

Algoritmo per il problema di LiDAR SLAM basato su simulazione in Unreal Engine

Filippo Ottaviani, Samuele Ceccarelli

Abstract—Questo lavoro è un report del progetto d'esame del corso Autonomous Robotics. L'obiettivo di questo lavoro è proporre una soluzione, in ambiente simulato MATLAB, al problema SLAM per un robot su ruote a guida differenziale con l'aiuto di un sensore LiDAR. Verrà descritto il problema tipico di localizzazione e mappatura per robot autonomi e definite le soluzioni secondo lo stato dell'arte attuale. In particolare, si mostrerà come è stato sfruttato il motore grafico *Unreal Engine* per la simulazione grafica, l'estrazione dei dati dai sensori LiDAR simulati e la loro elaborazione attraverso l'algoritmo LOAM per ottenere una stima accurata della traiettoria del veicolo e della mappa. Verranno infine mostrati i risultati ottenuti e si discuterà delle prestazioni dell'algoritmo proposto.

I. INTRODUZIONE

A. Problema di SLAM

Lo SLAM (Simultaneous Localization and Mapping) è un problema fondamentale nella robotica e nei sistemi autonomi, con applicazioni che spaziano dalla navigazione di veicoli autonomi alla robotica industriale e di servizio. Lo SLAM consente a un robot di costruire una mappa dell'ambiente sconosciuto e, contemporaneamente, di stimare la propria posizione all'interno di essa. Uno degli approcci più utilizzati per lo SLAM è basato sull'uso di sensori LiDAR, che offrono misurazioni ad alta risoluzione della geometria dell'ambiente circostante.

B. Stato dell'arte

Negli ultimi anni, il LiDAR SLAM ha subito notevoli miglioramenti grazie all'integrazione di tecniche di apprendimento automatico e algoritmi probabilistici avanzati. Algoritmi classici come GMapping[1] (basato su filtri di particelle) hanno dimostrato elevata affidabilità per applicazioni 2D, mentre approcci più recenti, come LOAM[2] (Lidar Odometry and Mapping), sono stati progettati per

ambienti tridimensionali. La combinazione di tecniche SLAM visive e LiDAR (V-LiDAR SLAM) ha permesso di migliorare la robustezza in scenari complessi e con scarsa struttura geometrica. Uno degli articoli più completi in merito è quello di J. Zhang e S. Singh[3].

L'adozione di simulatori avanzati per la validazione di algoritmi SLAM, come quelli integrati in MATLAB e ROS[4], ha consentito una fase di test più sicura e controllata prima dell'implementazione su sistemi reali. Il progetto descritto segue questa tendenza, sfruttando le capacità di Simulink per sviluppare e valutare la soluzione per un sistema in un ambiente virtuale, riducendo i costi e i rischi legati alle sperimentazioni sul campo.

MathWorks offre dei progetti di esempio che utilizzano Simulink e Unreal Engine per il LiDAR SLAM come [5] e [6] oppure incentrati sulla sola realizzazione della mappa come [7]. L'esempio più rilevante nella presente trattazione è stato quello relativo alla tecnica LOAM declinata per i dati LiDAR: [8].

II. INTENZIONI DEL PROGETTO

Nel presente progetto si intende realizzare un algoritmo di *full SLAM*¹ basato su LiDAR attraverso la simulazione in Simulink di un veicolo che si muove in una scena 3D. In particolare si vuole permettere all'utente di scegliere una traiettoria sulla mappa, simulare un veicolo che la segue ed estrarre le misure dal sensore LiDAR posizionato sopra di esso. Una volta ottenuti ed elaborati opportunamente i dati, si desidera implementare un algoritmo di SLAM che permetta di ottenere una ricostruzione della mappa dell'ambiente e una stima della traiettoria. Queste verranno poi confrontate con il riferimento reale² per verificare la bontà dell'algoritmo.

¹ricostruzione dell'intero percorso

²ground truth

Ci si prepone l'obiettivo di mostrare in maniera trasparente anche i risultati dei passaggi intermedi e gli effetti delle elaborazioni specifiche, oltre a rendere il sistema modulare e flessibile. Inoltre, per verificare la robustezza della soluzione, si è deciso di testarla su scenari di simulazione differenti e confrontare i risultati.

