



UNIVERSITÀ DEGLI STUDI DI FIRENZE
SCUOLA DI INGEGNERIA - DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

Tesi di Laurea Triennale in Ingegneria Informatica

**RICERCA E SVILUPPO DI UN SISTEMA DI
CALIBRAZIONE AUTOMATICA CAMERA-LIDAR
PER APPLICAZIONI DI GUIDA AUTONOMA.**

Candidato
Gianni Moretti

Relatore
Prof. Carlo Colombo

Correlatore
Marco Fanfani, PhD

Anno Accademico 2023/2024

Indice

| | |
|--|----|
| Introduzione | i |
| 1 Componenti Hardware e Integrazione con ROS | 1 |
| 1.1 Panoramica dei Sensori: Camera e LiDAR | 1 |
| 1.1.1 Camera Stereo ZED2i | 2 |
| 1.1.2 LiDAR Velodyne Puck Hi-Res | 3 |
| 1.2 ROS | 4 |
| 1.2.1 ROS Filesystem Level | 5 |
| 1.2.2 ROS Computational Graph Level | 5 |
| 2 Il Nodo velodyne_light | 9 |
| 2.1 Funzionamento dei pacchetti Velodyne | 9 |
| 2.2 Funzionamento dei nodi Velodyne | 10 |
| 2.3 Il nodo velodyne_light | 13 |
| 3 Calibrazione Estrinseca Automatica tra Camera Stereo e Li-DAR | 16 |
| 3.1 Introduzione alla calibrazione estrinseca | 17 |
| 3.2 Perchè effettuare una calibrazione estrinseca automatica | 17 |
| 3.3 L'articolo di riferimento | 18 |
| 3.3.1 Metodo di calibrazione utilizzato | 18 |

| | | |
|----------|---|-----------|
| 3.3.2 | Processo di calibrazione tra camera stereo e LiDAR | 20 |
| 3.4 | Criticità riscontrate nell'articolo | 25 |
| 3.5 | La soluzione proposta | 26 |
| 3.6 | Risultati del metodo di calibrazione | 30 |
| 3.7 | Miglioramento dell'Intersection over Union con l'aumentare delle posizioni del pattern | 32 |
| 3.8 | Confronto con misurazioni manuali | 33 |
| 3.9 | Proiezione dei punti LiDAR nell'immagine della camera sinistra | 35 |
| 3.10 | Comparazione tra le due soluzioni | 36 |
| 4 | Idee e Sviluppi futuri: Integrazione dei punti LiDAR su ORB_SLAM3 | 38 |
| 4.1 | Introduzione a ORB_SLAM3 | 39 |
| 4.2 | Idea per integrare i punti del LiDAR nell'algoritmo | 40 |
| 4.3 | Sincronizzazione dei frame | 41 |
| 4.4 | Conclusioni | 42 |
| | Bibliografia | 44 |

Ringraziamenti

Desidero esprimere la mia più sincera gratitudine ai Prof. Carlo Colombo e Marco Fanfani per la loro costante disponibilità e pazienza. Probabilmente sono stato uno dei peggiori studenti da seguire a causa dei miei numerosi impegni che hanno sottratto tempo a questo progetto, prolungandolo oltre il dovuto. Grazie della vostra dedizione.

Voglio ringraziare tutti i miei compagni di corso, mi hanno aiutato a passare questi anni di università in maniera più leggera e spensierata, aiutandomi e facendomi ridere. Grazie soprattutto per avermi incoraggiato, anche nei momenti in cui mi bloccavo e non riuscivo ad andare avanti. Vi ringrazio per i momenti passati fuori dall'università, le partite perse di calcetto (quelle un po' meno) e le serate piene di ricordi. Grazie del vostro affetto.

Un grosso ringraziamento va ai miei amici di sempre, quelli che mi conoscono da una vita, quelli che non dimentichi mai, quelli che, anche se ci allontaniamo per un po', sono pronti a darti una mano. Grazie di esserci sempre.

Un capitolo fondamentale del mio percorso è stato il Firenze Race Team. Potrei stare qui per ore a raccontare i momenti vissuti insieme. Questi tre anni sono stati tra i più importanti della mia vita, e non li dimenticherò mai. Abbiamo affrontato alti e bassi, e l'ultimo anno è stato una vera montagna russa, piena di sfide, ma non ci siamo mai arresi. Mi avete insegnato che nulla

è impossibile e che, non importa quante volte si cada, ci si rialza sempre. Siete la mia seconda famiglia.

Un ringraziamento speciale va al mio collega e amico Emanuele Nencioni, con cui ho condiviso gran parte del mio tempo a Firenze. Grazie per il tuo aiuto nello sviluppo di questa tesi. Spero che continueremo a scrivere codice insieme ancora per molto tempo. Più che un amico, sei stato un fratello.

Nonostante possa sembrare che io sia un fratello distaccato e assente, vi assicuro che penso spesso alle mie sorelle. Devo ringraziarle perché loro sono letteralmente i miei angeli custodi, anche se non li vedi, ti aiutano costantemente. Vivere la vita sapendo che c'è sempre qualcuno che veglia su di me è un dono immenso. Spero di poter diventare lo stesso per voi.

Un ringraziamento speciale va alla mia ragazza, che mi ha sostenuto con una pazienza infinita, come si fa con un bambino, senza mai lasciarmi solo. Mi ha aiutato nei momenti più difficili, quando sembrava che nessuno potesse farlo, e mi ha dato la forza di andare avanti anche quando tutto mi sembrava impossibile. Grazie di esserci. Grazie di noi.

