Progetto Autonomous Robotics

Filippo Ottaviani, Samuele Ceccarelli

***Abstract*—Questo lavoro è un report del progetto d'esame del corso *Autonomous Robotics*. L'obbiettivo di questo lavoro è proporre una soluzione, in ambiente simulato MATLAB, al tipico problema SLAM per un robot su ruote a guida differenziale. 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 mototore grafico *Unreal Engine* per la simulazione grafica e l'estrazione dei dati dai sensori LiDAR simulati, per ottenere la soluzione proposta al problema di SLAM. Verrano infine mostrati i risulati ottenuti e si discuterà delle performance 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. Il 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 il 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.

L'adozione di simulatori avanzati per la validazione di algoritmi SLAM, come quelli integrati in MATLAB e ROS[3], 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 un sistema LiDAR SLAM 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 [4] e [5] oppure incentrati sulla sola realizzazione della mappa come [6]. L'esempio più rilevante nella presente trattazione è stato quello relativo alla tecnica LOAM declinata per i dati LiDAR: [7]

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 reale[2] per verificare la bontà dell'algoritmo.

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.

[1] ricostruzione dell'intero percorso
[2] ground truth

III. Approccio proposto

A. Toolbox richiesti

Per il corretto funzionamento dell'algoritmo realizzato si rendono necessari i seguenti toolbox MATLAB:

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

B. 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, la funzione restituisce al workspace la variabile `refPoses`, un array a tre dimensioni che localizza i waypoint nello spazio. Questi vengono poi processati dalla funzione `smoothPathSpline` per ottenere un tracciato continuo più conforme alla cinematica del veicolo.

Fig. 1: Inserimento della traiettoria

C. Simulazione

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

- 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 e ne definisce le caratteristiche visive. Il secondo configura l'ambiente 3D in cui si svolge l'azione; in prima istanza si è usato *Large parking lot*. Il terzo blocco fornisce un'interfaccia al sensore LiDAR montato sul veicolo. Restituisce una nuvola di punti in base al campo visivo specificato e alla risoluzione angolare del sensore. Nel nostro caso estraiamo 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 costituite da un insieme di punti nello spazio tridimensionale, ciascuno con coordinate (x, y, z). Questi punti rappresentano le superfici degli oggetti e dell'ambiente rilevate durante la simulazione.

Una volta avviata la simulazione, si apre una finestra che mostra il movimento del veicolo nella scena grazie al motore grafico Unreal Engine. Terminato il tempo di simulazione (generalmente 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.

D. Acquisizione dei dati LiDAR

Estratti dati LiDAR dal 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 misura del sensore è visibile sotto forma di punto della nuvola.

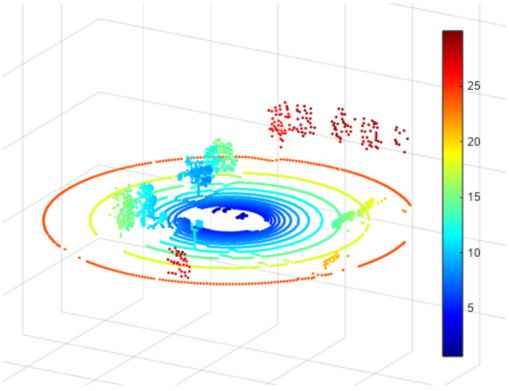

Fig. 2: Animazione della nuvola LiDAR

E. SLAM

L'algoritmo scelto è LOAM, uno dei due trattati a lezione nell'ambito del problema di SLAM basato su dati LiDAR. Rispetto alla controparte DLO, questa soluzione sfrutta il concetto di feature per ottenere una comprensione semantica della scena; questo lo rende preciso e computazionalmente efficiente. Le feature rilevate sono di due tipi:

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

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 della superficie stradale

Una volta posta l'attenzione sull'algoritmo LOAM, però, si è preso atto che questa pratica fosse espressamente sconsigliata dal documento [7] 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 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 punti troppo lontani dal sensore che potrebbero affaticare la computazione.

