Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

Ministero della Transizione Ecologica

Ricerca di Sistema elettrico

Analisi dello stato dell'arte e sperimentazione in ambiente simulato di algoritmi per l'integrazione di funzionalità autonome in un veicolo elettrico

E. Bellocchio, F. Crocetti, G. Costante, M.L. Fravolini, P. Valigi

A.D. 1308 unipg
UNIVERSITÀ DEGLI STUDI DI PERUGIA

ISARlab

Report RdS/PTR(2020)/025

ANALISI DELLO STATO DELL'ARTE E SPERIMENTAZIONE IN AMBIENTE SIMULATO DI ALGORITMI PER L'INTEGRAZIONE DI FUNZIONALITÀ AUTONOME IN UN VEICOLO ELETTRICO

E. Bellocchio, F. Crocetti, G. Costante, M.L. Fravolini, P. Valigi (Intelligent Systems, Automation and Robotics Laboratory (ISARLab) - Dipartimento di Ingegneria, Università degli Studi di Perugia)

Aprile 2021

Report Ricerca di Sistema Elettrico

Accordo di Programma Ministero dello Sviluppo Economico - ENEA
Piano Triennale di Realizzazione 2019-2021 - II annualità
Obiettivo: Tecnologie
Progetto: Tecnologie per la penetrazione efficiente del vettore elettrico negli usi finali
Work package: Local Energy District
Linea di attività: LA43 - Integrazione di funzionalità autonome in un veicolo elettrico
Responsabile del Progetto: Claudia Meloni, ENEA
Responsabile del Work package: Claudia Meloni, ENEA

Il presente documento descrive le attività di ricerca svolte all'interno dell'Accordo di collaborazione "Integrazione di funzionalità autonome in un veicolo elettrico e interoperabilità con l'infrastruttura "smart road""
Responsabile scientifico ENEA: Sergio Taraglio
Responsabile scientifico Dipartimento di Ingegneria - Università degli Studi di Perugia: Mario Luca Fravolini

Responsabile tecnico Dipartimento di Ingegneria - Università degli Studi di Perugia: Gabriele Costante

Indice

Sommario

Questo documento contiene il report delle attività svolte dal gruppo di ricerca ISARLab del Dipartimento di Ingegneria dell'Università degli Studi di Perugia nel periodo 01/01/2020 - 31/12/2020 relativamente al progetto "Integrazione di funzionalità autonome in un veicolo elettrico e interoperabilità con l'infrastruttura "smart road"", di seguito indicato per brevità "Smart Road".
In questo progetto le attività dell'ISARLab si sono concentrate principalmente sullo sviluppo di algoritmi per la localizzazione e la navigazione di veicoli autonomi utilizzando dati e informazioni raccolte da sensori eterogenei. Dal punto di vista tecnologico sono state analizzate, sviluppate e integrate metodologie allo stato dell'arte per quanto riguarda i sistemi di visione, controllo e navigazione. In particolare i sistemi di localizzazione utilizzati permettono l'integrazione di misure raccolte da sensori GPS-RTK, sensori inerziali (IMU), sensori laser, sensori LIDAR e di sistemi di visione stereoscopici.
La validazione delle soluzioni individuate, in questo primo anno di attività è stata effettuata utilizzando ambienti simulati 3D fotorealistici. Questo approccio ha consentito una rapida verifica preliminare dell'efficacia e dell'efficienza dei vari algoritmi impiegati, riducendo gli oneri ed i rischi legati a sperimentazioni e alle configurazioni in ambiente reale. In particolare, è stata realizzata una versione simulata di un ambiente stradale cittadino all'interno del quale si muove una autovetture, equipaggiata con sensori virtuali con caratteristiche simili a quelli reali che verranno installati nel prototipo di veicolo. In questo ambiente sono quindi state condotte delle sperimentazioni qualitative e quantitative volte alla valutazione dell'efficacia degli algoritmi di localizzazione e navigazione autonoma.
Infine, per effettuare una prima validazione degli algoritmi in un contesto reale, è stata condotta una campagna sperimentale utilizzando una piattaforma robotica mobile in dotazione al Dipartimento di Ingegneria (il robot Turtlebot).

1 Introduzione (stile Titolo 1)

L'obiettivo del progetto SMART ROAD è la realizzazione di una infrastruttura stradale intelligente in cui sensori eterogenei e distribuiti su pali per l'illuminazione stradale interagiscono e scambiano dati con veicoli elettrici autonomi. Lo scambio continuo di dati e informazioni ha un impatto diretto sulla riduzione della richiesta energetica da parte dei veicoli e sull'incremento della sicurezza.

Al fine di misurare e comprovare i benefici dell'infrastruttura proposta, è sicuramente fondamentale sviluppare e implementare funzionalità per la localizzazione e la guida autonoma da integrare nel veicolo elettrico considerato per le sperimentazioni. Questi aspetti sono oggetto delle attività svolte dal gruppo di ricerca ISARLab del Dipartimento di Ingegneria dell'Università degli studi di Perugia. I risultati discussi in questo documento fanno riferimento alle attività svolte nel periodo 01/01/2020 - 31/12/2020.

La finalità delle soluzioni e delle metodologie proposte e descritte in questo report è quella di consentire al veicolo elettrico messo a disposizione dal Laboratorio DTE-SEN-IDRA dell'ENEA di localizzarsi e navigare autonomamente tramite il processamento di dati e informazioni provenienti dai sensori eterogenei presenti a bordo.

In questo contesto, le attività svolte si possono suddividere in 3 fasi:

1. Analisi dello stato dell'arte in merito ad algoritmi e soluzioni per la localizzazione e la navigazione di piattaforme mobili. Gli approcci presi in considerazione includono strategie per il Simultaneous Localization and Mapping (SLAM) basato su varie tipologie di sensori, con particolare attenzione a quelli di visione e ai sensori laser.
2. Implementazione e test delle strategie di localizzazione e path planning in ambiente simulato. In questa fase, sono stati realizzati ambienti con elevato grado di fotorealismo al fine di testare gli algoritmi implementati in un contesto controllato e facilmente riproducibile. Sono stati inoltre messi a confronto diversi algoritmi di localizzazione combinando le misure dei sensori virtuali tramite tecniche di sensor fusion allo scopo di incrementarne la precisione. È importante sottolineare che l'utilizzo degli ambienti simulati è stato cruciale per sviluppare gli algoritmi di localizzazione e navigazione senza necessità di avere a disposizione il veicolo elettrico. Questo consentirà, nella seconda fase del progetto, di accelerare sensibilmente l'integrazione del software nel veicolo elettrico, riducendo al minimo la necessità di sviluppare ulteriore codice.
3. Test preliminari su piattaforme reali. Ultimata la validazione in ambiente simulato, al fine di effettuare dei primi test in scenari reali, sono stati effettuati alcuni test sperimentali impiegando una piattaforma robotica a disposizione del Dipartimento di Ingegneria (ossia la piattaforma didattica Turtlebot).

Il resto del documento è organizzato come segue: nella sezione 2 viene discusso lo stato dell'arte; nella sezione 3 vengono descritti gli algoritmi e le strategie implementate e discussi i risultati sperimentali; infine, la sezione 4 contiene le conclusioni e gli sviluppi futuri del progetto previsti per l'anno 2021.

2 Stato dell'Arte

In questo paragrafo viene presentata l'analisi delle tecniche a stato dell'arte per quanto riguarda la localizzazione e la navigazione di sistemi robotici, sia in ambienti outdoor che indoor. La prima parte di questo paragrafo è dedicata ai metodi di localizzazione e mapping (SLAM, Simultaneous Localization And Mapping) ed è suddivisa in diverse sezioni in base alla tipologia di sensore che viene utilizzata; nella seconda parte vengono illustrati i metodi per la navigazione.

2.1 Sistemi di localizzazione

In questo paragrafo verrà descritta la letteratura a stato dell'arte per quanto riguarda i metodi di SLAM. Come anticipato nel paragrafo precedente i vari metodi verranno suddivisi in base al tipo di sensori utilizzati per la localizzazione. Prima di scendere nel dettaglio delle tecniche più avanzate, è opportuno

specificare cosa si intende in genere per tecniche di tipo SLAM, qual è l'architettura tipica di un algoritmo SLAM e le principali tecniche algoritmiche alla base del loro funzionamento.

Con il termine SLAM, acronimo di Simultaneous Localization And Mapping, si intende la stima simultanea dello stato di un robot dotato di sensori a bordo, e la costruzione di un modello (la mappa) dell'ambiente che i sensori stanno percependo. Tipicamente, in questo contesto, lo stato di un robot è identificato dalla sua posa (posizione ed orientamento), mentre la mappa è una rappresentazione degli aspetti di interesse che descrivono l'ambiente in cui opera il robot (Cadena et al. 2016) (Huang et al., 2020).

In figura 1 è riportato lo schema a blocchi di una generica architettura di un sistema SLAM; questa è composto da due componenti, il front-end, che si occupa di estrarre i dati dai sensori e di renderli compatibili e fruibili per le successive attività di elaborazione, ed il back-end, che si occupa di risolvere per via algoritmica il problema dello SLAM utilizzando i dati messi a disposizione dal front-end.

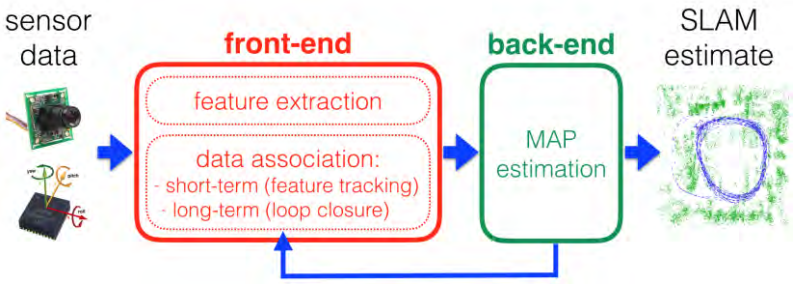

Figura 1. Generica architettura di SLAM

Il back-end solitamente utilizza tecniche di filtraggio basate su filtri di Bayes, come l'Extended Kalman Filter (EKF) o i filtri particellari (in particolare Rao-Blackwellized Particle Filter - RBPF). Una alternativa molto diffusa a tali tecniche è costituita dalle strategie basate sull'ottimizzazione su grafo (Grisetti et al., 2010) (un esempio in appicazioni di SLAM è riportato in figura 2). In particolare, quest'ultima soluzione prevede, in prima istanza, la costruzione di un grafo i cui nodi codificano le posizioni del robot e gli elementi salienti della mappa. Successivamente, nodi ed archi vengono ottimizzati minimizzando una funzione obiettivo che tiene conto dei vincoli di consistenza che emergono, ad esempio, dall'aver osservato un elemento della mappa da più posizioni.

Al back-end non è demandata, in genere, la gestione diretta dei dati estratti dai sensori; ad esempio, nel caso in cui i dati provenienti dai sensori sono immagini, non è, in pratica, effettuata la modellazione della relazione che lega l'intensità di ogni pixel e le posizioni del robot. La stessa considerazione vale per i dati acquisiti da sensori più semplici.

In effetti è tipicamente necessario un processamento preliminare dei dati grezzi al fine di estrarre da questi le caratteristiche e gli elementi fondamentali che ne riassumono il contenuto informativo, rimuovendo gli aspetti non significativi per la localizzazione ed attenuando gli effetti dei rumori di misura. Questa serie di operazioni è svolta dalla componente di front-end.

Nel caso di SLAM basato su visione vengono normalmente estratte delle caratteristiche, o features, dalle immagini, che sono associate ad elementi dell'ambiente altamente discriminativi e quindi particolarmente adatte ad essere impiegate come dati di riferimento per la localizzazione del robot. In questo modo, al back-end viene messa a disposizione una rappresentazione del mondo semplificata, più robusta ed adatta alla elaborazione.

Il front-end si occupa anche di associare ad ogni set di misure provenienti dai sensori uno specifico punto di riferimento (localizzazione) all'interno dell'ambiente (3D o 2D, a seconda del tipo di mappa che si sta via via costruendo ed alla tipologia dei sensori di cui dispone il veicolo).

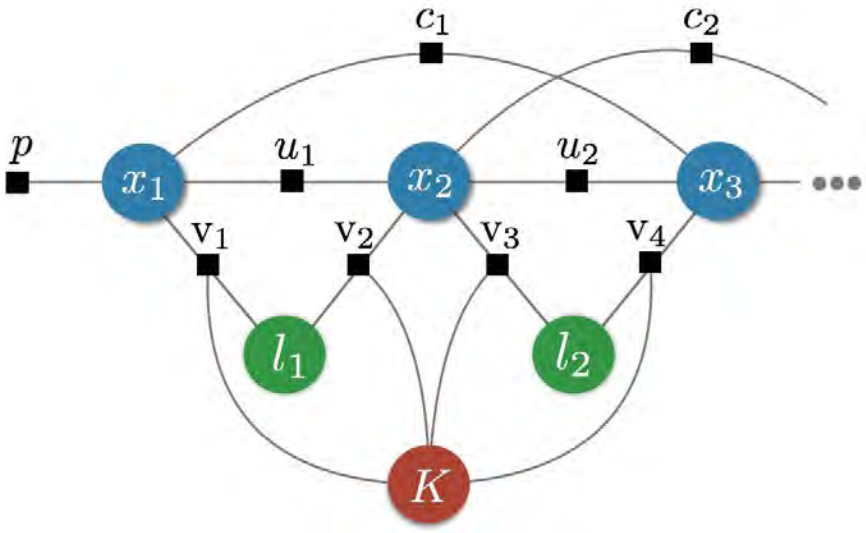

Figura 2. Esempio di rappresentazione a grafo per un problema di SLAM: i nodi blu (x_1, x_2, x_3) rappresentano le pose del robot in istanti successivi, l'arco p rappresenta la dipendenza delle pose del robot con le pose precedenti non rappresentate dal grafo. I nodi in verde (l_1, l_2) rappresentano le posizioni dei landmark, ossia i punti dell'ambiente facilmente riconoscibili (possono essere angoli, spigoli, zone ad alto contrasto visivo, ecc..). gli archi (u_1, u_2) rappresentano i vincoli odometrici e cinematici del robot in posizioni successive. Gli archi (c_1,c_2) rappresentano le chiusure di loop, ossia il fatto che il robot sia tornato in istanti successivi negli stessi punti dell'ambiente esplorato. Gli archi (v_1, v_2, v_3, v_4) rappresenta il fatto che i landmark vengano osservati in specifiche posizioni del robot, infine il nodo in rosso K rappresenta l'insieme dei parametri del sensore o dell'algoritmo che si occupa dell'estrazione dei landmark.