Ho lasciato per ultimi i miei genitori, perché non esistono parole sufficienti per ringraziarli dell'opportunità che mi hanno dato. Mi hanno sempre sostenuto, credendo in me e nelle mie capacità, accompagnandomi in tutte le mie passioni e i miei capricci. Mi hanno permesso di vivere questi anni esattamente come desideravo, e per un dono del genere non esiste ringraziamento adeguato.

Grazie per tutti i vostri sacrifici.

Gianni Moretti

Firenze, 07/10/2024

Introduzione

Il contesto in cui si svolge questa tesi è quello del Firenze Race Team (FRT), il team di Formula Student dell'Università degli Studi di Firenze. La Formula Student è una competizione ingegneristica di livello internazionale che coinvolge studenti universitari di ingegneria meccanica, elettronica e informatica. Lo scopo della competizione è quello di progettare, costruire e testare una vettura da corsa monoposto con o senza pilota (*driverless*). Una delle sfide principali nella progettazione di questi veicoli, è la loro capacità di navigare autonomamente in ambienti sconosciuti. Una tecnologia chiave per affrontare questa sfida è nota come SLAM (*Simultaneous Localization and Mapping*), che consente ai veicoli di creare una mappa dell'ambiente circostante e di determinare la loro posizione all'interno di esso, in tempo reale.

Negli ultimi anni, molti team di Formula Student hanno potenziato i loro sistemi di visione aggiungendo, oltre alla tradizionale camera stereo, sensori LiDAR che migliorano la percezione dell'ambiente.

In linea con questi sviluppi, la presente tesi si propone di integrare il sensore LiDAR nel sistema di visione del FRT [4], basato su ORB_SLAM3 [2] e sviluppato in ROS (*Robot Operating System*). In particolare, il lavoro si è concentrato nello studiare il funzionamento dei LiDAR Velodyne, sulla creazione di un nodo ROS personalizzato a partire da quello già esistente e

sullo sviluppo di un metodo di calibrazione automatica tra la camera stereo e il LiDAR, testandone i risultati e analizzando i relativi vantaggi e svantaggi.

In ultima analisi questa tesi propone un metodo di integrazione dei punti del LiDAR all'interno dell'algoritmo di visione ORB_SLAM3 [2] come sviluppo futuro, con l'obbiettivo di migliorare la precisione e l'affidabilità dell'algoritmo di guida autonoma.

La vettura utilizzata in questo progetto è un'auto da corsa sviluppata dal FRT. Questo veicolo è stato equipaggiato con tutto il necessario per renderlo a guida autonoma. A differenza delle normali auto autonome, il principale scopo di questo veicolo è massimizzare le performance in pista anziché garantire la sicurezza su strada.

La camera è montata frontalmente, nella parte superiore del veicolo, in una posizione rialzata e centrale. Questo posizionamento consente di avere una visione ampia e chiara dell'ambiente circostante, così da poter individuare più coni possibili che caratterizzano la pista.

Il LiDAR è posizionato sull'anteriore della vettura, in particolare sull'ala anteriore. Questa posizione permette di avere una visione più accurata della pista ed identificare meglio i coni posizionati sul tracciato.



Figura 1: Zed2i camera



Figura 2: Velodyne LiDAR

Capitolo 1

Componenti Hardware e Integrazione con ROS

Nel presente capitolo viene fornita una panoramica delle tecnologie e dei sistemi utilizzati nel progetto, con particolare attenzione all'hardware e all'architettura di integrazione. In primo luogo, verranno descritti i sensori impiegati, ossia la Camera e il LiDAR, con un'analisi delle loro caratteristiche tecniche e del ruolo che rivestono nel contesto di un veicolo autonomo. Successivamente, verrà illustrato il sistema ROS (*Robot Operating System*), il framework che permette l'integrazione e la gestione efficiente dei sensori e degli algoritmi a bordo del veicolo del FRT.

1.1 Panoramica dei Sensori: Camera e LiDAR

In questa sezione verranno descritti i sensori principali utilizzati nel progetto: una camera stereo *ZED2i* di *Stereolabs* e un LiDAR *Velodyne Puck Hi-Res* a 16 canali. Questi due dispositivi, facenti parte del sistema di percezione del veicolo autonomo, forniscono informazioni per la comprensione

dell’ambiente circostante, ciascuno con caratteristiche uniche e complementari. Di seguito, verranno analizzate le specifiche tecniche e le funzionalità di entrambi i sensori.

1.1.1 Camera Stereo ZED2i

La *ZED 2i* è una camera stereo prodotta da *Stereolabs*, utilizzata per applicazioni di visione artificiale e robotica avanzata. Dotata di due sensori CMOS, è in grado di acquisire immagini stereoscopiche ad alta risoluzione fino a 4K (3840x2160 pixel). Le caratteristiche principali della *ZED 2i* includono:

- **Risoluzione massima:** fino a 4K (3840x2160 pixel) a 15 FPS. Può anche operare a risoluzioni inferiori, come 2K (2560x1440) a 30 FPS, o 720p (1280x720) a 100 FPS, permettendo una regolazione tra qualità e frame rate.
- **Campo visivo:** 110° orizzontali, 70° verticali, con un range di rilevamento fino a 20 metri in esterni.
- **Frequenza di acquisizione:** regolabile in base alla risoluzione, fino a 100 FPS a risoluzioni inferiori.
- **IMU e Sensori Ambientali Integrati:** la *ZED 2i* è equipaggiata con un’unità di misura inerziale (IMU) a 9 assi, e con sensori di temperatura, pressione e umidità, utili per un tracciamento spaziale accurato.
- **Resistenza agli agenti atmosferici:** grazie alla certificazione IP66, è protetta da acqua e polvere, rendendola ideale per applicazioni in ambienti esterni ostili.

