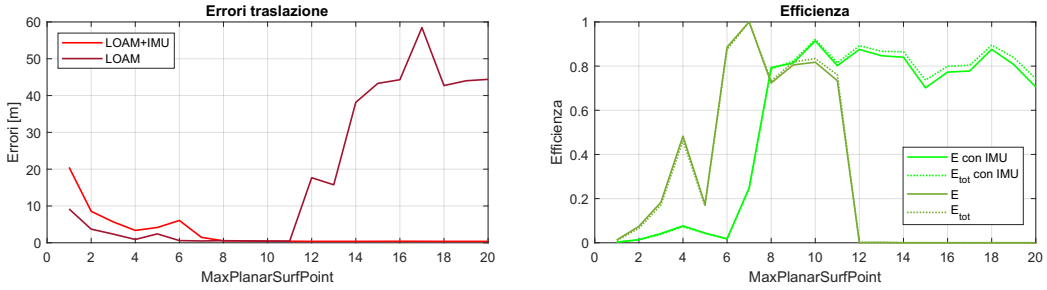
(a) Tempi computazionali LOAM e MAP

(b) Tempi computazionali complessivi

Figura 5.34: Tempi computazionali in funzione del numero massimo di punti per superficie piana

In figura 5.34 sono riportati i tempi computazionali ottenuti. Si nota come aumentando i punti studiati, aumenta quasi linearmente anche il tempo computazionale

(figura 5.34b). Inoltre, se si osserva la figura 5.34a si potrà notare come questo succeda sia per i tempi di registrazione LOAM che per la parte di registrazione nella mappa.



(a) Errori di traslazione (di *rms*)

(b) Efficienza

Figura 5.35: Risultati in funzione del numero massimo di punti per superficie piana

Molto interessante è notare come la simulazione si comporti in termini di errori. Ci si aspetterebbe che all'aumentare del numero di punti planari l'accuratezza migliori, ma ciò succede ma solo fino ad un certo valore (circa 11). La simulazione con numero di punti maggiori diverge. Questo tuttavia non succede per il metodo assistito da IMU, forse perché nel caso senza IMU la registrazione cerca un numero più ampio di punti e rischia di accettare anche *outlier*, mentre nel caso assistito da IMU la trasformazione di primo tentativo garantisce comunque una buona risposta. Complessivamente se si osserva l'efficienza si potrà notare come essa sia molto alta in valori compresi tra 6 e 11 per LOAM map pura, mentre rimanga alta dopo 8 punti per LOAM. Ecco giustificato il valore di 8 punti per superficie piana utilizzato nelle simulazioni di questa tesi.

Capitolo 6

Conclusioni

Conclusioni

Analizziamo quali erano gli obiettivi della tesi, cosa è stato fatto per raggiungerli e che risultati si sono ottenuti.

Lo **scopo della tesi** era quello di confrontare algoritmi SLAM basati su LiDAR, introducendo dapprima algoritmi basati solo sulle letture LiDAR e sviluppando poi algoritmi che implementassero una fusione debole con un'unità inerziale IMU. Quindi procedere al confronto non solo tra algoritmi di registrazione differenti ma anche tra algoritmi SLAM differenti nella loro interezza.

Un ulteriore proposito era quello di dimostrare come fosse effettivamente possibile e conveniente affidare la generazione di dati ad un **ambiente virtuale**, che fosse possibile e agevole generare dati LiDAR per svariate traiettorie senza la necessità di molteplici esperimenti sul campo.

Lo sviluppo di questa tesi è stato dettato dalle finalità inizialmente formulati. Dapprima si è creata l'interconnessione tra Matlab e l'**ambiente virtuale** Unreal Engine al fine di permette la generazione e l'immagazzinamento delle letture LiDAR. Successivamente si è sviluppato un codice per la generazione di letture IMU a partire dalle traiettorie prodotte. L'integrazione inizialmente ha dato alcuni problemi, tuttavia una volta sviluppata in maniera solida si è visto come essa permettesse effettivamente una grande libertà di simulazione, senza la limi-

tazione di tempistiche e costi. Inoltre, essa ha permesso una grande variabilità di traiettorie e settaggi per le specifiche dello strumento LiDAR.