III. APPROCCIO PROPOSTO

A. Scelta di LOAM

I principali algoritmi di SLAM basati su LiDAR sono Direct LiDAR Odometry (DLO) e LiDAR Odometry And Mapping (LOAM); il primo sfrutta i dati grezzi del sensore, mentre il secondo prevede l'utilizzo delle feature estratte da essi. La scelta è ricaduta su LOAM per la sua elevata accuratezza e velocità. La procedura di base di questo algoritmo prevede:

- 1) Pre-elaborazione dei dati: viene corretta la distorsione delle scansioni causata dal movimento del sensore e vengono filtrati i punti rumorosi.
- 2) Feature extraction: si identificano due tipi principali di feature: bordi e superfici.
- 3) Odometria dalle point cloud: si calcola la trasformazione relativa tra due scansioni consecutive ad alta frequenza.
- 4) Mapping a bassa frequenza: la nuvola di punti viene raffinata e integrata in una mappa globale. L'algoritmo riduce gli errori accumulati nell'odometria attraverso ottimizzazioni locali.
- 5) Ottimizzazione della mappa e correzione degli errori: avviene un'ottimizzazione basata su GraphSLAM e la ricerca di chiusure del loop.

B. Toolbox richiesti

Per il corretto funzionamento dell'algoritmo realizzato si rendono necessari, oltre a Simulink stesso, i seguenti toolbox MATLAB:

- Simulink 3D Animation
- Automated Driving Toolbox
- Computer Vision Toolbox
- Image Processing Toolbox
- Navigation Toolbox
- Lidar Toolbox

Nei paragrafi che seguono si descriverà l'implementazione MATLAB dell'algoritmo LOAM applicato ai dati raccolti dalla simulazione.

C. Tracciamento della traiettoria

Per permettere all'utente di scegliere la traiettoria da assegnare al veicolo, ci si è serviti della funzione ausiliaria `helperSelectSceneWaypoints`. Questa apre una finestra che mostra una vista dall'alto della scena e chiede all'utente di indicare dei waypoint. Una volta selezionati arbitrariamente, la funzione restituisce al workspace la variabile `refPoses`, un array di dimensione $N \times 3$ che localizza le relative pose nello spazio. Questi vengono poi processati dalla funzione `smoothPathSpline`: il suo scopo è quello di interpolare i punti selezionati in modo da ottenere un tracciato continuo più conforme alla cinematica del veicolo e alla simulazione.



Fig. 1: Inserimento della traiettoria

D. Simulazione

Per simulare lo scenario desiderato in Simulink sono necessari tre blocchi forniti dai toolbox sopracitati:

- Simulation 3D Vehicle with Ground Following
- Simulation 3D Scene Configuration
- Simulation 3D Lidar

Il primo prende in ingresso la traiettoria e simula la cinematica del veicolo, definendone inoltre le caratteristiche visive. Il secondo configura l'ambiente 3D in cui si svolge l'azione, permettendo di selezionare delle scene preconfigurate; in prima istanza si è usato *Large parking lot*. Il terzo blocco fornisce un'interfaccia al sensore LiDAR, montato sul veicolo, per impostare alcuni parametri come la

distanza massima raggiungibile e la risoluzione angolare. Quest'ultimo blocco restituisce tre variabili tra cui una nuvola di punti, in base al campo visivo specificato e alla risoluzione angolare del sensore, l'orientamento e le coordinate.

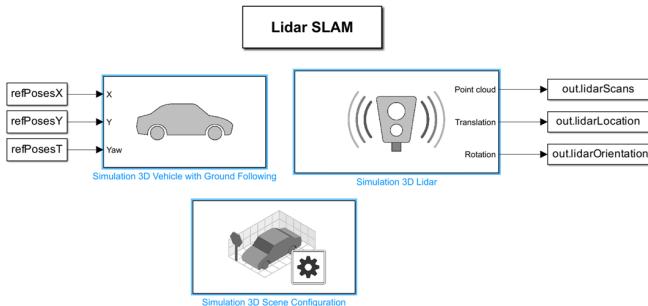


Fig. 2: Schema Simulink per la simulazione

Nella trattazione presente si estrae dal LiDAR anche il ground truth della sua roto-traslazione durante la simulazione, così da confrontare la traiettoria corretta con quella che verrà stimata attraverso la nuvola di punti. Le nuvole di punti (*point cloud*) sono oggetti di tipo `PointCloud` e sono costituite da un insieme di punti nello spazio tridimensionale, ciascuno con coordinate (x, y, z). Questi punti appartengono alle superfici degli oggetti e dell'ambiente, rilevate dal sensore.