2.1.1 Sistemi basati su sensori laser e LIDAR

Storicamente, i primi sensori impiegati per lo SLAM (oltre ai sonar) sono stati i sensori laser. I sistemi SLAM basati su laser sono oggigiorno caratterizzati da una tecnologia consolidata e da una buona robustezza ottenuta anche grazie alla elevata affidabilità che caratterizza i sensori laser. I sensori laser più diffusi sono quelli a fascio planare e sono in grado di rilevare la presenza e la posizione degli ostacoli nell'ambiente in cui naviga il robot misurando l'angolo di emissione ed il tempo di volo del singolo raggio laser. Sfruttando queste misure, acquisite continuamente durante il movimento, è possibile costruire una mappa metrica planare dell'ambiente e contemporaneamente localizzare il robot stimando le coordinate x ed y (planari) e l'angolo di imbardata (yaw: γ). I metodi SLAM laser-based si basano quasi tutti su un approccio di tipo bayesiano e possono essere divisi in tre principali famiglie:

- Metodi basati su Extended Kalman Filter (EKF): ne è un esempio HectorSLAM (Kohlbrecher et al., 2011).
- Metodi basati su filtro particellare Rao-Blackwellized (RB Particle filter): come ad esempio FastSLAM (Montemerlo et al., 2002) (Montemerlo et al., 2003) ed il consolidato GMapping (Grisetti et al., 2007), a sua volta una versione che migliora il sopracitato FastSLAM.
- Metodi basati su grafi: ad esempio KartoSLAM (Konolige et al., 2010)

Lo svantaggio principale dell'utilizzo di tecniche basate su sensori laser 2D è proprio il fatto che la mappatura e la localizzazione si limita ad aspetti e features di tipo planari. Non è infatti possibile stimare la quota, gli angoli di rollio (roll) e di beccheggio (pitch) del robot, e non è possibile mappare ambienti che si "sviluppano" in altezza o su più piani. Questa limitazione può essere risolta utilizzando laser 3D, ma il loro costo è sensibilmente più elevato. Inoltre, in genere, l'output di sistemi basati su laser si limita a fornire informazioni relativamente alla presenza di ostacoli mentre risulta molto complesso, se non impossibile, estrapolare caratteristiche visuali più strutturate come le categorie degli oggetti o le loro relazioni.

Durante le prime fasi del progetto Smart Road sono state individuate due principali tipologie di metodi di localizzazione che utilizzano misure provenienti da sensori LIDAR (Laser Imaging Detection and Ranging). La prima tipologia sfrutta unicamente i dati provenienti da questo sensore, mentre una seconda tipologia si basa sulla combinazione di dati raccolti con il LIDAR e quelli raccolti da sensori inerziali (cioè sensori provenienti dalla Inertial Measurement Unit - IMU). In particolare, tra gli approcci appartenenti alla prima tipologia (senza IMU), sono stati testate le seguenti varianti del metodo LOAM (LIDAR Odometry and Mapping) (Zhang and Singh, 2014):

- FLOAM (Fast LOAM): versione ottimizzata del metodo LOAM (Zhang and Singh, 2014)
- Lego-LOAM (Lightweight and Ground-Optimized LOAM): approccio basato su LOAM che, oltre ad implementare algoritmi di loop closure, è anche computazionalmente più efficiente e specificatamente ottimizzato per l'utilizzo su ambienti con terreni accidentati (Shan and Englot, 2018).

Tra i metodi più recenti, che sfruttano le misure provenienti dal sensore inerziali, è stato testato il metodo LIO-SAM (Shan et al., 2020), anch'esso basato sull'implementazione di LeGO-LOAM: la presenza del sensore inerziale permette una stima della posizione del robot più precisa, in particolare in quegli scenari in cui il robot percorre traiettorie che non sono planari, come ad esempio negli ambienti outdoor. Altri approcci interessanti di tecniche di SLAM basate su LIDAR sono:

- IMLS-SLAM (Deschaud, 2018) - che implementa un algoritmo di SLAM low-drift tramite un framework che permette di elaborare i dati 3D del LIDAR con una corrispondenza detta scan-to-model. L'idea è la stessa alla base di Kinect Fusion (Newcombe et al., 2011a), ma con il LIDAR al posto dello scanner 3D a luce strutturata presente nel dispositivo Microsoft.
- Cartographer (Hess et al., 2016) - un sistema SLAM sviluppato da Google. Questo algoritmo in particolare implementa un sistema di loop-closure (ossia una strategia di riconoscimento dei punti della mappa esplorati più di una volta) e può essere utilizzato con una gran varietà di piattaforme e sensori, permettendo anche di lavorare con mappe 2D e 3D.

2.1.2 Approcci per lo SLAM basati su Sistemi di visione

A differenza di sensori di tipo Laser o LIDAR, i sensori di visione sono molto più economici e allo stesso tempo estremamente informativi. Le immagini, infatti, possono essere processate per estrarre elementi ad alto contenuto semantico.

La localizzazione del robot può essere fatta impiegando flussi video di una singola telecamera (in questo caso si parla di metodi di visual SLAM monoculari) o attraverso una coppia di telecamere la cui acquisizione dei fotogrammi avviene in maniera sincrona (in questo caso si parla di metodi di visual SLAM stereo).

Nel primo caso il costo dell'hardware è ridotto al minimo, ma non è possibile ottenere una stima della scala assoluta della traiettoria del robot e degli elementi della mappa. Nel secondo caso, a fronte di un hardware più costoso ed un software leggermente più complesso, corrisponderà una stima della traiettoria e della mappa metrica più precisa.

Attualmente, i metodi di localizzazione e mapping possono essere divisi in due tipologie principali: i metodi model-based, che sfruttano le proprietà e le relazioni geometriche degli oggetti presenti nella stanza, ed i metodi data-driven. In quest'ultimo caso, le features che descrivono la mappa e che consentono la localizzazione del robot vengono calcolate da una rete neurale convoluzionale (Convolutional Neural Network, CNN), opportunamente addestrata.

I metodi model-based possono essere di varie tipologie a seconda di come vengono utilizzate le informazioni contenute nelle immagini:

- Metodi sparsi: si definiscono così i metodi che non sfruttano tutti i pixel dell'immagine, ma si concentrano su punti di interesse all'interno dell'immagine (keypoints), associati a delle codifiche che ne descrivono le caratteristiche (descrittori). Le coppie keypoint+descrittore sono genericamente dette features e molti sono gli algoritmi in grado di estrarle a partire da immagini,

ad esempio ORB (Mur-Artal et al., 2015) o altri tipi di approcci (Zhang et al., 2015)). Una volta estratte le features, queste vengono triangolate nel mondo 3D (Mur-Artal and Tardos, 2017) o (Schlegel et al., 2017) - e vengono quindi tracciate (tracking) nel susseguirsi delle immagini del video. I keypoints, una volta posizionati nello spazio 3D, concorrono a costituire la mappa metrica dell'ambiente che viene utilizzate per localizzare il robot e stimarne la traiettoria percorsa. Architetture come quelle implementate in OpenVSLAM (Sumikura et al., 2019) supportano nativamente differenti tipi di camere, sia stereo che mono, anche con obiettivi di tipo fisheye o equirettangolari. I metodi più recenti permettono di sfruttare, oltre ai dati provenienti dal sistema di visione, anche le misure raccolte dal sensore inerziale (Campos et al., 2020), per una stima della posa del robot ancora più precisa.

- Metodi semidensi: metodi che utilizzano direttamente i pixel dell' immagine, senza ricavare da essa delle features caratteristiche. In questo aspetto sono simili ai metodi densi, con la differenza che, rispetto a questi ultimi, non vengono utilizzati tutti i pixel dell'immagine bensì solo una parte di essi. Gli approcci forse più famosi che rientrano in questa tipologia di algoritmi sono Large-Scale Direct Monocular SLAM (LSD-SLAM) (Engel et al., 2014) e la sua estensione ad applicazioni con telecamere stereo (Engel et al., 2015), Direct Sparse Odometry (DSO) (Engel et al., 2016) e Semi-direct Visual Odometry (SVO) (Forster et al., 2016).
- Metodi densi: come i metodi semidensi, anche questi non utilizzano features sparse, bensì tutti i pixel presenti all'interno dell'immagine per stimare lo spostamento del robot. Alcuni esempi sono Dense Tracking and Mapping (DTAM) (Civera et al., 2008) (Newcombe et al., 2011b), Multi-Level Mapping SLAM (MLM-SLAM) (Greene et al., 2016) e SOFT-SLAM (Cvišić et al., 2017). La maggior parte di questi metodi tuttavia fanno uso di camere di tipo RGB-D, ovvero in grado di fornire una mappa di profondità (depth-map) associata all'immagine RGB contenente le distanze tra ogni pixel e il relativo punto nello spazio 3D.

I metodi data-driven si differenziano a seconda del compito che viene affidato alla rete neurale. Similmente agli approcci model- based, anche reti neurali profonde possono essere impiegate per individuare ed estrarre features dall'immagine e permettere quindi di descrivere tramite queste la mappa dell'ambiente e di effettuare la localizzazione del robot (Yang et al., 2016) (Yi et al., 2016) (Tang et al., 2019) (Chaplot et al., 2020). In alternativa, le reti neurali convoluzionali vengono impiegate per ottenere immagini segmentate (in cui ogni pixel è associato ad un categoria specifica di oggetti associate alle varie regioni segmentate) al fine di costruire la mappa (Salas-Moreno et al., 2013), per stimare la depth-map (Tateno et al., 2017) o ricostruire direttamente il movimento del robot (Konda and Memisevic, 2015) (Costante et al., 2015).

2.2 Sistemi di navigazione

In questo paragrafo verranno illustrate alcune tecniche a stato dell'arte per quanto riguarda gli approcci per la navigazione dei veicoli autonomi. Affinché un robot sia in grado di navigare in maniera sicura in un ambiente non noto a priori senza l'intervento diretto di un operatore, un moderno sistema di navigazione deve essere in grado di capire dove si trova il veicolo e deve essere in grado di controllare i suoi movimenti, conducendolo fino al punto di arrivo prefissato senza andare in collisione con i possibili ostacoli presenti nell'ambiente circostante. Per fare ciò devono essere risolti tre principali problemi:

- la localizzazione del robot: viene risolta attraverso uno degli algoritmi SLAM analizzati nei paragrafi precedenti;
- la percezione degli ostacoli presenti nell' ambiente. L' insieme degli ostacoli percepiti dal robot andranno a creare una mappa che potrà essere utilizzata per creare traiettorie libere da collisioni;
- la generazione di una traiettoria che conduce il robot fino al punto di arrivo. Tale traiettoria deve evitare gli ostacoli e deve essere percorribile dal robot, deve cioè rispettare i vincoli cinematici del veicolo.

2.2.1 Percezione degli ostacoli (Obstacles Map)

Per quanto riguarda la generazione delle mappe contenenti gli ostacoli rilevati dal robot, le strategie più consolidate in letteratura prevedono la creazione di una occupancy grid (Thrun, 2003) (Birk, and Carpin, 2006), ossia una griglia metrica 2D che discretizza l'ambiente circostante al robot, dove ciascuna cella può assumere tre stati, lo stato "occupato", se la rispettiva "porzione" di ambiente è occupata da un ostacolo, lo stato "libero", se la zona è attraversabile dal robot, e lo stato "ignoto" se tale porzione di ambiente non è stata ancora esplorata. Estensione di tale strategia è rappresentata da OctoMap (Hornung et al, 2013), metodo che permette di creare una mappa degli ostacoli in uno spazio 3D. Tale strategia, benché necessiti di un maggiore costo computazionale, risulta molto più flessibile, in quanto in grado di modellare un ambiente senza dover porre alcuna assunzione su di esso, al contrario una occupancy grid 2D può essere utilizzata solamente nel caso di un ambiente che si estende principalmente in maniera "planare".

2.2.2 Generazione e controllo delle traiettorie (Path Planning)

Lo studio della generazione e controllo delle traiettorie (o path planning) è un ambito di ricerca molto esplorato non solo in robotica mobile (in quanto per l'appunto componente fondamentale del sistema di navigazione di un veicolo autonomo), ma anche in altre aree della robotica, come la robotica medicale e la robotica industriale. Strategie ampiamente consolidate per affrontare il problema del path planning prevedono lo sfruttamento di strutture dati a grafo o ad albero, come ad esempio approcci basati su algoritmi A-star (A*) (Rivera, Baier, Hernandez, 2013), (Ducho, 2014) , Time Enhanced A-star (TEA*) (Santos, 2014) e Rapidly exploring Random Tree (RRT) (Xu, Dolan, 2014). Questo tipo di strategie prevedono la costruzione di un grafo che rappresenti la discretizzazione dello spazio circostante il robot, con il conseguente calcolo del percorso a costo minimo (attraverso un algoritmo di ottimizzazione) che metta in connessione il nodo rappresentante la posizione iniziale del robot ed il traguardo desiderato. I principali limiti di molti di questi metodi riguardano la necessità di una mappa degli ostacoli fissa e nota a priori ed il fatto che non è possibile gestire la cinematica del robot. Questi fattori limitano di fatto l'applicabilità degli algoritmi con robot che possiedono specifici vincoli cinematici ed in scenari con ambienti dinamici e non noti a priori. Una possibile soluzione a queste problematiche è rappresentata dagli approcci Sampling-based (Lau, Sprunk, and Burgard, 2009). I metodi Sampling-based prevedono la generazione, eventualmente anche in maniera periodica, di centinaia di traiettorie possibili verso il punto di arrivo. Le traiettorie generate vengono ordinate secondo una policy specifica (come ad esempio il tempo di percorrenza), andando quindi a selezionare quella che risulta migliore. Questo tipo di metodi sono in grado di gestire mappe dinamiche degli ostacoli, inoltre strategie più recenti di questa tipologia, come ad esempio Timed Elastic Band (TEB) (Rösmann, Hoffmann and Bertram, 2017), prevedono la generazione di traiettorie che tengono conto dei vincoli cinematici del robot, risultando particolarmente efficienti ed efficaci nell'applicazione pratica. Varianti dei metodi sampling-based prevedono anche l'utilizzo di algoritmi genetici (L. Changan, Y. Xiaohu, L. Chunyang, and L. Guodong,2010) per la generazione e l'ottimizzazione delle traiettorie. Attualmente un'area di studio emergente risulta essere la trattazione del problema del path planning attraverso strategie di Reinforcement Learning (Xie, et al. 2021). Metodi di questo tipo prevedono la creazione di un modello che impari a generare e controllare le traiettorie attraverso una fase di addestramento che prevede l'utilizzo di un segnale di "reward", ossia una funzione che dipende dal modo in cui avviene l'interazione con l'ambiente circostante e che incoraggi i comportamenti corretti. Questo tipo di algoritmi hanno il principale vantaggio di poter essere utilizzati senza dover porre assunzioni sull'ambiente di lavoro o sulla tipologia di robot utilizzato.