- **Elaborazione della profondità:** utilizza avanzati algoritmi di visione stereoscopica per generare mappe di profondità 3D in tempo reale, con una precisione di 100mm a 10m di distanza.
- **Compatibilità con ROS:** pienamente integrata con ROS, facilita l'uso della camera per la percezione in tempo reale.

Grazie a queste caratteristiche, la *ZED 2i* fornisce dati di profondità dettagliati e ad alta risoluzione.

1.1.2 LiDAR Velodyne Puck Hi-Res

Il *Velodyne Puck Hi-Res* è un LiDAR tridimensionale a 16 canali, progettato per offrire un'elevata risoluzione angolare e una scansione dettagliata dell'ambiente. Di seguito vengono elencate le sue principali caratteristiche tecniche:

- **Numero di canali:** 16 laser, disposti in verticale, consentendo una scansione completa dell'ambiente circostante.
- **Risoluzione angolare:** il *Velodyne Puck Hi-Res* offre una risoluzione orizzontale fino a 0.1°, che permette di acquisire dati estremamente dettagliati rispetto ai modelli standard.
- **Frequenza di scansione:** il sensore può operare a una velocità di rotazione compresa tra 5 Hz e 20 Hz, permettendo di regolare il trade-off tra risoluzione spaziale e frequenza di aggiornamento.
- **Portata massima:** il LiDAR ha una portata operativa fino a 100 metri, con una precisione di misurazione delle distanze inferiore a 3 cm.

- **Campo visivo:** 360° orizzontali e 20° verticali.
- **Tecnologia a impulsi laser:** utilizza la tecnologia Time-of-Flight (ToF) per misurare con precisione la distanza tra il sensore e gli oggetti.
- **Flessibilità e peso ridotto:** il *Velodyne Puck Hi-Res* pesa soltanto 830 grammi, risultando ideale per applicazioni che richiedono sensori leggeri e compatti.
- **Compatibilità con ROS:** il sensore è compatibile con ROS e dispone di driver specifici, facilitando l'integrazione nei sistemi di percezione e localizzazione.
- **Resistenza agli agenti atmosferici:** grazie al design robusto, il *Velodyne Puck Hi-Res* è in grado di operare in condizioni ambientali difficili, inclusa la pioggia e la polvere, con un'ampia gamma di temperature operative.

1.2 ROS

ROS sta per *Robot Operating System* anche se non si tratta di un vero e proprio sistema operativo, ma più di un *framework* di basso livello che viene eseguito su Linux. Una specie di *middleware*, cioè un *layer* che è responsabile di gestire le comunicazioni tra programmi in un sistema distribuito. Questo rende possibile la programmazione in sottoprogrammi, ognuno che si occupa di funzioni specifiche, le quali, messe insieme, riescono a risolvere problemi complessi, come nel caso della guida autonoma. ROS è stato progettato per aiutare a creare rapidamente applicazioni per robot che integrino sensori, attuatori e schede di controllo. Si hanno 3 livelli di concetti: Livello *Filesystem*, Livello *Computational Graph*, Livello *Community*.

1.2.1 ROS Filesystem Level

I concetti che coprono le risorse di ROS su disco sono i seguenti:

- **Packages.** I pacchetti sono l'unità principale per l'organizzazione del software in ROS. Un package può contenere processi ROS, i cosiddetti *nodi* (vedi 1.2.2), una libreria, datasets, file di configurazione e molteplici altri strumenti che necessitano di essere utilizzati insieme.
- **Metapackages.** Pacchetti ROS utilizzati per raggruppare logicamente un insieme di pacchetti correlati tra loro, semplificando la gestione e l'installazione di più pacchetti necessari per una particolare funzionalità o stack di software.
- **Package Manifests.** Manifesti che forniscono metadati su un pacchetto, includendo nome, versione, descrizione, informazioni sulla licenza etc.
- **Message types.** Descrizione di messaggio. Definisce la struttura di un messaggio.
- **Service types.** Descrizione di un servizio. Definisce la struttura dati della richiesta e della risposta.

1.2.2 ROS Computational Graph Level

Rete *peer-to-peer* di processi ROS che stanno elaborando dati insieme.

Concetti basilari di *Computation Graph*

- **Nodes:** processi che eseguono il calcolo. Un nodo ROS è scritto usando una libreria *client* di ROS come roscc o rospy (per c++ e Python rispettivamente).

- *Master*: fornisce la registrazione del nome e la ricerca per il resto del *Computational Graph*. Senza di questo, i nodi non sarebbero in grado di scambiarsi messaggi o richiamare servizi.
- *Messages*: nodi che comunicano tra loro passandosi i messaggi (definiti in 1.2.1).
- *Parameter Server*: permette ai dati di essere immagazzinati in una locazione centrale. Fa parte del master.
- *Topics*: identificatori di *Bus* dove i nodi possono scambiarsi i messaggi. In generale, i nodi non sanno con chi stanno comunicando, invece, un nodo che genera dei dati li **pubblica** (*Publisher*) su un *topic* rilevante. Un nodo interessato a dei dati specifici, si **iscrive** (*Subscriber*) a quel *topic* e riceverà tutti i messaggi. Ad un *topic* possono collegarsi multipli *Subscriber* e *Publisher*. Ogni topic è fortemente tipizzato dal tipo di messaggio ROS usato per la pubblicazione. Il nodo *Master* o i *Publisher* non impongono la consistenza di tipo, ma i *Subscribers* non stabiliranno il trasporto dei messaggi se non c'è corrispondenza di tipo. Ogni *Subscriber* si avvale di una funzione di *callback* per poter ricevere ed elaborare i messaggi che arrivano sul topic in ascolto in *real-time* (fig 1.1).
- *Services*: permettono di creare una comunicazione sincronizzata in stile *client/server* tra i nodi. Molto utile per cambiare un' impostazione o richiedere una specifica azione da un nodo.
- *Bags*: sono formati per il salvataggio e la riproduzione di messaggi ROS. Questi sono un meccanismo importante per immagazzinare i dati,