Una volta avviata la simulazione, si apre una finestra che mostra il movimento del veicolo nella scena grazie al motore grafico Unreal Engine integrato. Terminato il tempo di simulazione (inizialmente scelto 30 secondi), viene chiusa la finestra e il modello fornisce al workspace MATLAB sia la nuvola di punti (*point cloud*) registrati dal LiDAR sia il ground truth della traiettoria del robot.



Fig. 3: Scenario simulato

E. Acquisizione dei dati LiDAR

Attraverso l'apposito riferimento si richiama la variabile di workspace relativa ai dati LiDAR

da Simulink, si visualizza l'evoluzione della nuvola di punti durante la simulazione attraverso un'animazione di grafici di tipo *scatter*. Non è ancora stata fatta nessuna elaborazione delle misure, dunque ogni scansione del sensore comprende tutti i punti acquisiti in forma grezza. Sono, infatti, visibili i punti della nuvola relativi alla strada che prendono la forma di cerchi concentrici e il cosiddetto "ego" del veicolo, ossia la sua stessa sagoma.

Si riporta in figura 4 un frame dell'animazione con la legenda relativa alla distanza (in metri) dei punti dal sensore.

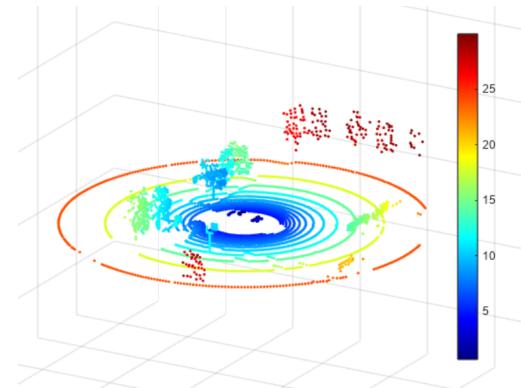


Fig. 4: Visualizzazione di una scansione LiDAR

F. SLAM

Ricordiamo che questa soluzione sfrutta il concetto di feature per ottenere una comprensione semantica della scena; questo la rende precisa e computazionalmente efficiente. Come anticipato, le feature rilevate sono di due tipi:

- **Bordi:** sono rilevati lungo spigoli e contorni e vengono usati principalmente per stimare la rotazione.
- **Superfici:** sono rilevate su aree planari e vengono usate principalmente per stimare la traslazione.

La funzione usata per rilevarle è `detectLOAMFeatures` che restituisce un oggetto speciale di tipo `LOAMPoints`. Per ricavare la trasformazione tra due nuvole di punti successive si ricorre a `pcregisterloam` che restituisce un oggetto `rigidtform3d` contenente la matrice di roto-traslazione. Lo scopo finale è ricavare tutte le matrici relative, comporle e moltiplicarle per la posa iniziale che deve essere nota. Questo procedimento permette di ottenere per

ogni scansione LiDAR la posa del veicolo associata e, dunque, di ricavare la traiettoria complessiva.

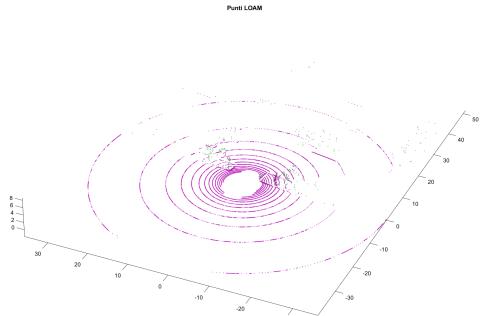


Fig. 5: Punti LOAM di una singola scansione

Negli sviluppi iniziali del progetto, la prima elaborazione dei dati LiDAR sotto forma di array di point cloud era un preprocessing che prevedeva:

- Downsampling,
- Denoising,
- Rimozione del cosiddetto "ego" del veicolo e dei punti appartenenti alla superficie stradale.

Una volta posta l'attenzione sull'algoritmo LOAM, però, si è preso atto che questa pratica fosse esplicitamente sconsigliata dal documento [8] in quanto è il *feature detector*³ ad occuparsi della scelta dei punti utili. L'unico filtraggio che avviene a monte dell'algoritmo LOAM, consiste nella rimozione di tutti i punti esterni a un cilindro di raggio prefissato e centrato nel veicolo (usando la funzione `findPointsInCylinder`): quest'operazione è utile quindi per eliminare punti troppo vicini al sensore che potrebbero appartenere al veicolo e i punti troppo lontani che potrebbero appesantire la computazione senza fornire un contributo rilevante alla stima.