3 Descrizione delle attività svolte e risultati

In questa sezione verranno illustrate le attività svolte relativamente alla implementazione e validazione dei vari algoritmi di localizzazione e navigazione testati. Per quanto riguarda la localizzazione in ambiente urbano del veicolo, si è optato per un approccio di sensor fusion, che permette di utilizzare contemporaneamente informazioni estratte da diverse tipologie di sensori al fine di ottenere un risultato più accurato. Per rendere più agevole e veloci le fasi di sviluppo e di sperimentazione preliminari degli algoritmi, è stato utilizzato un ambiente simulato che permette di ricreare ambienti 3D urbani piuttosto fedeli. Le caratteristiche di tale ambiente saranno descritte nei paragrafi successivi.

3.1 Implementazione della fusione sensoriale per la localizzazione

Il problema della localizzazione consiste in prima istanza nello stimare la posizione e l'orientamento del robot all'interno dell'ambiente per mezzo delle informazioni estratte dai sensori a sua disposizione. Allo stesso tempo il sistema di localizzazione deve anche fornire una misura dell'incertezza di tale stima.

Nelle tecniche di fusione sensoriale (sensor fusion) vengono processate contemporaneamente le stime di posizione ed orientamento del robot e della relativa incertezza nella stima (fornite dai differenti algoritmi di localizzazione impiegati e dove ciascuno di essi può far uso di sensori e di tecniche specifiche) al fine di produrre una singola stima più accurata.

Idealmente, la stima fornita dalla sensor fusion dovrebbe non solo essere di miglior qualità, quindi avere un'incertezza minore, ma dovrebbe essere più robusta. Questo perché tramite questa tecnica è possibile valutare dinamicamente la stima prodotta da ciascun sensore sulla base della sua incertezza e pesare quindi in modo differente le stime di posizione ed orientamento in differenti situazioni. In questo modo quando una tecnica fornisce una stima non particolarmente affidabile in condizioni a lei sfavorevoli (ad esempio una tecnica di visione in un ambiente con scarsa illuminazione), l'incertezza associata a questa stima è maggiore ed il suo contributo verrà quindi pesato in modo meno rilevante nel computo (fusione delle stime) della stima complessiva. Questa tecnica di fusione viene implementata tramite l'utilizzo di un filtro di Kalman esteso (EKF), una versione non lineare del filtro di Kalman che linearizza la stima ad ogni passo.

L'EKF è un approccio alla localizzazione estremamente consolidato ed è considerato oggi lo standard in molti contesti (Moore and Stouch, 2014).

Nel caso specifico del presente progetto di ricerca il vettore di stato considerato per la stima prende in considerazione solamente le componenti sul piano orizzontale per la posizione e l'angolo di imbardata per l'orientamento, trascurando durante questa prima fase della sperimentazione la componente relativa alla altezza (quota), gli angoli di rollio e beccheggio, e le relative velocità. Il vettore di stato considerato è dunque costituito come segue: $[x, y, \gamma, \dot{x}, \dot{y}, \dot{\gamma}]$.

3.2 Implementazione dello stack di navigazione e dell'interfaccia con il simulatore

La strategia di navigazione utilizzata nel progetto Smart Road è basata sulle metodologie messa a disposizione nel Navigation Stack di OSRF (Open Source Robot Foundation) (Marder-Eppstein et al., 2010). In figura 3 è riportata l'architettura del sistema di navigazione a cui si fa riferimento.

La tecnica OSRF permette di elaborare autonomamente dei percorsi privi di ostacoli a partire dalla mappa generata dalle tecniche SLAM descritte nel capitolo precedente. Questo metodo è particolarmente efficiente in quanto utilizza mappe planari generate da sensori laser permettendo inoltre la gestione di ostacoli dinamici ricalcolando in tempo reale la traiettoria.

La tecnica OSRF utilizza due tipologie di mappe, chiamate costmap: una mappa globale, che è quella fornita dal modulo SLAM e che viene usata per pianificare il percorso ottimo ad alto livello, ed una mappa locale, che viene invece calcolata in linea tramite a partire da misure di un sensore laser. Quest'ultima ha una dimensione molto inferiore, e viene utilizzata per evitare eventuali ostacoli dinamici o non presenti sulla mappa SLAM. Tipicamente si avvale dell'impiego di informazioni odometriche (fornite ad esempio da encoder), per tenere traccia degli spostamenti del robot all'interno dell'ambiente, ma è possibile affiancargli anche tecniche di localizzazione più avanzate al fine di evitare che gli errori di drift introdotti dall'odometria rendano la posizione del robot nella mappa poco accurata e causino di conseguenza errori nella pianificazione del percorso e nella navigazione. È oggigiorno disponibile una versione più aggiornata dello stack di navigazione di OSRF (Macenski et al., 2020), che implementa algoritmi più efficienti in termini di costo computazionale. Nella nuova versione, infatti, è stato aggiunto il behavior tree come nucleo centrale dell'intero stack di navigazione. Questo componente ha un elevato grado di customizzazione e permette di configurare in modo puntuale il comportamento del robot durante la navigazione ed ottimizzare il modo in cui dovrà comportarsi per recuperare il percorso pianificato in caso di errore.

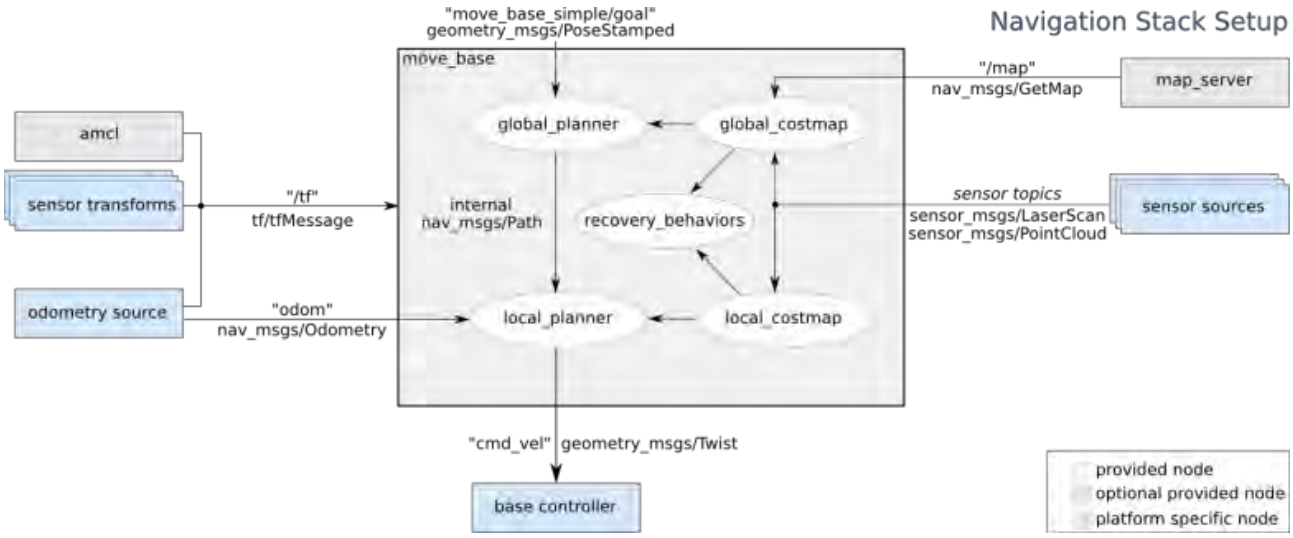

Figura 3. Architettura dello stack di navigazione

Esistono inoltre degli algoritmi di navigazione che permettono di muoversi autonomamente ed evitare ostacoli all'interno di ambienti 3D. Ad esempio il pacchetto MoveIt! (Coleman et al., 2014) permette di implementare la navigazione 3D tramite la libreria OMPL (Open Motion Planning Library) (Şucan et al., 2012).

In figura 4 è mostrato il diagramma a blocchi dell'ambiente utilizzato per effettuare le sperimentazioni che saranno analizzate in dettaglio nel prossimo capitolo.

Per supportare lo sviluppo dello stack software si è utilizzato un ambiente di simulazione che permette di riprodurre una versione simulata e semplificata della piattaforma robotica; nello specifico il simulatore permette di simulare la cinematica del veicolo, considerando inerzie ed attriti a cui il robot viene normalmente sottoposto durante il moto. Inoltre è possibile dotare il veicolo di una set di sensori virtuali che riproducono fedelmente la risposta dei sensori fisici che verranno impiegati nel controparte reale. In particolare negli esperimenti effettuati sono stati utilizzati sensori LIDAR e laser, stereo-camera, encoder delle ruote ed il GPS. Inoltre, il software di simulazione consente di ricreare ambienti fotorealistici molto fedeli agli scenari di utilizzo effettivo del robot. L'utilizzo del simulatore ha, pertanto, un duplice vantaggio:

- permette di sviluppare e testare algoritmi di localizzazione e navigazione evitando i costi (in termini di tempo e di materiali) e i rischi legati a sperimentazioni in ambienti reali;
- permette di raccogliere immagini e dati che possono essere utilizzati come dataset per l'addestramento di modelli di deep learning che possono essere eventualmente utilizzati in fasi successive.

In particolare il sistema sviluppato è composto da tra parti principali:

- il simulatore Unreal Engine. Unreal Engine è un ambiente pensato principalmente per lo sviluppo dei videogiochi, ma la possibilità di riprodurre in modo fedele la cinematica e dinamica del robot, la fedeltà delle risposte della strumentazione virtuale unitamente alla disponibilità di ambienti fotorealistici lo rende anche adatto ad essere utilizzato come ambiente di simulazione per sistemi robotici;
- l'ecosistema ROS: Robot Operating System (ROS) è un framework pensato per lo sviluppo di sistemi software per la robotica. ROS mette a disposizione protocolli di comunicazioni e strumenti di visualizzazione che permette la comunicazione di diversi moduli software che operano in tempo reale su computer e robot differenti connessi alla stessa rete locale;

- l'interfaccia Unreal-ROS: si tratta di una serie di moduli software sviluppati dal gruppo di ricerca Intelligent Systems, Robotics and Automation Laboratory (ISARLab) e permette la comunicazione tra il simulatore Unreal e l'ecosistema ROS.

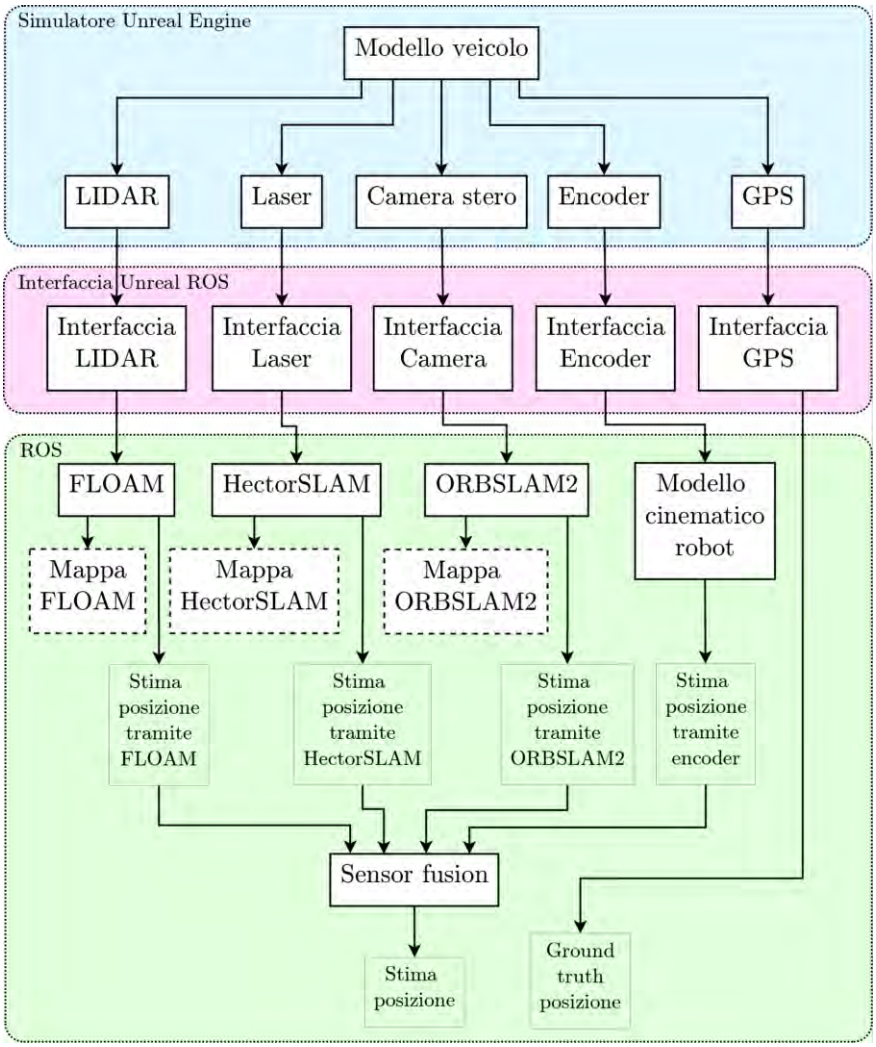

Figura 4. Schema a blocchi del sistema di navigazione implementato e dell'interfaccia verso il simulatore

Come già anticipato all'interno del simulatore è possibile "assemblare" e "sensorizzare" il veicolo robotico e l'ambiente in cui il robot si troverà ad operare. Al veicolo in questione vengono quindi associati i sensori che si intende utilizzare per la localizzazione ed il mapping durante i vari esperimenti; nel nostro caso lidar, laser, stereocamera ed encoder. In aggiunta, ai sensori appena citati, durante gli esperimenti viene anche utilizzato il sensore GPS, in grado di fornire la posizione spaziale ed angolare del robot. Negli studi effettuati Il sensore GPS non è stato effettivamente impiegato per la localizzazione ma è stato usato come utile riferimento per il calcolo delle performance. Durante la simulazione Unreal Engine mette a disposizione un canale virtuale di trasmissione, tramite socket TCP/UDP, su cui trasmette tutti i dati provenienti dai sensori: scansioni dei sensori laser e lidar, immagini della stereocamera, letture degli encoder e posizione del sensore GPS. Il compito dell'interfaccia Unreal-ROS è quindi quella di utilizzare il socket TCP/UDP per raccogliere i dati provenienti dai sensori simulati e ri-trasmetterli su topic ROS, rendendo così i dati disponibili per tutti i nodi in esecuzione all'interno del framework ROS. All'interno dell'ambiente ROS vengono eseguiti quattro nodi ROS che implementano altrettanti modelli di localizzazione:

- Fast Lidar Odometry And Mapping (FLOAM): sistema di localizzazione in grado di costruire una mappa 3D dell' ambiente utilizzando le scansioni del sensore lidar, per poi eseguire la localizzazione del robot all'interno della mappa.
- HectorSLAM: Sistema SLAM in grado di costruire una mappa 2D planare dell'ambiente usando le scansioni laser.
- ORBSLAM2: Approccio di SLAM che si basa sull'utilizzo della stereocamera, in grado di estrarre features ORB dalle immagini, eseguire matching e triangolazione e di conseguenza generare una mappa 3D delle features che verrà poi usata per la localizzazione del robot.
- Odometria tramite encoder: il modello cinematico del robot viene utilizzato per generare la traiettoria percorsa dal veicolo usando le letture degli encoder presenti sulle ruote.