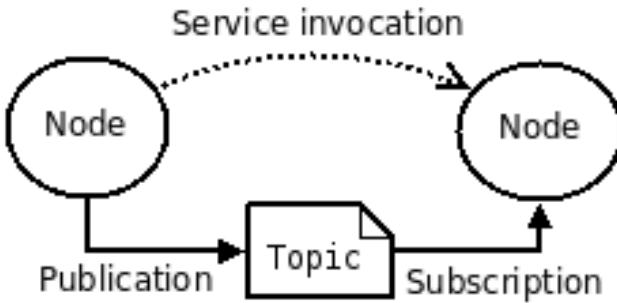


Figura 1.1: Schema del funzionamento di uno scambio di messaggi tra due nodi usando un *topic*

ad esempio quelli provenienti dai sensori che possono essere difficili da collezionare ma sono necessari per sviluppare e testare gli algoritmi.

Il *Master* agisce come un *nameservice* (gestisce la mappatura e i nomi del sistema ai loro indirizzi di rete) nel ROS *Computational Graph*, esso immagazzina le registrazioni delle informazioni su *topics*, *services* per i nodi ROS. Siccome i nodi comunicano con il *Master* per riportare la loro registrazione delle informazioni, potrebbero ricevere le informazioni su altri nodi registrati e quindi stabilire connessioni appropriate.

I nodi, tra loro, stabiliscono direttamente la propria connessione; il *Master* fornisce solo informazioni di *lookup*, come un server DNS, perciò i nodi che si sottoscrivono ad un *topic* richiederanno la connessione ad un nodo che pubblica su quel *topic*. Il protocollo viene stabilito concordatamente tra i nodi, il più comune in ROS è chiamato TCPROS che utilizza *socket* TCP/IP standard.

Questa architettura permette di avere operazioni disaccoppiate dove i nomi sono i mezzi principali per costruire un sistema più grande e complesso. I nomi hanno un ruolo molto importante in ROS: *nodi*, *topics*, *servizi* e *parametri* hanno tutti un nome.

ROS Community Level

ROS Community Level è parte che gestisce le risorse per consentire alla comunità di scambiare software e conoscenze.

Ogni messaggio pubblicato ha associato un *header* che contiene un *timestamp*, un numero di sequenza e anche un campo chiamato *frame_id*.

Frames

I frame in ROS non sono altro che sistemi di riferimento relativi ad un particolare sensore o punto del veicolo. Esistono dei frame predefiniti:

- **map.** Il frame ha la sua origine in un punto fisso arbitrario scelto nel mondo.
- **odom.** Ha la sua origine dove il robot è inizializzato. Il punto è comunque fissato nel mondo.
- **base_footprint.** Posizione in 2D al centro del robot (senza l'altezza) che rappresenta il movimento del robot, perciò questo sistema di riferimento non è fisso ma è solidale al robot.
- **base_link.** Posizione in 3D solidale al robot.

Infine ci possono essere dei *frame_id* personalizzati in base alla tipologia dei sensori che si utilizzano. La trasformazione da un frame ad un altro è statica nel caso in cui i sistemi di coordinate sono fissi tra loro (c'è solo una rototraslazione che non varia nel tempo); invece avviene una trasformazione più complessa tra frame non vincolati, le cui posizioni possono cambiare nel tempo in modo indipendente, ad esempio tra *base_link* che si muove rispetto a *map*. ROS fornisce un *package* chiamato **tf** (*transform listener*) che aiuta ad effettuare le trasformazioni cercando di semplificare questi concetti.

Capitolo 2

Il Nodo `velodyne_light`

Il secondo capitolo della tesi è dedicato al nodo `velodyne_light`, sviluppato per l'integrazione del sensore LiDAR *Velodyne Puck Hi-Res* all'interno dell'ecosistema *ROS* (*Robot Operating System*), partendo dal pacchetto di nodi *Velodyne* già esistente. Il nodo `velodyne_light` è uno dei componenti fondamentali che consente di ricevere, elaborare e visualizzare i dati del LiDAR in tempo reale.

Questo nodo è responsabile dell'acquisizione delle nuvole di punti 3D generate dal LiDAR e della loro conversione in un formato utilizzabile dai vari moduli del sistema. Attraverso una serie di modifiche, il nodo è stato adattato per gestire le specifiche esigenze del progetto e risolvere alcuni errori presenti all'interno dei nodi Velodyne.

2.1 Funzionamento dei pacchetti Velodyne

I LiDAR Velodyne emettono dati sotto forma di pacchetti UDP che contengono le informazioni relative ai punti 3D rilevati nell'ambiente. Ogni

pacchetto UDP ha una struttura specifica e contiene diverse sezioni chiave (fig. 2.1).