Con i dati ottenuti si sono potuti sviluppare gli algoritmi base fondati su **ricostruzione ICP e LOAM**. Il confronto tra i vari algoritmi ha prodotto una grande quantità di informazioni. Tendenzialmente si può affermare che entrambi gli algoritmi hanno punti di forza e di debolezza. L'algoritmo ICP è indubbiamente più veloce, tuttavia tendenzialmente presenta una ricostruzione più imprecisa. L'algoritmo LOAM permette sicuramente una ricostruzione migliore, ma a discapito del tempo computazionale. Si può comunque affermare che entrambi gli algoritmi sono validi se utilizzati nelle loro condizioni ottimizzate per una specifica situazione, e sfruttando tutte le funzionalità e i parametri messi a disposizione da Matlab. Si ricorda inoltre che Matlab è un linguaggio di programmazione ad alto livello e, in quanto tale, non consente una grande libertà di modifiche e soprattutto non consente si sfruttare al massimo le potenzialità del computer. Per quanto è stato possibile si è cercato di ottene il miglior risultato con quanto messo a disposizione dal software.

Si è proseguito sviluppando l'algoritmo di pre-integrazione dell'IMU, che permettesse di creare una **fusione debole LiDAR-IMU**. Si sono quindi eseguite le simulazioni precedentemente effettuate utilizzando questa volta l'algoritmo di fusione debole, al fine di confrontarne i risultati. Si è potuto osservare come i risultati ottenuti con questo algoritmo permettessero in buona parte miglioramenti degli algoritmi precedenti e che, sebbene a volte potessero inficiare alcuni dei risultati, i vantaggi si sono dimostrati essere maggiori degli svantaggi.

In ultima analisi si sono eseguite un gran numero di simulazioni con tutti i metodi proposti facendo variare uno o più parametri contemporaneamente al fine di valutare la sensibilità di tali parametri ai suddetti algoritmi. Lo studio ha permesso sia di verificare che i parametri precedentemente utilizzati nelle simulazioni avessero valori prossimi ai valori ottimali, sia di dimostrare gli andamenti complessivi di tempi ed errori, facendo quindi uno **studio di sensibilità**. Da queste simulazioni si è anche potuto evidenziare la superiorità in termini di robustezza degli algoritmi assistiti da unità IMU.

Si può concludere quindi che gli obiettivi prefissati, seppur modesti, sono stati raggiunti, e che le varie implementazioni hanno ottenuto risultati apprezzabili.

Futuri sviluppi

Il campo della LiDAR SLAM è immensamente vasto e ha diramazioni in molti ambiti. Lo sviluppo degli ultimi anni, grazie alla domanda sempre crescente di dispositivi autonomi, ha prodotto innumerevoli ricerche, parte delle quali sono state citate e spiegate in questa tesi. Il lavoro qui effettuato si può considerare come una primo approccio a questo mondo, il quale tuttora sta facendo molti passi avanti, spinto sia dall'industria che dall'avanzamento tecnologico in campo informatico e robotico. La naturale prosecuzione di questa tesi è sicuramente provare a implementare algoritmi fortemente accoppiati con IMU (*tightly coupled*), partendo dall'algoritmo allo stato dell'arte FAST-LIO-2 e provando a fare dei piccoli passi avanti nella accuratezza e nella rapidità di calcolo di quest'ultimo, comprendendolo a fondo e cercando di svilupparne una parte. Esistono poi molti approcci di fusione differenti che sfruttano altri strumenti, quali sistemi di visione, piattaforme INS e strumenti GNSS, che possono garantire accuratezze sempre migliori.
Durante lo sviluppo dell'ottimizzazione grafica ci si è imbattuti nell'evidenza che il metodo *ScanContext* implementato in Matlab non conteneva al suo interno l'indipendenza dalla rotazione. Questo significa che nelle traiettorie in cui c'è una chiusura del loop ma con rotazione differente, questo non verrà rivelata. In futuri sviluppi si potrebbe pensare di implementare mediante codice Matlab l'algoritmo *ScanContext* completandolo con l'invarianza alla rotazione.