Gli output generati dai quattro algoritmi possono essere combinati attraverso un algoritmo di fusione sensoriale. L'uso della fusione sensoriale permette di mantenere la localizzazione molto precisa anche quando uno dei quattro algoritmi sopra menzionati sbaglia (ad esempio a causa di immagini scure, misure dei sensori rumorose o errate, ecc.), minimizzando così l'errore totale di stima della posizione.

3.3 Sperimentazione in ambiente simulato (esperimenti con Unreal Engine)

In questa sezione verranno illustrate le modalità con le quali sono stati svolti i vari esperimenti e come sono stati messi a confronto tra loro i diversi algoritmi di mapping e localizzazione (SLAM). In generale è stato fatto muovere un robot all'interno di un ambiente simulato in contesti differenti e sono state raccolti i dati e le informazioni provenienti da tutti i sensori a bordo, registrando quindi le varie sequenze. Gli stream di dati registrati sono state quindi processati in diverse modalità di funzionamento, sia singolarmente sia in diverse combinazioni utilizzando algoritmi di fusione sensoriale. Nei seguenti paragrafi saranno analizzati in maggior dettaglio l'ambiente di simulazione e descritti gli ambienti 3D in cui sono state eseguite le sperimentazioni.

3.3.1 Ambiente di simulazione

Le motivazioni alla base della scelta dell'ambiente di simulazione sono guidate dalle possibilità e flessibilità che l'ambiente mette a disposizione per svolgere gli esperimenti sopra descritti. Il software scelto per effettuare le sperimentazioni è Unreal Engine 4, un motore grafico sviluppato da Epic Games pensato principalmente per lo sviluppo di videogiochi. Questo simulatore mette a disposizione un ambiente graficamente fotorealistico con un'ottima gestione di luci ed ombre dinamiche ed un motore fisico sufficientemente preciso, così da permettere anche prove soddisfacenti di guida con il modello dinamico di un veicolo. Il motore grafico fornisce inoltre un editor completo per permettere agli utenti di creare mappe, ambienti, modelli ed oggetti estremamente dettagliati e gode di un esteso supporto da parte della comunità online che mette a disposizione modelli di elementi di costruzione o d'arredo, veicoli od ambienti di varie tipologie che rendono possibile l'implementazione di scenari molto diversi tra loro in modo semplice.

Unreal Engine mette a disposizione inoltre una vasta gamma di personalizzazione tramite la possibilità di integrare plugin di terze parti. È stato infatti realizzato un plugin specifico che permette di estrarre dall'ambiente simulato informazioni dettagliate circa lo stato del dinamico del veicolo simulato e di posizionare a bordo del modello dei sensori virtuali come telecamere (monoculari e stereo), encoder sulle ruote, sensore inerziale (IMU), laser, lidar e la possibilità di pilotare il veicolo tramite comandi esterni. Tutti i sensori sono stati implementati con un modello matematico che introduce sul valore misurato i più comuni errori tipici della strumentazione reale e fornisce la covarianza sui valori stimati. Questo rende le simulazioni più fedeli alla realtà e limita parzialmente il problema di "idealità" dei dati prodotti da un ambiente simulato. Sono stati quindi applicati tali sensori a bordo del veicolo che è stato poi guidato con comandi manuali all'interno dell'ambiente simulato in modo da raccogliere i dati usati successivamente per testare e confrontare i vari algoritmi di localizzazione.

3.3.2 Scenari e sequenze raccolte

In questo paragrafo verranno illustrati gli scenari utilizzati per gli esperimenti. All'interno di questi ambienti virtuali è stato fatto navigare un modello realistico di veicolo e sono state raccolte le informazioni acquisite dai sensori a bordo del veicolo stesso. I dati raccolti sono stati quindi elaborati dai vari algoritmi di localizzazione, prima singolarmente e in seguito attraverso varie configurazioni facendo uso di un sistema di fusione sensoriale e sono stati quindi confrontati i risultati ottenuti su un totale di sei sequenze differenti, tre sequenze per ciascuno dei due ambienti utilizzati per le sperimentazioni.

Nello specifico sono stati predisposti due scenari di simulazione, il primo, UrbanCity, è uno scenario cittadino in cui viene ricostruito un quartiere di una città, mentre il secondo, Rural, è un ambiente rurale, caratterizzato da terreno collinare e dalla presenza di piantagioni, fattorie piccoli sentieri. Per ogni ambiente le sequenze sono state raccolte cercando di variare traiettoria percorsa, lunghezza della sequenza e condizioni di luminosità delle immagini. Le sequenze sono state raccolte seguendo la seguente procedura:

- viene inizializzato l'ambiente di simulazione, l'ecosistema ROS, i sistemi di localizzazione da analizzare, ed il tool di registrazione Rosbag per il salvataggio di tutti i dati sensoriali durante l'esperimento;
- il veicolo viene guidato all'interno dell'ambiente di simulazione, viene raccolta la traiettoria di ground truth (attraverso il sensore GPS + IMU che restituisce posizione spaziale e angolare attuale del robot) e quelle stimate dai sistemi di localizzazione;
- il bagfile ottenuto viene processato per combinare le stime dei vari algoritmi SLAM attraverso differenti configurazioni di fusione sensoriale;
- vengono registrati nuovi bagfile con le stime delle varie configurazioni di fusione sensoriale.

Attraverso i bagfile ottenuti è quindi possibile estrarre i campioni per il calcolo ed il confronto delle performance dei vari sistemi di localizzazione e degli algoritmi di fusione sensoriale. Nella figura 5 e 6 sono riportati due screenshot relativi all'interfaccia del software di simulazione Unreal Engine 4 e del software di visualizzazione ROS Rviz. In particolare, nello screenshot di Rviz è possibile notare l'immagine catturata dal punto di vista della telecamera (in basso a sinistra), l'output del metodo di localizzazione ORB-SLAM2 (in alto a sinistra) ed al centro sono mostrate le mappe ricostruite dai vari metodi SLAM e la posizione stimata del robot. La mappa planare in grigio è ottenuta dal metodo HectorSLAM, mentre La mappa 3D composta dai punti colorati è quella ottenuta con FLOAM, infine i punti grigi sono i key-frame estratti da ORBSLAM2 e posizionati nello spazio 3D. Infine le linee colorate sono le scansioni del sensore lidar.

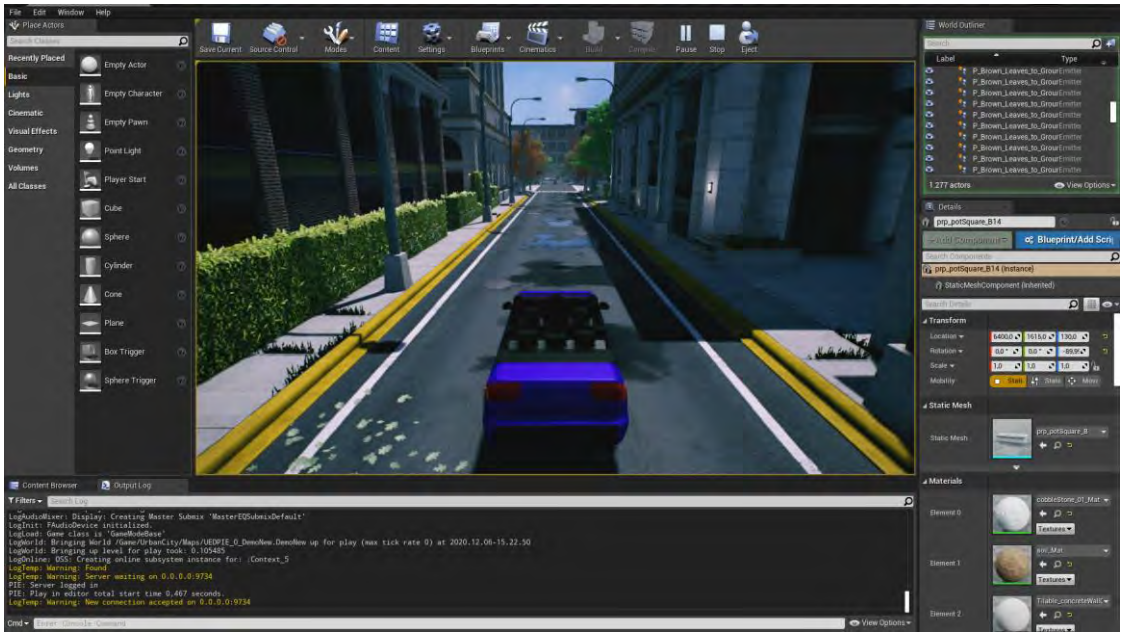

Figura 5. Interfaccia del Simulatore Unreal Engine 4

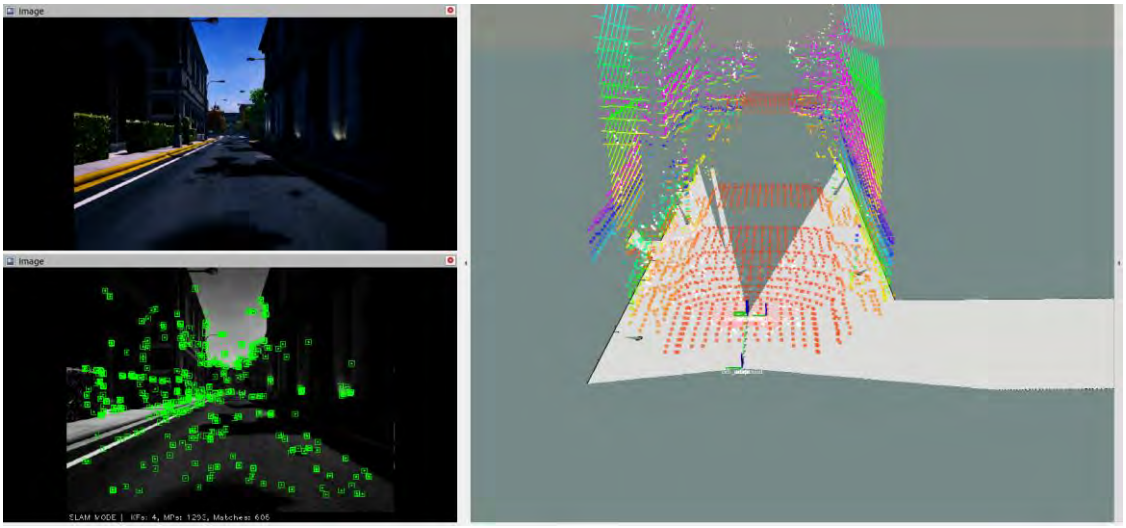

Figura 6. Schermata estratta dal software di visualizzazione Rviz

Ambiente UrbanCity

Lo scenario UrbanCity è un ambiente urbano che costituito da un quartiere cittadino, alcune strade ben definite, da edifici, un grosso incrocio ed aiuole con piante ed alberi.

Questo permette di testare gli algoritmi di localizzazione in ambienti piani e delimitati in modo piuttosto preciso per quanto gli spazi di lavoro siano comunque vasti e tipici di un ambiente outdoor. Questo scenario, in particolare, per quanto sia sensibilmente diverso dall'ambiente di lavoro dove dovrà operare il robot reale, mette a disposizione degli ottimi modelli ed un'ottima resa fotorealistica, è pertanto un ambiente ottimale per effettuare la messa a punto ed il test, in particolare la robustezza, degli algoritmi di visione. Nella figura 7 e 8 sono mostrate una vista dell'ambiente di simulazione UrbanCity ed una mappa ricostruita attraverso le scansioni del sensore lidar.