Struttura di un pacchetto LiDAR Velodyne

- **Header:** I pacchetti UDP iniziano con un header, che contiene informazioni come il numero del frame, un timestamp, e alcuni metadati.
- **Dati delle scansioni (Data Blocks):** La parte principale del pacchetto contiene i *Data Blocks*, che rappresentano le misure di distanza e intensità ricevute dai diversi laser del LiDAR. Ogni Data Block include:
 - **Azimuth:** L'angolo di rotazione (in gradi) corrispondente alla posizione del fascio laser.
 - **Distanze:** Le distanze misurate dal LiDAR in ogni punto, calcolate dal tempo impiegato dal laser per tornare al sensore.
 - **Intensità:** La forza del segnale di ritorno per ciascuna misura, utile per riconoscere materiali o superfici con riflettività differenti.
- **Informazioni per il posizionamento dei laser:** Ogni Velodyne LiDAR ha più laser posizionati a varie inclinazioni. Il pacchetto include informazioni sulle inclinazioni angolari (*pitch*) per ciascun fascio laser.
- **Footer:** In alcuni casi vengono aggiunte ulteriori informazioni, come i dati GPS per sincronizzare le letture LiDAR con altri sensori.

2.2 Funzionamento dei nodi Velodyne

Il sistema di gestione dei LiDAR Velodyne in ROS utilizza due nodi principali: `velodyne_driver` e `velodyne_pointcloud`. Questi nodi lavorano in

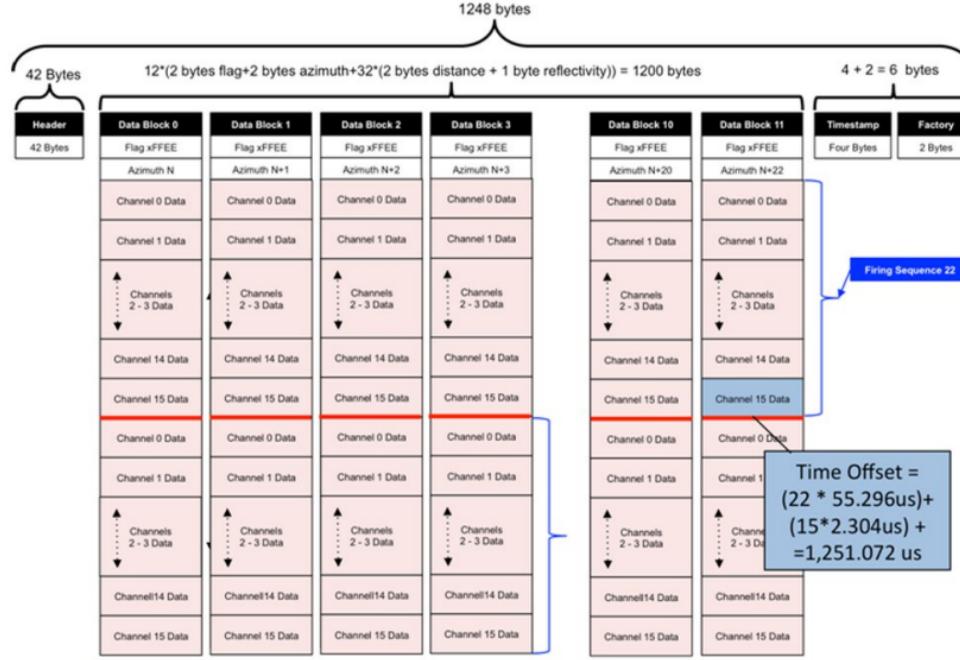


Figura 2.1: Velodyne data block

sinergia per ricevere, elaborare e pubblicare i dati acquisiti dal LiDAR sotto forma di nuvole di punti 3D.

Premessa sulla struttura Il processo di gestione dei dati può essere suddiviso in due parti principali:

- **Nodo *velodyne_driver*:** Riceve i pacchetti UDP inviati dal LiDAR e li decodifica.
- **Nodo *velodyne_pointcloud*:** Raggruppa i dati ricevuti, esegue le conversioni necessarie (da coordinate polari a cartesiane) e li pubblica come nuvole di punti 3D (fig. 2.2).

Il processo complessivo avviene in più fasi:

1. **Ricezione dei pacchetti UDP (*velodyne_driver*):** Il nodo *velodyne_driver* utilizza una socket UDP per ricevere i pacchetti inviati dal LiDAR.

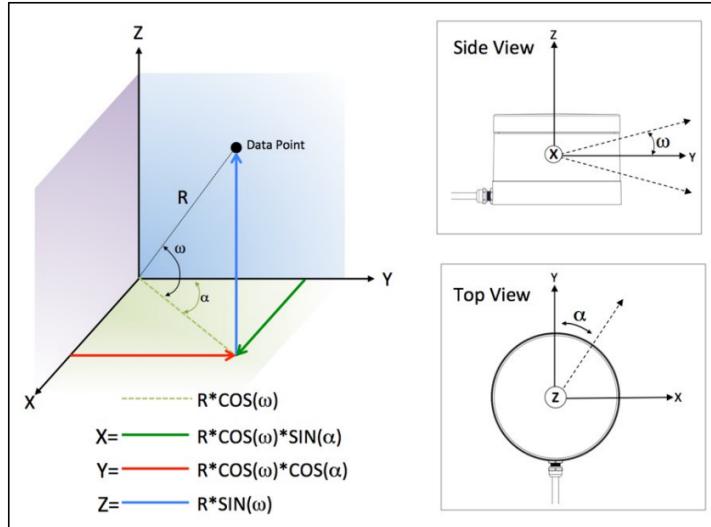


Figura 2.2: Velodyne coordinate system

Ogni pacchetto contiene le misure di distanza e intensità dai laser del sensore.