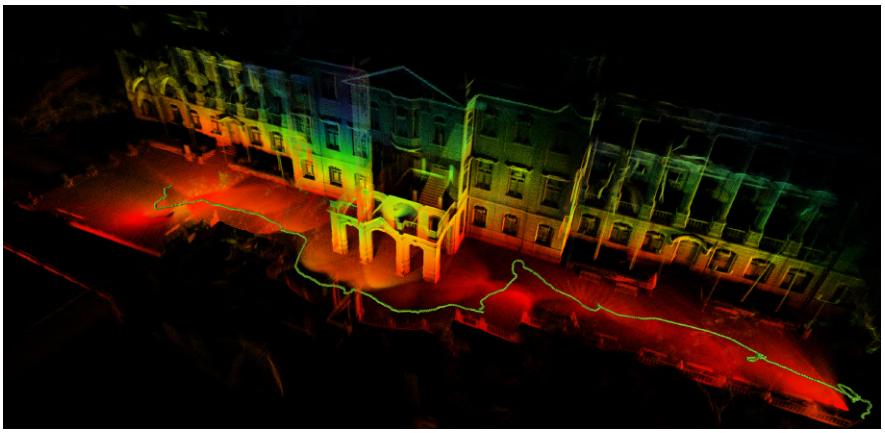


Figura 6.1: Esempio di LiDAR SLAM con algoritmo FAST-LIO2 [34]

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Henry Lim. *Introduction to SLAM (Simultaneous Localization and Mapping)*. https://ouster.com/insights/blog/introduction-to-slam-simultaneous-localization-and-mapping. 31 August 2022, (accessed: 24 November 2023).

[2] Diego Tiozzo Fasiolo, Lorenzo Scalera e Eleonora Maset. «Comparing LiDAR and IMU-based SLAM approaches for 3D robotic mapping». In: *Robotica* (2023), pp. 1–17.

[3] Julien Bo. *Choosing the best drone for LiDAR mapping*. https://www.yellowscan.com/knowledge/choosing-the-perfect-drone-for-lidar-mapping/. yellowscan. 19 December 2019, (accessed: 24 November 2023).

[4] Hugh Durrant-Whyte e Tim Bailey. «Simultaneous localization and mapping: part I». In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110.

[5] T. Bailey e H. Durrant-Whyte. «Simultaneous localization and mapping (SLAM): part II». In: *IEEE Robotics & Automation Magazine* 13.3 (2006), pp. 108–117. DOI: 10.1109/MRA.2006.1678144.

[6] Cyrill Stachniss. *Short Introduction to SLAM (From a Photogrammetry Perspective)*. https://www.ipb.uni-bonn.de/html/teaching/photo12-2021/2021-pho2-15-slam-intro.pptx.pdf. 2023, (accessed: 24 November 2023).

[7] Prof. Walter Gander. *The Singular Value Decomposition*. https://www2.math.ethz.ch/education/bachelor/lectures/hs2014/other/linalg_INFK/svdneu.pdf. ETH Zurich. 12 December 2008, (accessed: 24 November 2023).

[8] Div Gill. *Introduction to LiDAR*. https://www.mistywest.com/posts/the-411-on-lidar. Engineering Physicist EIT and former Westie. (accessed: 24 November 2023).

[9] Ouster Inc San Francisco (CA). *OS1: Mid-Range High-Resolution Imaging Lidar*. https://data.ouster.io/downloads/datasheets/datasheet-revd-v2p0-os1.pdf. 2022, (accessed: 24 November 2023).

[10] Feng Huang et al. «Point Wise or Feature Wise? A Benchmark Comparison of Publicly Available Lidar Odometry Algorithms in Urban Canyons». In: *IEEE Intelligent Transportation Systems Magazine* 14.6 (2022), pp. 155–173. DOI: 10.1109/MITS.2021.3092731.