Figura 7. Immagine ottenuta dal simulatore Unreal Engine 4 dell'ambiente UrbanCity

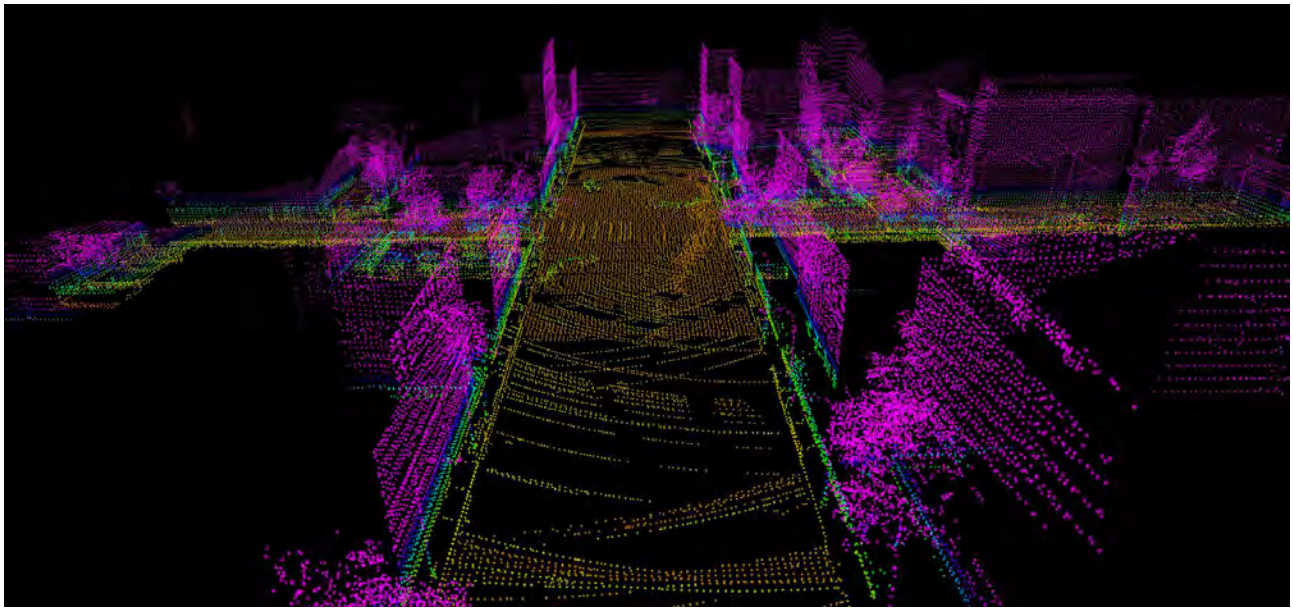

Figura 8. Mappa calcolata con il sensore lidar simulato nell'ambiente UrbanCity

Ambiente Rural

Lo scenario Rural è composto da un terreno prevalentemente collinare, da edifici che rappresentano fattorie e abitazioni rurali, da piantagioni e da strade e sentieri che richiamano appunto l'ambiente rurale. L'impiego dell'ambiente rural ha il principale vantaggio di permettere il test degli algoritmi di localizzazione in ambienti con differenti altitudini e terreni accidentati. Nelle figure 9 e 10 sono mostrate una vista dell'ambiente Rural ed una mappa ricostruita attraverso il sensore lidar.

Figura 9. Vista dell'ambiente di simulazione Rural

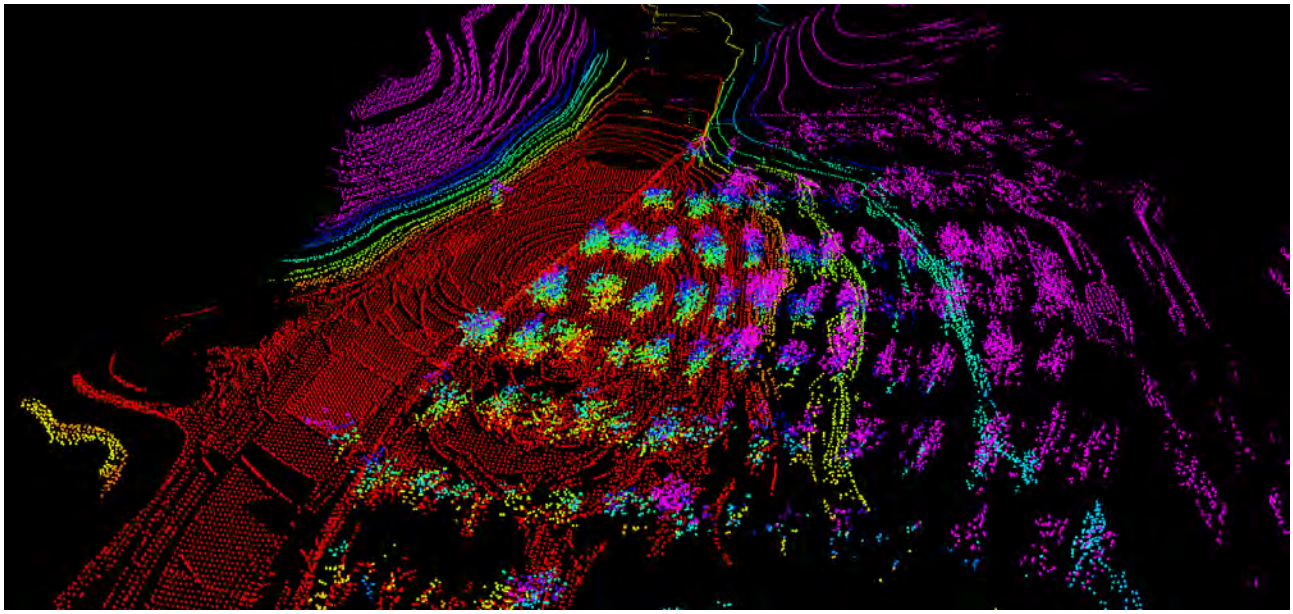

Figura 10. Mappa ottenuta tramite LIDAR simulato in ambiente Rural

3.3.3 Algoritmi analizzati

Sono state quindi confrontate le prestazioni di algoritmi di stima della posizione del robot utilizzando varie tecniche SLAM e varie configurazioni del metodo di fusione sensoriale. In particolare gli algoritmi di SLAM analizzati sono quelli a stato dell'arte della letteratura in questo momento per questo settore, che si differenziano principalmente in base alla tipologia di sensori utilizzati per la localizzazione.

- In particolare è stato utilizzato un algoritmo di fusione sensoriale basato su Extended Kalman Filter per fondere i diversi output prodotti dai metodi SLAM analizzati. I metodi di localizzazione utilizzati nel presente studio sono:
- HectorSLAM: Algoritmo di SLAM (Simultaneous localization and Mapping) che utilizza il sensore laser e si basa su metodi di stima della posa tramite Extended Kalman Filter.
- FLOAM: Algoritmo di SLAM che sfrutta il sensore LIDAR.
- ORBSLAM2: algoritmo di SLAM che sfrutta le immagini raccolte dalla stereocamera e che permette la localizzazione del robot attraverso l'estrazione di features ORB.
- Encoder: E' stata ricostruita, a scopo comparativo, anche la stima della posizione del robot basato sul calcolo dell'odometria ottenuta dagli encoder presenti sulle ruote del veicolo simulato.

Come anticipato, le stime ottenuti dai metodi semplici appena illustrati sono state aggregate per mezzo di algoritmi di fusione sensoriale, fondendo insieme i dati di diverse sorgenti fisiche. Sono state comparate le prestazioni di stima ottenute nelle seguenti configurazioni:

- HectorSLAM + ORBSLAM2 + FLOAM + Encoder
- HectorSLAM + ORBSLAM2 + FLOAM
- HectorSLAM + ORBSLAM2
- HectorSLAM + FLOAM

Le stime delle posizioni del robot ottenute da questi metodi sono state comparate con quelle "ideali" ottenute dal sensore GPS e dalla IMU (supposti noise free), sensori che, essendo simulati, non sono affetti da alcun tipo di rumore e pertanto sono in grado di fornire valori di ground truth accurati per quanto riguarda le posizioni spaziali ed angolari assunti dal robot durante il moto. Di seguito si riportano i link di due video che contengono esempi di esperimenti di localizzazione e mapping nei due scenari simulati (UrbanCity e Rural):

- https://www.youtube.com/watch?v=gzoDJuJj0BI
- https://www.youtube.com/watch?v=OGejWbkjnzg

In entrambi gli scenari è stata utilizzata la stessa configurazione: viene infatti eseguita la fusione della posa stimata tramite sensori odometrici, quella ottenuta tramite sistema di visione (con l'algoritmo ORBSLAM2, di cui si può vedere il risultato del processing delle immagini nella parte in alto a sinistra del video) e quella ottenuta tramite sensore LIDAR (attraverso l'algoritmo FLOAM, di cui è possibile vedere la mappa stimata nella parte a destra del video). In basso a destra è mostrata una vista dell'ambiente di simulazione mentre in basso a sinistra sono mostrate le immagini catturate dalla telecamera del veicolo.

3.3.4 Risultati

In questa sezione vengono illustrati i risultati ottenuti sulle sequenze raccolte nei due ambienti di test. La metrica utilizzata per il calcolo delle performance è l'errore di posa assoluto (Absolute Pose Error, APE). I valori di APE sono estratti per ogni campione della sequenza di riferimento, dopodichè è stato estratto il valore medio per l'intera sequenza e riportati nelle tabelle 1, 2, 3, 4, 5, 6.

Per tutte le sequenze e per ogni sensore simulato presente sul veicolo sono stati prodotti misure ad una frequenza di 5 Hz. Per quanto riguarda le lunghezze delle sequenze raccolte nell'ambiente UrbanCity, la sequenza UrbanCity-seq1 è lunga 333 secondi, la sequenza UrbanCity-seq2 332 secondi e UrbanCity-seq3 è lunga 734 secondi. Le lunghezze delle sequenze raccolte nell'ambiente Rural sono di 304 secondi (Rural-seq1), 275 secondi (Rural-seq2) e 669 secondi (Rural-seq3). Come già anticipato la metrica utilizzata per misurare le performance degli algoritmi analizzata è l'Absolute Pose Error (APE), in particolare data la posa attuale del robot (vera) $P_{ref,i}$ all'istante i e quella stimata $P_{est,i}$, è possibile ottenere il valore di APE E_i come:

$$E_i = P_{est,i}^{-1} P_{ref,i}$$

Di questo segnale di errore è possibile estrarne diversi valori statistici. Per gli esperimenti mostrati nei paragrafi seguenti in particolare si è deciso di utilizzare il valore medio e la deviazione standard calcolati come:

$$Mean = \frac{1}{N}\sum_{i=1}^{N} E_i$$

$$Std = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(E_i - Mean)^2}$$

I parametri utilizzati per i vari algoritmi di localizzazione sono quelli consigliati dagli autori dei paper che presentano i metodi che abbiamo analizzato negli esperimenti presentati nei seguenti paragrafi. Per quanto riguarda i tempi di processamento, ORBSLAM2 è in grado di processare un immagine ad una risoluzione di 800 X 600 pixel in circa 120 millisecondi (tempistiche in linea con quelle calcolate in (Aldegheri, et al., 2019)). Tra i metodi analizzati risulta essere quello con tempi di calcolo maggiori, FLOAM infatti possiede un tempo di calcolo per singola scansione del sensore lidar di circa 55 millisecondi e di circa 20 millisecondi per HectorSLAM. Queste tempistiche sono legate, oltre che alla complessità degli algoritmi, anche al contenuto informativo delle misure raccolte dai vari sensori, che risulta molto alto nel caso della stereocamera (utilizzata da ORBSLAM2), un po' più basso nel caso del LIDAR (utilizzato da FLOAM) e minimo nel caso del laser scanner 2D (utilizzato da HectorSLAM).

Risultati con ambiente UrbanCity

I risultati ottenuti nell'ambiente UrbanCity sono riportati nelle tabelle 1, 2 e 3. In questo particolare ambiente urbano le strade si estendono in maniera planare, favorendo i sistemi di localizzazione di tipo bidimensionale, come ad esempio HectorSLAM. Nella sequenza UrbanCity-seq1 (i risultati sono mostrati

nella tabella 1) infatti, HectorSLAM ottiene l'errore minore, mentre FLOAM ed ORBSLAM2 ottengono errori più alti. L'odometria degli encoder risulta avere le performance peggiori, sia in questa che in altre sequenze. Tra le configurazioni del sistema di fusione, la migliore, per questa sequenza, risulta essere quella che combina HectorSLAM e FLOAM, date le già ottime performance della localizzazione laser. La configurazione peggiore risulta essere quella che combina tutti i sistemi di localizzazione a causa delle pessime performance fornite dall'algoritmo di odometria degli encoder. Per le altre due sequenze (UrbanCity-seq2 e UrbanCity-seq3) i risultati, (riportati rispettivamente nella tabella 2 e 3) sono risultati sensibilmente diversi. In questi due casi infatti, HectorSLAM ha fornito un valore di APE tra i più alti. Valori piuttosto alti dell'errore vengono restituiti anche da FLOAM e dall'odometria degli encoder, mentre il sistema di localizzazione che ottiene la performance migliori è ORBSLAM2. Tra le configurazioni di fusione sensoriale, quella che ottiene il valore di APE più basso nelle due sequenze, è quella che combina HectorSLAM e ORBSLAM2. La configurazione di sensor fusion peggiore risulta essere quella che combina i sensori laser e lidar, a causa delle pessime performance di entrambi i metodi SLAM in questi esperimenti. Le performance ottenute dai sistemi di localizzazione in queste due sequenze sono molto probabilmente dovuti ad una maggiore velocità, complessità e lunghezza delle sequenze (in particolare la sequenza UrbanCity-seq3 è quella più lunga e con il maggior numero di curve, la lunghezza di UrbanCity-seq3 è infatti di 734 secondi, mentre quella di UrbanCity-seq1 e di UrbanCity-seq2 è rispettivamente di 333 e 332 secondi). La maggiore velocità del veicolo comporta due diversi problemi, il primo è che i campioni raccolti dai sensori variano molto velocemente tra istanti successivi, causando un veloce aumento della covarianza della posa del robot stimata da FLOAM e HectorSLAM. Inoltre una maggiore velocità può provocare lo slittamento delle ruote (che fa aumentare l'errore degli encoder) e porta il veicolo ad imbarcarsi, aumentando l'errore dei sistemi SLAM che non calcolano gli angoli di pitch e roll del robot, come fanno per l'appunto FLOAM e HectorSLAM. In definitiva ORBSLAM2 risulta quindi il metodo più robusto, riuscendo a stimare in maniera precisa posizione angolare e spaziale del robot anche con immagini molto diverse tra loro ad istanti successivi, inoltre la stima precisa dei tre angoli di roll, pitch e yaw permette di ottenere un mapping preciso e di conseguenza un basso errore di APE.