2. **Decodifica dei pacchetti (velodyne_driver):** Il nodo decodifica i pacchetti e separa le informazioni su azimut, distanza e intensità di ogni punto rilevato.
3. **Conversione e raggruppamento (velodyne_pointcloud):** Una volta che i pacchetti sono stati decodificati, il nodo *velodyne_pointcloud* raccoglie i dati dai vari laser e converte le misure dalle coordinate polari (azimut, distanza) a coordinate cartesiane (X, Y, Z), tenendo conto delle correzioni di calibrazione e aggiustando l'azimuth in base al tempo passato dopo la scansione del laser. Questo nodo si occupa anche di raggruppare i punti rilevati da più pacchetti in una singola nuvola di punti 3D.
4. **Pubblicazione della nuvola di punti (velodyne_pointcloud):** Una volta elaborata, la nuvola di punti viene pubblicata su un topic ROS uti-

lizzando il formato `sensor_msgs/PointCloud2`. Questo messaggio contiene le coordinate spaziali di ciascun punto rilevato e le informazioni sull'intensità del segnale di ritorno.

2.3 Il nodo *velodyne_light*

Il nodo *velodyne_light* è stato sviluppato partendo dai nodi Velodyne preesistenti, modificando e accorpando *velodyne_driver* e *velodyne_pointcloud* in un unico nodo più efficiente e con funzionalità aggiuntive. L'obiettivo di queste modifiche è stato migliorare le prestazioni e ridurre il consumo di risorse di sistema, aspetti particolarmente critici in contesti come i veicoli autonomi, dove le risorse hardware sono spesso limitate.

Premessa sulla struttura A differenza dei nodi Velodyne originari, il nodo *velodyne_light* esegue tutte le operazioni di ricezione, decodifica, conversione e pubblicazione dei pacchetti LiDAR in modo più integrato. La struttura principale del nodo è la seguente:

- **Ricezione e decodifica in tempo reale:** Il nodo riceve i pacchetti UDP, li decodifica immediatamente separando le informazioni su angolo, distanza e intensità.
- **Conversione e correzione:** Dopo la decodifica, i dati vengono convertiti da coordinate polari a coordinate cartesiane, applicando eventuali correzioni necessarie per la calibrazione del sensore.
- **Pubblicazione della nuvola di punti:** Una volta raccolti tutti i punti che coprono un angolo giro (360°), viene pubblicata una nuvola di punti che rappresenta l'ambiente circostante (fig. 2.3).

Funzionalità aggiuntive Il nodo *velodyne_light* introduce diverse ottimizzazioni:

- **Scarto di pacchetti basato su angoli:** Sono stati aggiunti dei parametri che permettono di scartare i pacchetti che rientrano in un range di angoli definito dall'utente. Questo consente di evitare la memorizzazione di dati non necessari, riducendo così l'utilizzo della memoria e aumentando l'efficienza.
- **Riduzione del numero di messaggi ROS:** A differenza dei nodi Velodyne che separano le operazioni su più nodi e inviano più messaggi ROS, il nodo *velodyne_light* accorpa le operazioni, riducendo il numero di messaggi pubblicati e ottimizzando l'uso della memoria RAM.

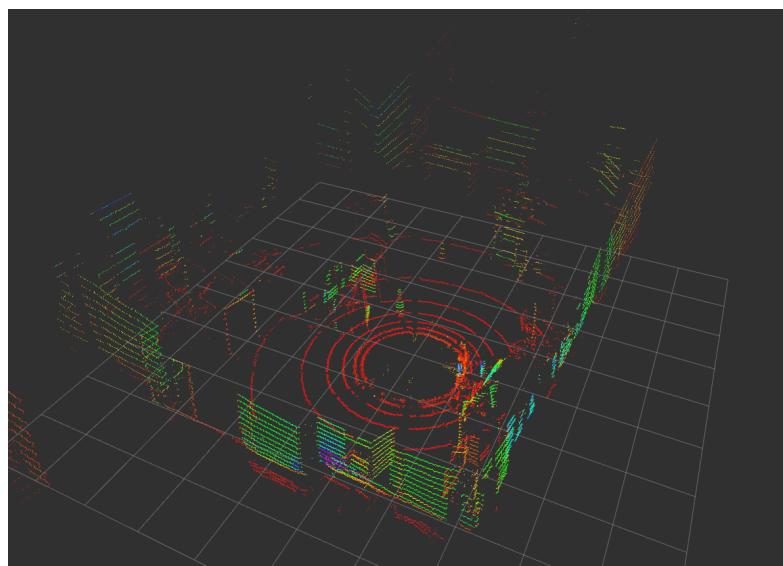


Figura 2.3: Punti del LiDAR visualizzati nell'ambiente RQT

Efficienza e prestazioni Le ottimizzazioni introdotte nel nodo *velodyne_light* lo rendono più prestazionale rispetto alla configurazione tradizionale, ridu-

cendo l'utilizzo delle risorse di sistema. Inoltre, il nodo risulta più intuitivo da configurare, richiedendo meno parametri di inizializzazione rispetto ai nodi Velodyne originali.

Capitolo 3

Calibrazione Estrinseca Automatica tra Camera Stereo e LiDAR

In questo capitolo sarà descritto il processo di calibrazione automatica estrinseca tra una telecamera e un sensore LiDAR, un aspetto cruciale per ottenere una corretta fusione dei dati sensoriali.