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.

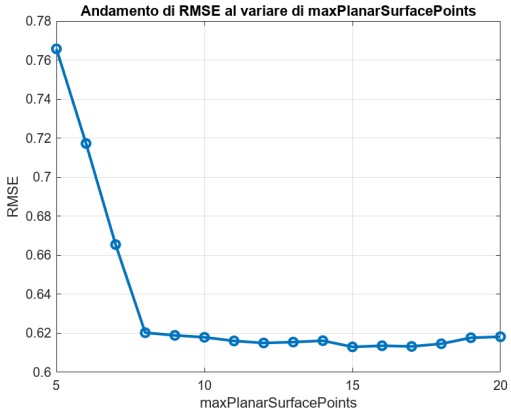

Fig. 3: Relazione del parametro con l'RMSE

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.

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 `pcmaploam` che gestisce le feature ricavate in precedenza, costruisce mappe efficienti[4] grazie all'aggiunta di nuove feature e permette di

[3] la funzione `detectLOAMFeatures` citata sopra

[4] grazie al downsampling basato su `VoxelSize` per gestire grandi nuvole di punti.

stimare le trasformazioni tra queste e quelle registrate in precedenza. 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 in base alle feature LiDAR e alla mappa LOAM generata finora.
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.

Conclusa questa fase, si mostra la nuova traiettoria e il riferimento.

F. Valutazioni finali

Per valutare la correttezza dell'algoritmo appena illustrato, ci si è avvalsi di rappresentazioni grafiche 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.

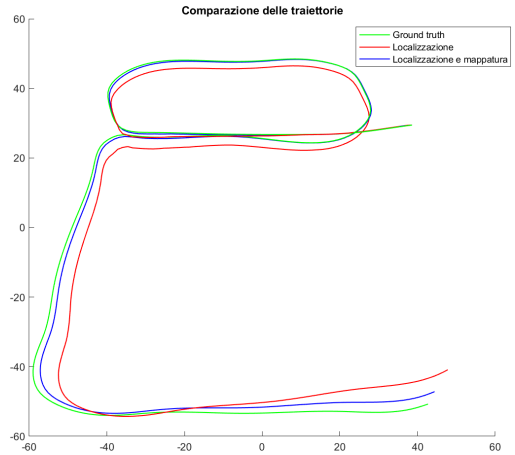

Fig. 4: Traiettorie a confronto

Viene, quindi, apposta la mappa creata nella seconda fase di SLAM alle tre traiettorie appena descritte. Inoltre, viene calcolato e stampato a schermo un confronto tra le traiettorie stimate nelle due fasi in termini di RMSE.

IV. Osservazioni sperimentali

la descrizione dell'esperimento, con figure e/o tabelle

V. Conclusioni

A. Commenti

B. Sviluppi futuri

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] MathWorks, "Build a map using lidar slam with ros in matlab," 2024, accessed: Feb 22, 2025. [Online]. Available: https://it.mathworks.com/help/ros/ug/build-a-map-using-lidar-slam-with-ros-in-matlab.html

[4] ——, "Design lidar slam algorithm using unreal engine simulation environment," 2024, accessed: Feb 22, 2025. [Online]. Available: https://it.mathworks.com/help/driving/ug/design-lidar-slam-algorithm-using-3d-simulation-environment.html

[5] ——, "Build occupancy map from 3-d lidar data using slam," 2024, accessed: Feb 26, 2025. [Online]. Available: https://it.mathworks.com/help/driving/ug/build-occupancy-map-from-lidar-data-using-slam.html

[6] ——, "Lidar localization with unreal engine simulation," 2024, accessed: Feb 23, 2025. [Online]. Available: https://it.mathworks.com/help/driving/ug/lidar-localization-with-unreal-engine-simulation.html

[7] ——, "Build a map with lidar odometry and mapping (loam) using unreal engine simulation," 2024, accessed: Mar 22, 2025. [Online]. Available: https://it.mathworks.com/help/lidar/ug/build-map-with-loam-using-unreal-engine.html