Tabella 1. Performance sulla sequenza UrbanCity-seq1, frequenza di acquisizione 5 Hz, lunghezza 333 s

Configurazione	**Mean APE (m)**	**APE Std (m)**
HectorSLAM	0.06	0.034
ORBSLAM2	0.61	0.459
FLOAM	1.40	1.024
Odometry	2.59	1.281
HectorSLAM+ OrbSLAM2 + FLOAM + Odometry	1.34	1.039
HectorSLAM+ OrbSLAM2 + FLOAM	0.94	0.426
HectorSLAM+ OrbSLAM2	0.93	0.422
HectorSLAM + FLOAM	0.30	0.229

Tabella 2. Performance sulla sequenza UrbanCity-seq2, frequenza di acquisizione 5 Hz, lunghezza 332 s

Configurazione	Mean APE (m)	APE Std (m)
HectorSLAM	2.34	3.063
ORBSLAM2	0.32	0.202
FLOAM	2.04	1.895
Odometry	3.07	1.645
HectorSLAM+ OrbSLAM2 + FLOAM + Odometry	1.97	1.894
HectorSLAM+ OrbSLAM2 + FLOAM	0.615	0.195
HectorSLAM+ OrbSLAM2	0.614	0.195
HectorSLAM + FLOAM	2.45	2.966

Tabella 3. Performance sulla sequenza UrbanCity-seq3, frequenza di acquisizione 5 Hz, lunghezza 734 s

Configurazione	Mean APE (m)	APE Std (m)
HectorSLAM	22.46	18.826
ORBSLAM2	0.64	0.254
FLOAM	4.51	4.141
Odometry	12.84	10.556
HectorSLAM+ OrbSLAM2 + FLOAM + Odometry	4.406	4.121
HectorSLAM+ OrbSLAM2 + FLOAM	0.934	0.345
HectorSLAM+ OrbSLAM2	0.932	0.348
HectorSLAM + FLOAM	22.723	18.702

Risultati con ambiente Rural

I risultati ottenuti nell'ambiente Rural sono riportati nelle tabelle 4, 5 e 6. In questo particolare ambiente rurale il territorio è prevalentemente collinare, con strade e sentieri che si sviluppano a diverse quote. Per questo motivo solamente i metodi che stimano la posa del robot in uno spazio tridimensionale riescono ad ottenere delle performance ragionevoli. In particolare si osserva che, tra i sistemi di localizzazione confrontati, in tutte e tre le sequenze il metodo che ottiene le performance peggiori è proprio HectorSLAM, proprio a causa del fatto che questo metodo stima la posa del robot supponendo il moto planare. I valori APE ottenuti dal metodo FLOAM sono anche essi piuttosto alti, principalmente a causa del fatto che tale metodo non è in grado di stimare gli angoli di pitch e roll in maniera precisa, portando ad una ricostruzione errata della mappa e di conseguenza ad una localizzazione affetta da errori significativi. Anche L'odometria calcolata attraverso gli encoder risulta essere affetta da una significativa componente di rumore, questo fatto è causato dall'impossibilità di stimare, attraverso il solo modello cinematico, la quota del robot o

angoli diversi da quello di yaw. L'unico modello in grado di stimare in maniera precisa mappa e posizione del robot è risultato essere l'ORBSLAM2.

Passando a valutare le varie configurazioni di fusione sensoriale analizzate le migliori sono risultate essere quelle che coinvolgono ORBSLAM2 ed escludono l'odometria calcolata con gli encoder, grazie proprio alle ottime performance del sistema visuale. La configurazione peggiore di sensor fusion risulta essere quella che coinvolge HectorSLAM e FLOAM, proprio a causa dei grandi errori di stima presenti in entrambi i metodi.

Tabella 4. Performance sulla sequenza Rural-seq1, frequenza di acquisizione 5 Hz, lunghezza sequenza 304 s

Configurazione	**Mean APE (m)**	**APE Std (m)**
HectorSLAM	5.98	1.777
ORBSLAM2	0.26	0.211
FLOAM	9.17	5.498
Odometry	4.41	2.866
HectorSLAM+ OrbSLAM2 + FLOAM + Odometry	9.04	5.619
HectorSLAM+ OrbSLAM2 + FLOAM	0.65	0.232
HectorSLAM+ OrbSLAM2	0.65	0.229
HectorSLAM + FLOAM	6.12	1.627

Tabella 5. Performance sulla sequenza Rural-seq2, frequenza di acquisizione 5 Hz, lunghezza sequenza 275 s

Configurazione	**Mean APE (m)**	**APE Std (m)**
HectorSLAM	202.71	175.166
ORBSLAM2	0.39	0.338
FLOAM	4.09	3.788
Odometry	4.14	2.424
HectorSLAM+ OrbSLAM2 + FLOAM + Odometry	4.01	3.810
HectorSLAM+ OrbSLAM2 + FLOAM	0.69	0.343
HectorSLAM+ OrbSLAM2	0.69	0.338
HectorSLAM + FLOAM	1387	2194.787

Tabella 6. Performance sulla sequenza Rural-seq3, frequenza di acquisizione 5 Hz, lunghezza sequenza 669 s

Configurazione	Mean APE (m)	APE Std (m)
HectorSLAM	113.29	56.156
ORBSLAM2	2.58	1.635
FLOAM	17.76	9.87
Odometry	14.03	8.339
HectorSLAM+ OrbSLAM2 + FLOAM + Odometry	17.07	10.171
HectorSLAM+ OrbSLAM2 + FLOAM	2.59	1.281
HectorSLAM+ OrbSLAM2	2.81	1.364
HectorSLAM + FLOAM	114.67	55.159

Considerazioni conclusive

Da questa prima fase sperimentale sono emersi diversi aspetti: le basse performance del metodo HectorSLAM evidenzia il fatto che il metodo risulta non applicabile in contesti reali, infatti il fatto che tale metodo SLAM estragga una mappa 2D dell'ambiente rende l'algoritmo inadatto alla modellazione di ambienti poco strutturati (come quelli extraurbani). La localizzazione tramite sensore odometrico risulta invece eccessivamente imprecisa a causa della deriva della stima dovuta alla rumorosità e agli errori di misurazione del sensore. Anche il metodo FLOAM ha ottenuto basse performance, benchè l'algoritmo sia in grado di modellare ambienti 3D. Le basse performance di FLOAM sono causate principalmente dalla bassa capacità del metodo di modellare ambienti poco strutturati (La localizzazione di FLOAM infatti sfrutta la presenza di elementi geometrici "regolari", come superfici piane e angoli, all'interno dell'ambiente). Il metodo che ha ottenuto i risultati migliori in questo set di esperimenti risulta essere ORBSLAM2, grazie anche alla sua capacità di estrarre una mappa 3D dell'ambiente che risulti efficace per la localizzazione del robot anche in ambienti strutturati.

3.3.5 Sperimentazione preliminare in ambiente reale controllato (esperimenti in laboratorio con Turtlebot)

La seconda parte della sperimentazione è stata effettuata in un ambiente reale all'interno degli spazi del Dipartimento di Ingegneria dell'Università degli studi di Perugia. In particolare è stata impiegata una piattaforma robotica per la didattica, il turtlebot-2, su cui è stata installato la sensoristica e l'hardware necessario a riprodurre una versione "in scala" della dotazione hardware che sarà installata sul veicolo elettrico autonomo reale e che verrà impiegato nelle fasi più avanzate di sperimentazione. In particolare il turtlebot-2 dispone di una struttura di "base" dotata di due motori elettrici in modalità "differential drive", da un laser-scanner 2D e da una telecamera dotata di sensore di profondità con reticolo ad infrarossi (analogo ad un Microsoft Kinect).

Per effettuare gli esperimenti utili per il progetto la piattaforma robotica è stata strumentata con il seguente hardware aggiuntivo:

- sensore LIDAR Velodyne VLP 16 Puck Lite: sensore LIDAR in grado di effettuare una misura a 360° dell' ambiente circostante;
- Swift Duro Inertial GPS: sensore GPS-RTK dotato di sensore inerziale (Inertial Measurement Unit - IMU) e di un enclosure progettato per applicazioni outdoor e resistente a urti, acqua e polveri;

- Nvidia Jetson TK-1 Single Board Computer (SBC): scheda dotata di GPU in grado di processare i dati provenienti dai sensori e di interfacciarsi con l'hardware di controllo di "basso livello" (e che gestisce i motori del robot);
- Switch D-Link DGS-1100-05 a 5 porte: essendo tutti i sensori e tutto l'hardware presente sulla piattaforma interfacciabile tramite porta ethernet, si è reso necessario l'installazione di uno switch per rendere possibile l'implementazione di una rete di sensori TCP/IP.

In figura 11 è mostrata la piattaforma robotica turtlebot-2 utilizzata per gli esperimenti, completo dell'hardware e la sensoristica aggiuntiva installata.

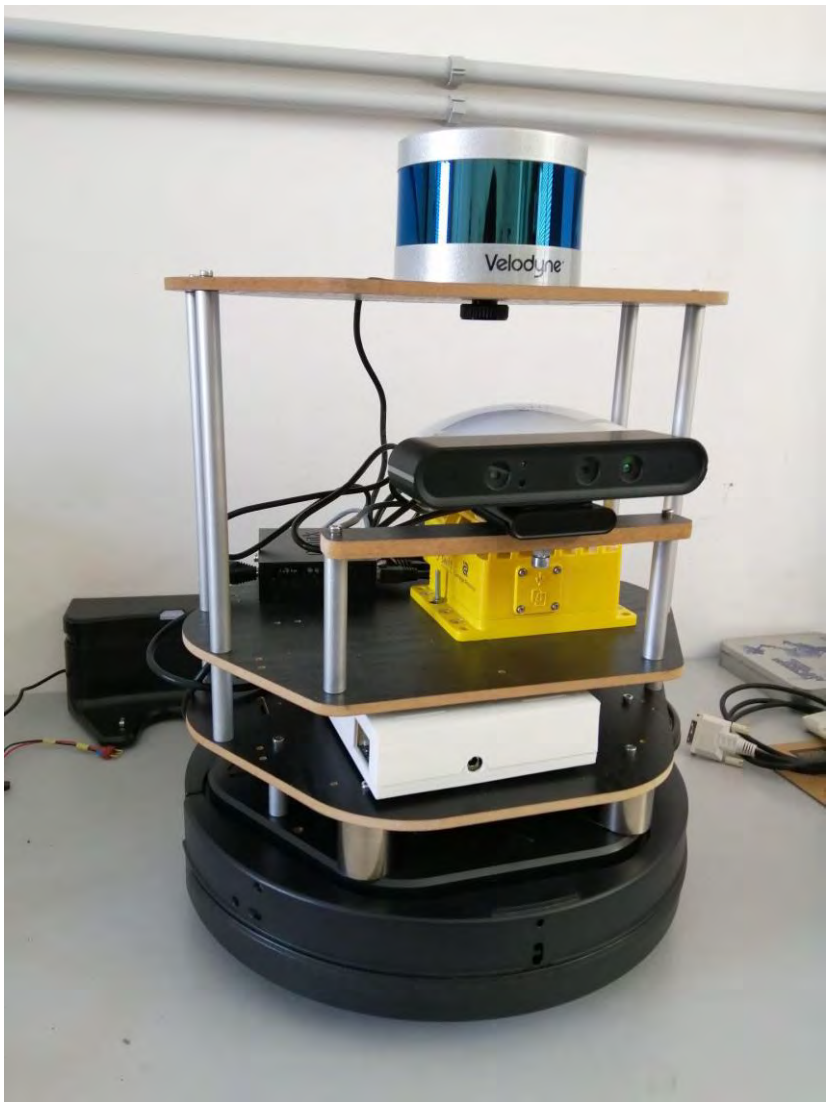

Figura 11. La piattaforma Turtlebot-2 utilizzata in questa fase di sperimentazione in ambiente reale, a cui sono stati aggiunti i sensori LIDAR e GPS-RTK

Sperimentazione preliminare in ambiente outdoor reale

Gli esperimenti sono stati condotti principalmente nel cortile interno del Dipartimento di Ingegneria dell'Università degli Studi di Perugia (di cui è mostrata una vista da satellite in Figura 12). Negli studi sperimentali sono stati validati i seguenti algoritmi:

- LIO-SAM (Shan et al., 2020): sistema di localizzazione e mapping (SLAM) che permette di processare i dati provenienti dal sensore LIDAR e di correlarli con le misure provenienti dalla IMU, creando una mappa 3D dell'ambiente e localizzando il robot all'interno della mappa metrica generata.

- Extended Kalman Filter (EKF) sensor fusion: sistema di fusione sensoriale basato su filtro di kalman esteso (EKF) che permette di fondere la posa del robot fornita dall'algoritmo LIO-SAM con quella fornita dal sensore GPS-RTK fornito di IMU.
- Time Elasting Band (TEB) Path Planner (Rösmann et al. 2017): questo algoritmo svolge tre funzioni fondamentali per la generazione ed il controllo della traiettoria:
    - Generare una mappa degli ostacoli presenti nell'ambiente in cui si muove la piattaforma robotica (coastmap) in base alle misure raccolte dai sensori di prossimità (nel nostro caso il LIDAR).
    - Generare una traiettoria globale (global planner) tenendo conto del punto di partenza del robot (start point), del punto di arrivo indicato dall' utente (goal) e degli ostacoli contenuti nella coastmap.
    - Generare una traiettoria locale (local planner) in base alla traiettoria generata dal global planner tenendo conto della cinematica specifica del robot utilizzato. Il local planner utilizzato in particolare è in grado di generare traiettorie che tengano conto dei vincoli cinematici del robot utilizzati, è possibile ad esempio impostare un raggio di sterzata minimo (che sarà diverso da zero per robot che non sono di tipo differential drive).