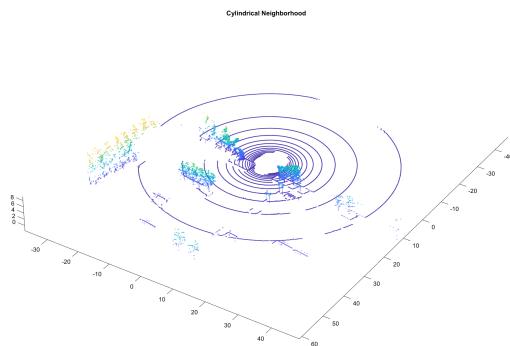


Fig. 6: Filtraggio cilindrico

³la funzione `detectLOAMFeatures` citata sopra

A questo punto si è deciso di introdurre il tuning di un iperparametro molto importante: `maxPlanarSurfacePoints` che controlla il numero massimo di punti su superfici planari che la funzione `detectLOAMFeatures` può estrarre da ogni regione: un valore basso permetterà di concentrarsi maggiormente su punti di spigoli e angoli, mentre un valore alto può migliorare la stabilità in ambienti con pochi spigoli, prediligendo l'estrazione di punti appartenenti a superfici planari. Lo scopo dell'ottimizzazione di tale parametro è quello di minimizzare l'RMSE per le features rilevate tra due scansioni successive.

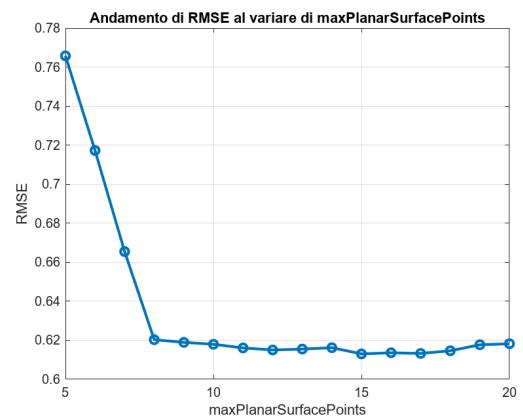


Fig. 7: Relazione del parametro con l'RMSE

La prima fase dell'algoritmo di SLAM prevede la stima della traiettoria senza effettuare la mappatura: in primis, viene definita la posa iniziale grazie alla ground truth proveniente dal LiDAR, si configura la finestra per la visualizzazione della traiettoria e si inizia il ciclo di calcolo della traiettoria. In particolare, ogni iterazione richiede i seguenti passaggi:

- 1) Viene selezionata e filtrata la point cloud corrente.
- 2) Vengono estratte le feature ed effettuato un downsample dei punti.
- 3) Viene calcolata la trasformazione tra le feature della point cloud precedente e quelle della corrente.
- 4) La trasformazione viene applicata alla posa precedente per ottenere quella attuale.
- 5) La stima della posa attuale viene aggiunta alla traiettoria e alla finestra di visualizzazione.

A valle si mostra la traiettoria risultante da questa prima elaborazione a confronto con il riferimento e, nella maggior parte dei casi, si riscontra una crescente deriva rispetto ad esso.



Fig. 8: Traiettoria dopo la prima fase

La seconda fase prevede la creazione della mappa e la correzione della traiettoria grazie ad essa. La classe fondamentale in questa sezione dell'algoritmo è `pcmap1oam` che gestisce le feature ricavate in precedenza, costruisce mappe efficienti⁴ grazie all'aggiunta di nuove feature e permette di stimare le trasformazioni tra queste e quelle già registrate. Le iterazioni richieste da questa fase si articolano nei seguenti passaggi:

- 1) Viene selezionata e filtrata la point cloud corrente.
- 2) Vengono estratte le feature ed effettuato un downsampling dei punti.
- 3) Viene calcolata la trasformazione relativa tra le feature della point cloud precedente e quelle della corrente.
- 4) Si stima, grazie alla funzione `findPose`, la posa assoluta del robot.
- 5) La stima della posa attuale viene aggiunta alla traiettoria e alla finestra di visualizzazione.
- 6) La mappa LOAM viene aggiornata con le feature della scansione corrente riproiettate grazie alla posa assoluta appena calcolata.