[11] Radu Bogdan Rusu, Nico Blodow e Michael Beetz. «Fast Point Feature Histograms (FPFH) for 3D registration». In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[12] Paul J Besl e Neil D McKay. «Method for registration of 3-D shapes». In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992, pp. 586–606.

[13] Zhuli Ren, Liguan Wang e Lin Bi. «Robust GICP-based 3D LiDAR SLAM for underground mining environment». In: *Sensors* 19.13 (2019), p. 2915.

[14] Kenji Koide et al. «Voxelized GICP for fast and accurate 3D point cloud registration». In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 11054–11059.

[15] Peter Biber e Wolfgang Straßer. «The normal distributions transform: A new approach to laser scan matching». In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 3. IEEE. 2003, pp. 2743–2748.

[16] Ji Zhang e Sanjiv Singh. «LOAM: Lidar odometry and mapping in real-time.» In: *Robotics: Science and systems*. Vol. 2. 9. Berkeley, CA. 2014, pp. 1–9.

[17] Tixiao Shan e Brendan Englot. «Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain». In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4758–4765.

[18] Haoyang Ye, Yuying Chen e Ming Liu. «Tightly coupled 3d lidar inertial odometry and mapping». In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3144–3150.

[19] Qingshan Wang et al. «High-precision and fast LiDAR odometry and mapping algorithm». In: *Journal of Advanced Computational Intelligence and Intelligent Informatics* 26.2 (2022), pp. 206–216.

[20] Giseop Kim e Ayoung Kim. «Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map». In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 4802–4809. DOI: 10.1109/IROS.2018.8593953.

[21] Matlab. *Understanding SLAM Using Pose Graph Optimization — Autonomous Navigation, Part 3*. https://it.mathworks.com/videos/autonomous-navigation-part-3-understanding-slam-using-pose-graph-optimization-1594984678407.html. (accessed: 24 November 2023).

[22] Giorgio Grisetti et al. «A Tutorial on Graph-Based SLAM». In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43. DOI: 10.1109/MITS.2010.939925.

[23] Rainer Kümmerle et al. «g 2 o: A general framework for graph optimization». In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3607–3613.

[24] Simon Harris. *Inertial Measurement Unit (IMU) – An Introduction*. https://www.advancednavigation.com/tech-articles/inertial-measurement-unit-imu-an-introduction/. ADVANCE NAVIGATION. 7 July 2023, (accessed: 24 November 2023).

[25] TDK InvenSense. *ICM-20948: World's Lowest Power 9-Axis MEMS MotionTracking Device*. https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf. 2017, (accessed: 24 November 2023).

[26] Xiaobin Xu et al. «A review of multi-sensor fusion slam systems based on 3D LIDAR». In: *Remote Sensing* 14.12 (2022), p. 2835.

[27] Mathieu Labbé e François Michaud. «RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation». In: *Journal of field robotics* 36.2 (2019), pp. 416–446.

[28] Jiarong Lin e Fu Zhang. «Loam livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV». In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 3126–3131.

[29] Kenny Chen et al. «Direct lidar odometry: Fast localization with dense point clouds». In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2000–2007.

[30] Patrick Geneva et al. «Lips: Lidar-inertial 3d plane slam». In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 123–130.

[31] Tixiao Shan et al. «Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping». In: *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2020, pp. 5135–5142.

[32] Yi Zhang. «LILO: A Novel Lidar–IMU SLAM System With Loop Optimization». In: *IEEE Transactions on Aerospace and Electronic Systems* 58.4 (2022), pp. 2649–2659. DOI: 10.1109/TAES.2021.3135234.

[33] Wei Xu e Fu Zhang. «Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter». In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3317–3324.

[34] Wei Xu et al. «FAST-LIO2: Fast Direct LiDAR-Inertial Odometry». In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2053–2073. DOI: 10.1109/TRO.2022.3141876.

[35] Matlab. *Select Waypoints for Unreal Engine Simulation*. https://it.mathworks.com/help/driving/ug/select-waypoints-for-3d-simulation.html. Copyright 2019 The MathWorks Inc. 2019 (accessed: 24 November 2023).