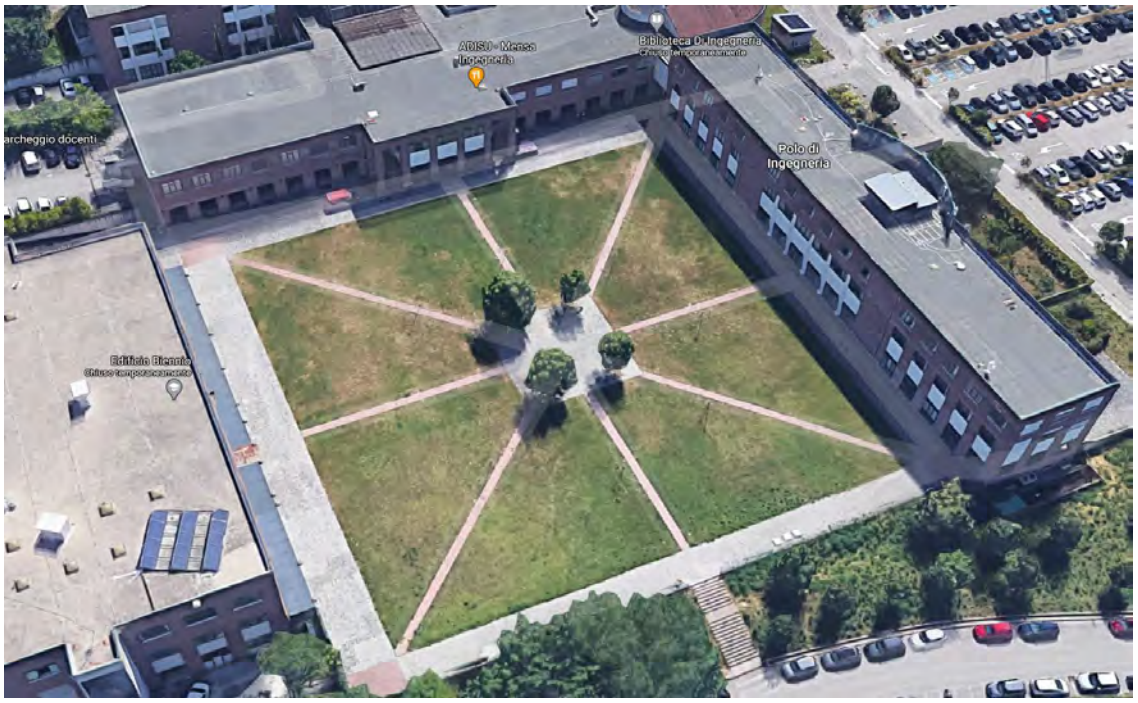

Figura 12. Cortile interno del Dipartimento di Ingegneria dell'Università degli studi di Perugia

Il path planner fornisce come segnale di output la sequenza dei punti che devono essere raggiunti dal robot e le velocità lineari ed angolari del robot che dovranno essere poi fornite al sistema di controllo "di basso livello" che tradurrà queste velocità in potenza elettrica da fornire ai motori.

Risultati qualitativi dei dati ricostruiti sono mostrati nelle figure 13 e 14. Nell'immagine in figura 13 è mostrata la mappa e la posa stimati dall'algoritmo LIO-SAM.

Nell'immagine in figura 14, è mostrata la mappa metrica generata da LIO-SAM (composta dalla "nuvola" di punti in arancione), la posizione del robot stimata dall'algoritmo di fusione sensoriale (raffigurata con i tre assi del sistema di riferimento solidale al robot, colorati in rosso, verde e blu), gli ostacoli rilevati nelle vicinanze del robot (raffigurati tramite "zone" della mappa colorate in azzurro/viola), la traiettoria generata

dal global planner (raffigurata come una linea verde) e quella generata dal local planner (raffigurata con la linea viola).

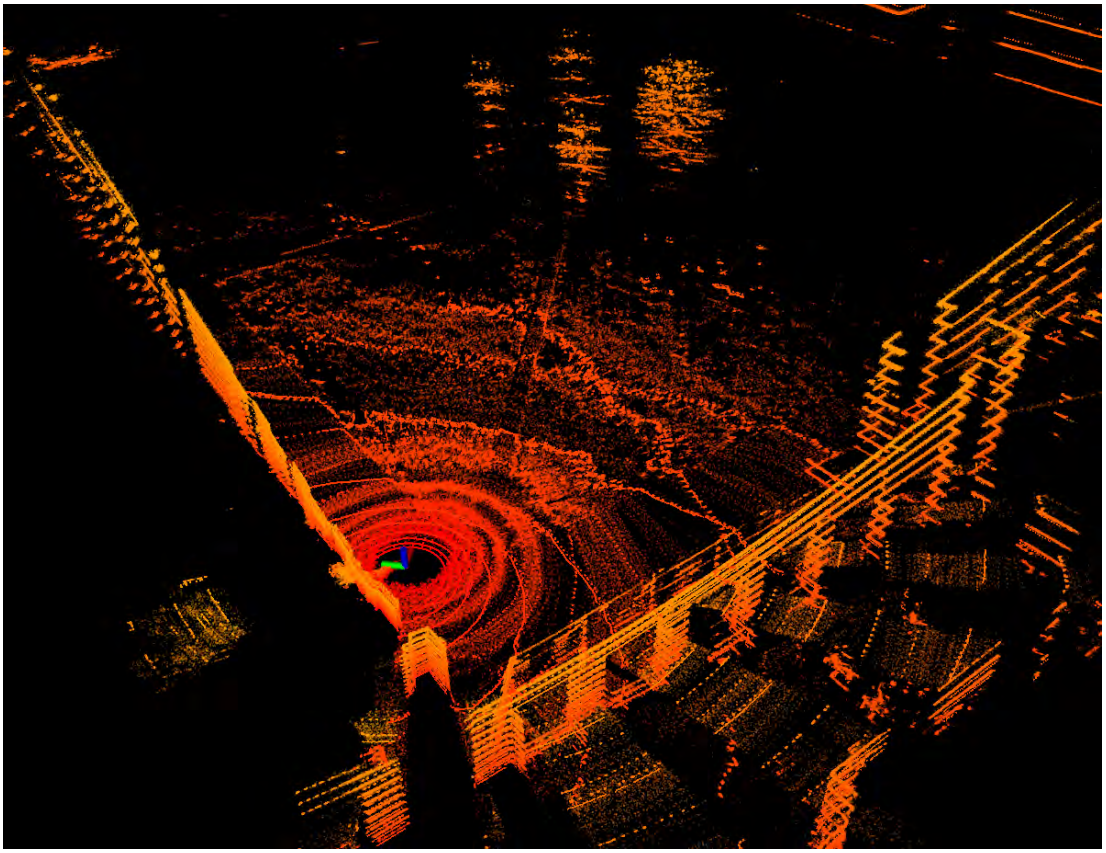

Figura 13. Mappa e posa del robot stimata con il LIO-SAM

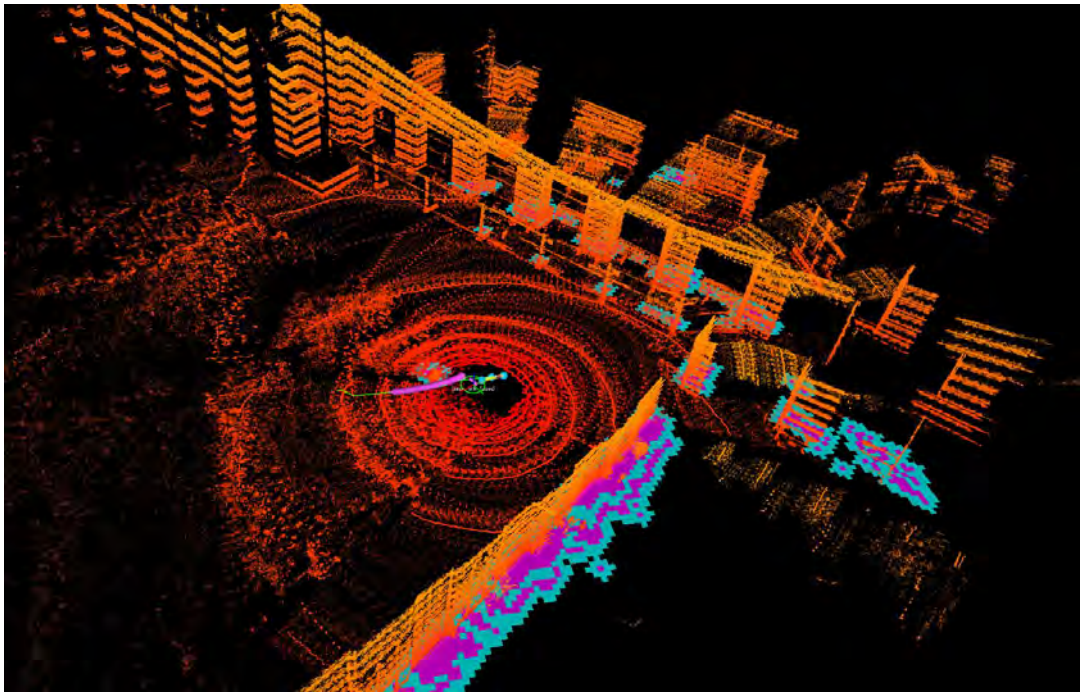

Figura 14. Mappa ottenuta con LIO-SAM, posa ottenuta tramite EKF e coastmap e traiettorie ottenute tramite TEB path planner

4 Conclusioni

In questo report è stato presentato il primo stato di avanzamento delle attività curate dal gruppo ISARLab relative al progetto Smart Road, attività che hanno lo scopo di sviluppare un sistema di localizzazione e navigazione autonoma basato su un veicolo elettrico in ambiente urbano. In questa prima fase del progetto, la sperimentazione svolta in ambiente simulato ha avuto un ruolo determinante. La possibilità di operare con un ambiente di simulazione 3D realistico, infatti, ha permesso lo sviluppo ed il test delle componenti software di localizzazione, mapping e navigazione senza i rischi, lunghi tempi di sviluppo e costi legati a test sperimentali in ambiente reale. In questo modo, nella seconda fase del progetto, sarà possibile integrare gli algoritmi nel veicolo elettrico più velocemente e riducendo al minimo la necessità di sviluppare codice aggiuntivo.

I dati sensoriali prodotti dai vari sensori e sistemi di localizzazione testati sono stati combinati con tecniche di sensor fusion, permettendo una localizzazione ancora più precisa. Le performance di alcuni algoritmi di SLAM allo stato dell'arte, nonché diverse configurazioni di algoritmi di fusione sensoriali, sono stati confrontate utilizzando sei sequenze generate navigando all'interno dell'ambiente di simulazione appositamente predisposto per simulare l'acquisizione di dati sensoriali dalla strumentazione di bordo. A tal fine sono stati sviluppati due diversi ambienti simulati, uno che ricostruisce uno scenario urbano, mentre l'altro ricostruisce un territorio rurale. I test sui dati simulati hanno permesso di mettere in evidenza i punti di forza e di debolezza dei vari algoritmi nei due diversi scenari, caratterizzati da diverse condizioni di terreno ed illuminazione. Tali studi hanno messo in evidenza la necessità dell'impiego contemporaneo di più sistemi di SLAM in parallelo e dell'utilità delle fusione sensoriale delle stime.

Sono stati inoltre effettuati dei test preliminari in ambiente reale predisponendo una piattaforma robotica (Turtlebot-2) con i sensori che verranno poi utilizzati sul veicolo elettrico nelle successive fasi della sperimentazione. I test hanno consentito di effettuare una valutazione qualitativa preliminare del funzionamento degli algoritmi di localizzazione e mapping e degli algoritmi di pianificazione e controllo delle traiettorie.

5 Riferimenti bibliografici

1. Ali, W., Abdelkarim, S., Zahran, M., Zidan, M., and Sallab, A. E. (2018). YOLO3D: end-to-end real-time 3d oriented object bounding box detection from lidar point cloud. CoRR, abs/1808.02350.

2. Beltrán, J., Guindel, C., Moreno, F. M., Cruzado, D., García, F., and de la Escalera, A. (2018). Birdnet: a 3d object detection framework from lidar information. CoRR, abs/1805.01195.

3. Biasutti, P., Lepetit, V., Aujol, J., Brédif, M., and Bugeau, A. (2019). Lu-net: An efficient network for 3d lidar point cloud semantic segmentation based on end-to-end-learned 3d features and u-net. CoRR, abs/1908.11656.

4. Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. (2016). Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. IEEE Transactions on Robotics, 32(6):1309–1332.

5. Campos, C., Elvira, R., Gomez, J. J., Montiel, J. M. M., and Tardos, J. D. (2020). ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. arXiv preprint arXiv:2007.11898.

6. Carlone, L., Aragues, R., Castellanos, J., and Bona, B. (2011). A linear approximation for graph- based simultaneous localization and mapping.

7. Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R. (2020). Learning to explore using active neural slam. In International Conference on Learning Representations (ICLR).

8. Civera, J., Davison, A., and Montiel, J. (2008). Inverse depth parametrization for monocular slam. Robotics, IEEE Transactions on, 24:932 – 945.

9. Coleman, D., Sucan, I. A., Chitta, S., and Correll, N. (2014). Reducing the barrier to entry of complex robotic software: a moveit! case study. CoRR, abs/1404.3785.

10. Costante, G., Mancini, M., Valigi, P., and Ciarfuglia, T. (2015). Exploring representation learning with cnns for frame-to-frame ego-motion estimation. IEEE Robotics and Automation Letters, 1:1–1.

11. Cvišić, I., Ćesić, J., Marković, I., and Petrovic, I. (2017). Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles. Journal of Field Robotics, 35.

12. Deschaud, J. (2018). Imls-slam: Scan-to-model matching based on 3d data. In 2018 IEEE Inter-national Conference on Robotics and Automation (ICRA), pages 2480–2485.

13. Dewan, A. and Burgard, W. (2019). Deeptemporalseg: Temporally consistent semantic segmentation of 3d lidar scans. CoRR, abs/1906.06962.

14. Dube, R., Cramariuc, A., Dugas, D., Nieto, J., Siegwart, R., and Cadena, C. (2018). Segmap: 3d segment mapping using data-driven descriptors.

15. Engel, J., Koltun, V., and Cremers, D. (2016). Direct sparse odometry. CoRR, abs/1607.02565. Engel, J., Schoeps, T., and Cremers, D. (2014). Lsd-slam: large-scale direct monocular slam. volume 8690, pages 1–16.

16. Engel, J., Stückler, J., and Cremers, D. (2015). Large-scale direct slam with stereo cameras, pages 1935–1942.

17. Forster, C., Zhang, Z., Gassner, M., Werlberger, M., and Scaramuzza, D. (2016). Svo: Semidirect visual odometry for monocular and multicamera systems. IEEE Transactions on Robotics, PP.

18. Greene, W., Ok, K., Lommel, P., and Roy, N. (2016). Multi-level mapping: Real-time dense monocular slam. pages 833–840.