Il lavoro presentato si basa sull'utilizzo del metodo proposto nell'articolo *Automatic Extrinsic Calibration Method for LiDAR and Camera Sensor Setups* [1] che descrive una calibrazione estrinseca automatica realizzata su ROS. Tuttavia, durante il processo di utilizzo sono emerse diverse problematiche che hanno richiesto modifiche sostanziali al metodo, rendendolo più adeguato al caso d'uso specifico di questa tesi. Nel corso del capitolo, saranno spiegati sia il metodo di calibrazione originale che le modifiche apportate per migliorarne l'efficacia e adattarlo alle esigenze del sistema scelto.

3.1 Introduzione alla calibrazione estrinseca

La calibrazione estrinseca si riferisce al processo di determinazione della trasformazione geometrica tra due sistemi di riferimento distinti, in questo caso tra una telecamera e un sensore LiDAR. Tale trasformazione è composta da una rotazione e una traslazione che consentono di esprimere le coordinate di un oggetto nello spazio viste da uno dei sensori nel sistema di riferimento dell'altro. In altre parole, la calibrazione estrinseca definisce come i due sensori sono posizionati l'uno rispetto all'altro, permettendo di combinare in modo coerente i dati raccolti. Questo è particolarmente importante in applicazioni di fusione sensoriale, come la percezione 3D o la mappatura, dove informazioni visive e dati di profondità devono essere sincronizzati e allineati spazialmente per fornire una rappresentazione accurata dell'ambiente circostante.

3.2 Perchè effettuare una calibrazione estrinseca automatica

L'uso di una calibrazione estrinseca automatica tra camera e LiDAR offre numerosi vantaggi rispetto all'adozione di una trasformazione ottenuta attraverso misurazioni manuali. Sebbene le misurazioni manuali possano fornire una stima iniziale della trasformazione, esse tendono ad essere approssimative e suscettibili a errori umani, soprattutto quando i sensori sono montati in posizioni complesse o difficili da misurare con precisione. Anche piccole imprecisioni possono influenzare negativamente l'allineamento dei dati, portando a errori significativi.

Inoltre, le configurazioni dei sensori possono variare leggermente nel tem-

po a causa di vibrazioni o cambiamenti strutturali, rendendo necessario ri-calibrare periodicamente il sistema. La calibrazione automatica non solo migliora la precisione della trasformazione tra i sensori, ma permette anche di adattarsi dinamicamente a eventuali cambiamenti nella configurazione, eliminando la necessità di interventi manuali ripetuti. Questo porta a un sistema più robusto e flessibile, capace di mantenere un allineamento accurato in diverse condizioni operative.

3.3 L’articolo di riferimento

La selezione del metodo proposto nell’articolo [1] scelto rispetto ad altri articoli presi in considerazione come ad esempio [5] e [3], è stata influenzata da diversi fattori. In primo luogo, i pattern utilizzati in altri articoli non avrebbero fornito risultati adeguati a distanze maggiori, come richiesto dal caso d’uso. Inoltre, la maggior parte degli studi trattava LiDAR con un solo canale, mentre in questo progetto viene utilizzato un LiDAR multi-canale. Infine, il metodo era già implementato nel framework ROS, che rappresenta l’ambiente in cui i sensori vengono integrati e utilizzati.

In questa sezione verrà descritto il processo di calibrazione estrinseca tra una camera stereo e un sensore LiDAR, con particolare attenzione alla registrazione automatica dei punti di riferimento per ottenere la trasformazione rigida tra i due sensori. Tutti i passaggi descritti sono stati realizzati utilizzando ROS.

3.3.1 Metodo di calibrazione utilizzato

Il metodo di calibrazione proposto nell’articolo [1] utilizza un bersaglio di calibrazione personalizzato (fig. 3.1) dotato di caratteristiche geometriche



Figura 3.1: Pattern di calibrazione

e visive che possono essere rilevate sia dal LiDAR che dalla camera stereo. Il processo di calibrazione si svolge in due fasi principali: segmentazione del bersaglio e calcolo della trasformazione rigida. Il metodo è progettato per essere robusto e adattabile a configurazioni di sensori con differenti risoluzioni e posizioni relative.

3.3.2 Processo di calibrazione tra camera stereo e LiDAR

La calibrazione tra una camera stereo e un sensore LiDAR avviene attraverso i seguenti passaggi:

1. Segmentazione del bersaglio di calibrazione

Il primo passo del processo consiste nell'individuare il bersaglio di calibrazione sia nei dati della camera stereo che in quelli del LiDAR. Il bersaglio è progettato con caratteristiche specifiche, come cerchi per la segmentazione geometrica. Esso viene posizionato nell'area di sovrapposizione del campo visivo tra la camera stereo e il LiDAR.

2. Pre-elaborazione dei dati della camera stereo

Per la camera stereo, i dati grezzi dell'immagine vengono convertiti in una nuvola di punti 3D utilizzando un algoritmo di *stereo matching*, in questo caso la variante *Semi-Global Block Matching (SGBM)* implementata in OpenCV. Una volta ottenuta la nuvola di punti, vengono applicati filtri per rimuovere quelli che si trovano al di fuori dell'area in cui si prevede di trovare il bersaglio. Viene, quindi, eseguita una segmentazione dei bordi per identificare i contorni del bersaglio (fig. 3.2).