Si mostra nella figura 9 la nuova traiettoria e il riferimento.



Fig. 9: Traiettoria dopo la prima fase

⁴grazie al downsampling basato su `Voxelsize` per gestire grandi nuvole di punti.

Si noti come la stima, dopo questa seconda elaborazione, risulti molto più aderente alla realtà.

Nel dettaglio, `findPose` esegue una stima della posa attuale del sensore sulla base della point cloud in esame confrontandola con la mappa ricavata fino a quel momento e estraendo la trasformazione rigida che "spiega" meglio le corrispondenze tra esse.

G. Valutazioni finali

Per valutare la correttezza dell'algoritmo appena illustrato, ci si è avvalsi di rappresentazioni grafiche (figura 10) e metriche di errore; in particolare, si mostra una comparazione tra le traiettorie stimate nelle due fasi e quella di riferimento registrata dal ground truth del LiDAR.

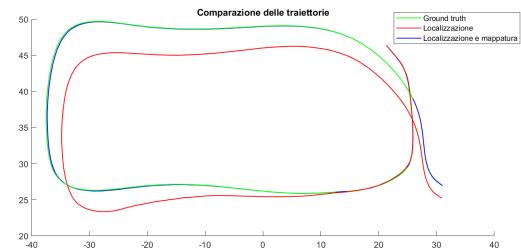


Fig. 10: Traiettorie a confronto

Viene, quindi, apposta la mappa creata nella seconda fase di SLAM alle tre traiettorie appena descritte.

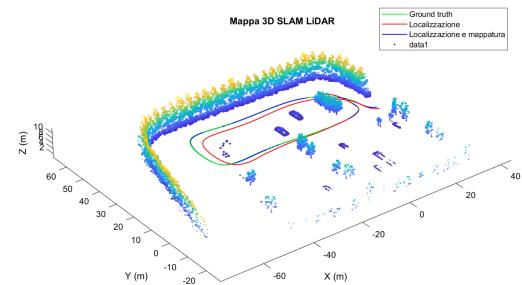


Fig. 11: Mappa e traiettorie

Inoltre, viene calcolato e stampato a schermo un confronto tra le traiettorie stimate nelle due fasi in termini di RMSE. Il risultato di una simulazione è riportato nella tabella I

Fase	Tipo	RMSE
1	odometria	3.1187 m
2	odometria e mappatura	0.1692 m

TABLE I: Confronto degli RMSE

IV. OSSERVAZIONI SPERIMENTALI

Per verificare la validità dell'algoritmo realizzato si è scelto di ripetere la simulazione in scenari diversi e valutare i relativi rendimenti. Si è quindi selezionata la scena *US City Block* che introduce delle caratteristiche diverse rispetto a *Large Parking Lot* come le ampie superfici dei palazzi.



Fig. 12: Scena simulata

Lo scenario risulta meno vario a causa della scarsa diversità tra le superfici, quindi il risultato della prima fase diverge rispetto al riferimento. La seconda fase migliora sensibilmente la stima come visibile nel confronto della tabella II.

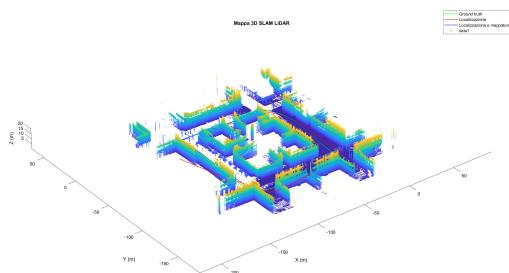


Fig. 13: Mappa e traiettorie a confronto

Si è, inoltre, riscontrato l'effetto dell'inclinazione del veicolo, e dunque del sensore, dovuto al transito sul marciapiede. Seppur queste condizioni la leggibilità della mappa, sembra non avere effetti sulla corretta stima della traiettoria, dimostrando la robustezza dell'algoritmo.

Fase	Tipo	RMSE
1	odometria	14.2871 m
2	odometria e mappatura	2.0552 m

TABLE II: Confronto degli RMSE

V. CONCLUSIONI

A. Commenti

Durante la realizzazione del progetto si è scelto esplorare in dettaglio sia gli aspetti teorici alla base del funzionamento del LiDAR SLAM, sia la

sua implementazione tramite MATLAB. Questo ha richiesto di trattare argomenti come trasformazioni rigide nello spazio tridimensionale, elaborazione di nuvole di punti e visualizzazioni 3D.