19. Grisetti, G., Kümmerle, R., Stachniss, C., and Burgard, W. (2010). A tutorial on graph-based slam. IEEE Intelligent Transportation Systems Magazine, 2(4):31–43.

20. Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pages 2432–2437.

21. Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. IEEE Transactions on Robotics, 23(1):34–46.

22. Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1271–1278.

23. Huang, B., Zhao, J., and Liu, J. (2020). A survey of simultaneous localization and mapping with an envision in 6g wireless networks.

24. Kohlbrecher, S., Meyer, J., von Stryk, O., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR). IEEE.

25. Konda, K. and Memisevic, R. (2015). Learning visual odometry with a convolutional network, volume 1.

26. Konolige, K., Grisetti, G., Kümmerle, R., Burgard, W., Limketkai, B., and Vincent, R. (2010). Efficient sparse pose adjustment for 2d mapping. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 22–29.

27. Lu, W., Zhou, Y., Wan, G., Hou, S., and Song, S. (2019). L 3 -net: Towards learning based lidar localization for autonomous driving.

28. Macenski, S., Martin, F., White, R., and Ginés Clavero, J. (2020). The marathon 2: A navigation system. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).
29. Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K. (2010). The office marathon: Robust navigation in an indoor office environment. In International Conference on Robotics and Automation.

30. Minemura, K., Liau, H., Monrroy, A., and Kato, S. (2018). Lmnet: Real-time multiclass object detection on CPU using 3d lidar. CoRR, abs/1805.04902.

31. Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In Proceedings of the AAAI National Conference on Artificial Intelligence, Edmonton, Canada. AAAI.

32. Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2003). Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. Proc. IJCAI Int. Joint Conf. Artif. Intell.

33. Moore, T. and Stouch, D. (2014). A generalized extended kalman filter implementation for the robot operating system. In Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13). Springer.

34. Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015). ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Transactions on Robotics, 31(5):1147–1163.

35. Mur-Artal, R. and Tardos, J. D. (2017). ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. IEEE Transactions on Robotics, 33(5):1255–1262.

36. Newcombe, R., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011a). Kinectfusion: Real-time dense surface mapping and tracking. pages 127–136.
37. Newcombe, R., Lovegrove, S., and Davison, A. (2011b). Dtam: Dense tracking and mapping in real-time, pp. 2320–2327.

38. Qi, C. R., Litany, O., He, K., and Guibas, L. J. (2019). Deep hough voting for 3d object detection in point clouds. CoRR, abs/1904.09664.

39. Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2016). Pointnet: Deep learning on point sets for 3d classification and segmentation. CoRR, abs/1612.00593.

40. Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. CoRR, abs/1706.02413.

41. Salas-Moreno, R., Newcombe, R., Strasdat, H., Kelly, P., and Davison, A. (2013). Slam++: Simul- taneous localisation and mapping at the level of objects, pp 1352–1359.

42. Schlegel, D., Colosi, M., and Grisetti, G. (2017). Proslam: Graph SLAM from a programmer's perspective. CoRR, abs/1709.04377.

43. Shan, T. and Englot, B. (2018). Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4758–4765. IEEE.

44. Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., and Daniela, R. (2020). Lio-sam: tightly-coupled lidar inertial odometry via smoothing and mapping. In IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5135–5142. IEEE.

45. Shi, S., Wang, X., and Li, H. (2018). Pointrcnn: 3d object proposal generation and detection from point cloud. CoRR, abs/1812.04244.

46. Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. IEEE Robotics & Automation Magazine, 19(4):72–82.

47. Sumikura, S., Shibuya, M., and Sakurada, K. (2019). Openvslam: A versatile visual SLAM framework. CoRR, abs/1910.01122.

48. Tang, J., Ericson, L., Folkesson, J., and Jensfelt, P. (2019). Gcnv2: Efficient correspondence prediction for real-time SLAM. CoRR, abs/1902.11046.

49. Tateno, K., Tombari, F., Laina, I., and Navab, N. (2017). CNN-SLAM: real-time dense monocular SLAM with learned depth prediction. CoRR, abs/1704.03489.

50. Uy, M. A. and Lee, G. H. (2018). Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. CoRR, abs/1804.03492.

51. Yang, B., Luo, W., and Urtasun, R. (2019). PIXOR: real-time 3d object detection from point clouds. CoRR, abs/1902.06326.

52. Yang, S., Song, Y., Kaess, M., and Scherer, S. (2016). Pop-up slam: Semantic monocular plane slam for low-texture environments, pp. 1222–1229.

53. Yi, K., Trulls, E., Lepetit, V., and Fua, P. (2016). Lift: Learned invariant feature transform.

54. Zhang, G., Liu, H., Dong, Z., Jia, J., Wong, T., and Bao, H. (2015). ENFT: efficient non-consecutive feature tracking for robust structure-from-motion. CoRR, abs/1510.08012.

55. Zhang, J. and Singh, S. (2014). Loam: Lidar odometry and mapping in real-time. In Robotics: Science and Systems, volume 2.

56. Zhou, Y. and Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4490–4499.

57. C. Rösmann, F. Hoffmann and T. Bertram: Integrated online trajectory planning and optimization in distinctive topologies, Robotics and Autonomous Systems, Vol. 88, 2017, pp. 142–153.

58. C. Rösmann, F. Hoffmann and T. Bertram: Kinodynamic Trajectory Optimization and Control for Car-Like Robots, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, Sept. 2017.

59. Thrun, Sebastian. "Learning occupancy grid maps with forward sensor models." *Autonomous robots* 15.2 (2003): 111-127.

60. Birk, Andreas, and Stefano Carpin. "Merging occupancy grid maps from multiple robots." *Proceedings of the IEEE* 94.7 (2006): 1384-1397.

61. Hornung, Armin, et al. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." Autonomous robots 34.3 (2013): 189-206.

62. N. Rivera, J. A. Baier, and C. Hernandez, “Weighted real-time heuristic search,” in Proc. of the 2013 Int. Conf. on Autonomous agents and multi- agent systems (AAMAS ’13).

63. F. Ducho, A. Babinec, M. Kajan, P. Beo, M. Florek, T. Fico, and L. Juriica, “Path planning with modified a star algorithm for a mobile robot,” Procedia Engineering, vol. 96, 12 2014.

64. J. Santos, P. Costa, L. Rocha, K. Vivaldini, A. Moreira, and G. Veiga, “Validation of a time based routing algorithm using a realistic automatic warehouse scenario,” vol. 418, 11 2016.

65. W. Xu and J. M. Dolan, “A real-time motion planner with trajectory optimization for autonomous vehicles,” in Proceedings of the Intl. Conf. on Robotics and Automation, 2012, pp. 2061–2067.
66. D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23–33, March 1997.

67. B. Lau, C. Sprunk, and W. Burgard, “Kinodynamic motion planning for mobile robots using splines,” in IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, USA, 2009, pp. 2427–2433.

68. Rösmann, Christoph, Frank Hoffmann, and Torsten Bertram. "Kinodynamic trajectory optimization and control for car-like robots." *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.

69. L. Changan, Y. Xiaohu, L. Chunyang, and L. Guodong, “Dynamic path planning for mobile robot based on improved genetic algorithm,” Chinese Journal of Electronics, vol. 19, 05 2010.

70. Xie, Ronglei, et al. "Unmanned Aerial Vehicle Path Planning Algorithm Based on Deep Reinforcement Learning in Large-Scale and Dynamic Environments." IEEE Access 9 (2021): 24884-24900.

71. Aldegheri, Stefano, et al. "Data flow orb-slam for real-time performance on embedded gpu boards." *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

72. I riferimenti bibliografici devono essere richiamati nel testo con numeri progressivi tra parentesi quadre e riportati a fine testo con il seguente formato:

6 Abbreviazioni ed acronimi

BRIEF: Features from Accelerated Segment Test

EKF: Extended Kalman Filter

FAST: Features from Accelerated Segment Test

FLOAM: Fast LOAM

GPS: Global Positioning System

IMU: Inertial Measurement Unit

LEGO-LOAM: Lightweight and Ground-Optimized LOAM

LIDAR: Laser Imaging Detection and Ranging

LIO-SAM: Lidar Inertial Odometry via Smoothing And Mapping

LOAM: LIDAR Odometry and Mapping

ORB: Oriented FAST and Rotated BRIEF

SBC: Single Board Computer

SLAM: Simultaneous Localization and Mapping

RTK: Real-Time Kinematic

TEB: Time Elastic Band

7 Breve Curriculum Vitae del Gruppo di Lavoro del Dipartimento di Ingegneria, Università degli Studi di Perugia

Enrico Bellocchio ha conseguito la Laurea Magistrale in Ingegneria dell'Informazione e dell'Automazione presso l'Università degli Studi di Perugia nel 2014. Dal 2014 fa parte dell'Intelligent Systems, Automation and Robotics Laboratory (ISARLab) come assegnista di ricerca. La sua attuale attività di ricerca è focalizzata su machine learning, computer vision e robotica.

Francesco Crocetti ha conseguito la Laurea Magistrale in Ingegneria dell'Informazione e dell'Automazione nel 2018 presso l'Università degli Studi di Perugia. Dal 2019 è uno studente di dottorato in Ingegneria Industriale e dell'Informazione presso il Dipartimento di Ingegneria dell'Università degli Studi di Perugia. I suoi interessi di ricerca sono principalmente l'automazione, il controllo, la robotica e il machine learning.

Gabriele Costante ha conseguito il Dottorato di Ricerca in Ingegneria dell'Informazione presso il Dipartimento di Ingegneria dell'Università degli Studi Perugia nel 2016. Attualmente è Ricercatore presso l'Intelligent Systems, Automation and Robotics Laboratory (ISARLab) e docente dei corsi di Computer Vision and Robot Perception e Machine Learning and data Analysis presso il Dipartimento di Ingegneria dell'Università degli Studi di Perugia. I suoi interessi di ricerca includono l'intelligenza artificiale, la robotica, la visione artificiale e il Machine Learning.

Mario Luca Fravolini ha conseguito il dottorato di ricerca in Ingegneria Elettronica presso l'Università degli Studi di Perugia nel 2000. Ha lavorato come assistente di ricerca nel gruppo di teoria dei controlli presso la School of Aerospace Engineering, Georgia Institute of Technology, e presso il Dipartimento di Ingegneria Meccanica e Aerospaziale della West Virginia University. Attualmente è professore associato presso il

Dipartimento di Ingegneria dell'Università degli Studi di Perugia. I suoi interessi di ricerca includono: Diagnosi dei guasti, Controllo intelligente e Controllo Intelligente, e Adattativo e Imaging Biomedico.

Paolo Valigi ha conseguito il dottorato di ricerca presso l'Università di Roma "Tor Vergata" nel 1991. Dal 1990 al 1994 ha collaborato con la Fondazione Ugo Bordoni. Dal 2004 è Professore Ordinario presso l'Università degli Studi di Perugia, Dipartimento di Ingegneria. Attualmente è responsabile dell'Intelligent Systems, Automation and Robotics Laboratory (ISARLab). I suoi interessi di ricerca includono la teoria e applicazioni del controllo automatico, l'identificazione dei sistemi, la robotica e la System Biology.

7.1 Lista delle pubblicazioni recenti più rilevanti

- Costante, G., Mancini, M. "Uncertainty Estimation for Data-driven Visual Odometry". IEEE Transactions on Robotics, 2020.
- Devo, A., Mezzetti, G., Costante, G., Fravolini, M. L., & Valigi, P. "Towards Generalization in Target-Driven Visual Navigation by Using Deep Reinforcement Learning". IEEE Transactions on Robotics, 2020.
- Fravolini, M. L., Costante, G., Yucelen, T., & Napolitano, M. R.. "Interval Prediction Models for Data-Driven Design of Aerial Vehicle's Robust Adaptive Controllers". Journal of Guidance, Control, and Dynamics, 1-17, 2020.
- Crocetti F., Costante G., Fravolini M.L., Valigi P. "A Data-Driven Slip Estimation Approach for Effective Braking Control under Varying Road Conditions " 2020 28th Mediterranean Conference on Control and Automation (MED), 496-501
- Cartocci N., Costante G., Napolitano M.R., Valigi P., Crocetti F., Fravolini M.L. "PCA Methods and Evidence Based Filtering for Robust Aircraft Sensor Fault Diagnosis" 2020 28th Mediterranean Conference on Control and Automation (MED), 550-555
- Cascianelli S., Costante G., Crocetti F., Ricci E., Valigi P., Fravolini M.L. "Data‐based design of robust fault detection and isolation residuals via LASSO optimization and Bayesian filtering" Asian Journal of Control, 2020
- Devo, Alessandro, Costante G., and Paolo Valigi. "Deep Reinforcement Learning for Instruction Following Visual Navigation in 3D Maze-Like Environments." IEEE Robotics and Automation Letters 5(2): 1175-1182, 2020.
- Costante G., Ciarfuglia T. A. "Ls-vo: Learning dense optical subspace for robust visual odometry estimation", IEEE Robotics and Automation Letters, 2018.
- Mancini M., Costante G., Valigi P., Ciarfuglia T. A. "J-MOD2: Joint Monocular Obstacle Detection and Depth Estimation", IEEE Robotics and Automation Letters, 2018.
- Abdollahyan M., Cascianelli S., Bellocchio E., Costante G., Ciarfuglia T.A., Bianconi F., Smeraldi F., Fravolini M.L. "Visual Localization in the Presence of Appearance Changes Using the Partial Order Kernel", 26th European Signal Processing Conference (EUSIPCO), 2018.
- Cascianelli, S.; Costante, G.; Bellocchio, E.; Valigi, P.; Fravolini, M. L.; Ciarfuglia, T. A. "Robust visual semi-semantic loop closure detection by a covisibility graph and CNN features.", Robotics and Autonomous Systems, 92, 53-65, 2017.
- Costante, G.; Mancini, M.; Valigi, P; Ciarfuglia, T.A. "Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation", IEEE Robotics and Automation Letters, 2016, Best Vision Paper Finalist alla conferenza IEEE International Conference on Robotics and Automation.
- Mancini, M.; Costante, G.; Valigi, P.; Ciarfuglia, T.A. "Fast robust monocular depth estimation for Obstacle Detection with fully convolutional networks", IEEE/RSJ International Conference on Intelligent Robotics and Systems, 2016.