La sperimentazione con la classe `pcmaploam` è stata particolarmente utile per comprendere il funzionamento dell'algoritmo e dei suoi metodi. L'esperienza è stata formativa anche per quanto riguarda la configurazione del sistema Simulink, delle sue componenti e della sua interazione con il workspace MATLAB.

Prima di giungere alla conclusione di utilizzare LOAM, sono stati effettuati diversi tentativi coinvolgendo gli algoritmi di LiDAR SLAM più diffusi, come quelli basati su mappe NDT⁵. Questa ricerca ha permesso di maturare una buona panoramica dello stato dell'arte in merito.

B. Sviluppi futuri

Le possibili evoluzioni del progetto presentato sono molte, ma quelle che più ci sembrano rilevanti sono:

- Confronto con altri algoritmi.
- Elaborazione di una metrica volta a valutare la qualità della mappa realizzata.
- Confronto con un algoritmo di VSLAM basato su una camera.
- Fusione delle osservazioni LiDAR con quelle proprietive di sensori IMU.

In merito a quest'ultima proposta, nel modello Simulink sono riportati in forma di commento due sottosistemi atti a simulare la presenza di un sensore IMU a bordo del veicolo. Il sottoblocco più piccolo aggiunge ai segnali di ground truth un rumore uniforme mentre quello più grande ospita il vero e proprio blocco IMU che restituisce le accelerazioni lineari e angolari del veicolo⁶. Contestualmente, è visibile nella parte finale dello script MATLAB una sezione di codice commentata che permette di visualizzare la traiettoria stimata dalle misurazioni proprietarie rumorose. Questo eventuale sviluppo futuro avrebbe adottato la soluzione presente nell'esempio [5] per ricostruire la mappa e, di conseguenza, la traiettoria stimata.

⁵ad esempio `pcregisterndt`.

⁶questo blocco di fatto non funziona ancora correttamente

C. Note

Il codice, le animazioni, le immagini e la documentazione sono consultabili nella repository GitHub dedicata [link].

BIBLIOGRAFIA

- [1] B. Balasuriya, B. Chathuranga, B. Jayasundara, N. Napagoda, S. Kumarawadu, D. Chandima, and A. Jayasekara, “Outdoor robot navigation using gmapping based slam algorithm,” in *2016 Moratuwa Engineering Research Conference (MERCon)*, 2016, pp. 403–408.
- [2] Y. Su, T. Wang, S. Shao, C. Yao, and Z. Wang, “Gr-loam: Lidar-based sensor fusion slam for ground robots on complex terrain,” *Robotics and Autonomous Systems*, vol. 140, p. 103759, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0921889021000440>
- [3] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time,” in *Proceedings of Robotics: Science and Systems (RSS)*. Berkeley, CA, USA: RSS Foundation, July 2014. [Online]. Available: <https://www.ri.cmu.edu/publications/loam-lidar-odometry-and-mapping-in-real-time/>
- [4] MathWorks. (2024) Build a map using lidar slam with ros in matlab. Accessed: Feb 22, 2025. [Online]. Available: <https://it.mathworks.com/help/ros/ug/build-a-map-using-lidar-slam-with-ros-in-matlab.html>
- [5] ——. (2024) Design lidar slam algorithm using unreal engine simulation environment. Accessed: Feb 22, 2025. [Online]. Available: <https://it.mathworks.com/help/driving/ug/design-lidar-slam-algorithm-using-3d-simulation-environment.html>
- [6] ——. (2024) Build occupancy map from 3-d lidar data using slam. Accessed: Feb 26, 2025. [Online]. Available: <https://it.mathworks.com/help/driving/ug/build-occupancy-map-from-lidar-data-using-slam.html>
- [7] ——. (2024) Lidar localization with unreal engine simulation. Accessed: Feb 23, 2025. [Online]. Available: <https://it.mathworks.com/help/driving/ug/lidar-localization-with-unreal-engine-simulation.html>
- [8] ——. (2024) Build a map with lidar odometry and mapping (loam) using unreal engine simulation. Accessed: Mar 22, 2025. [Online]. Available: <https://it.mathworks.com/help/lidar/ug/build-map-with-loam-using-unreal-engine.html>