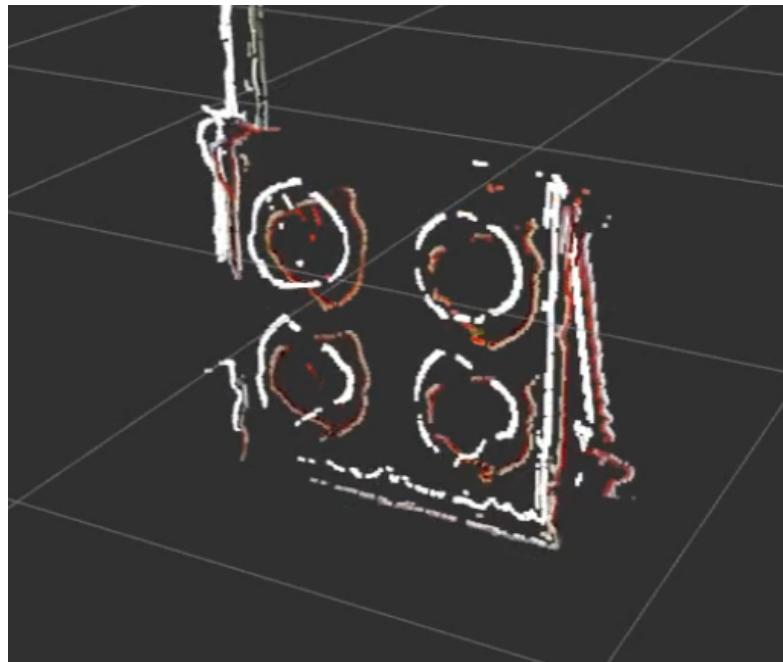


Figura 3.2: Punti del pattern trovati con SGBM, filtrati e proiettati nel piano

3. Pre-elaborazione dei dati del LiDAR

Similmente, i dati del LiDAR, rappresentati come una nuvola di punti 3D, vengono filtrati per rimuovere i punti esterni all'area di interesse. Successivamente, si esegue la segmentazione per rilevare i bordi del bersaglio basandosi sulle discontinuità di profondità nei dati LiDAR. Il metodo RANSAC viene utilizzato per trovare il piano su cui si trova il bersaglio e ridurre il set di dati a soli punti rilevanti per la calibrazione (fig. 3.3).

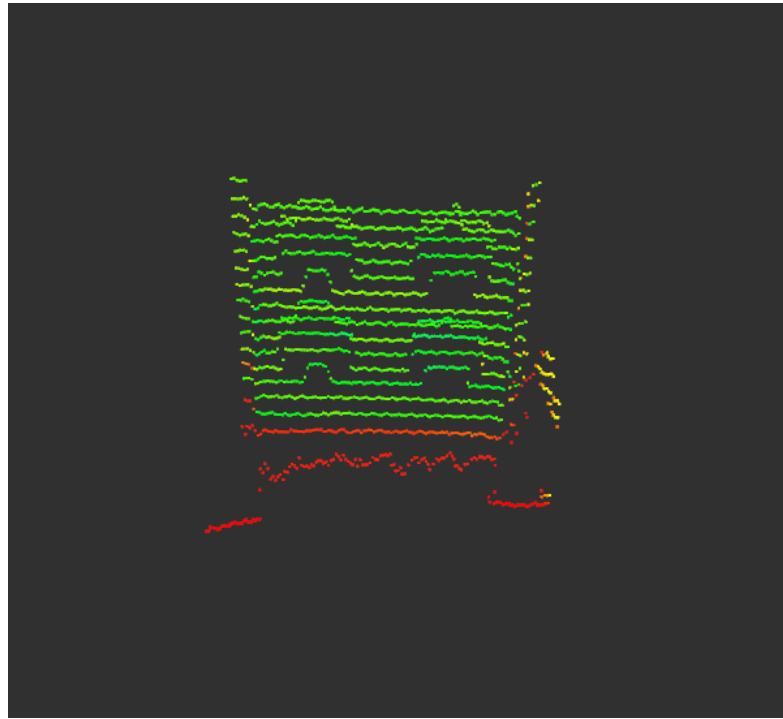


Figura 3.3: Punti del LiDAR appartenenti al pattern

4. Segmentazione dei cerchi del bersaglio

Dopo la segmentazione dei bordi, il passo successivo è identificare i cerchi presenti nel bersaglio, sia nei dati della camera stereo che del LiDAR. Questa segmentazione si basa su modelli geometrici e cerca di estrarre i centri dei cerchi come punti di riferimento. In questa fase vengono applicate verifiche di consistenza geometrica per garantire che i cerchi rilevati abbiano dimensioni coerenti con quelle del bersaglio (fig. 3.4).

5. Aggregazione dei punti di riferimento

Una volta ottenuti i punti di riferimento dai dati di entrambe le modalità sensoriali, questi vengono aggregati in due nuvole di punti separate, una per la camera stereo e una per il LiDAR. La robustezza del metodo viene migliorata

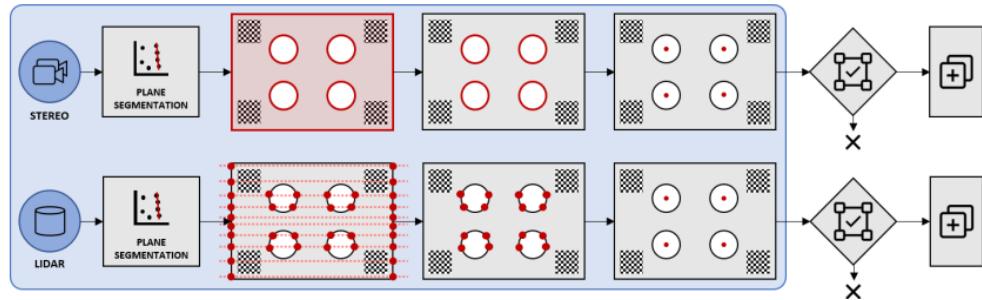


Figura 3.4: Processo di acquisizione dei punti del LiDAR e della Camera

attraverso l’accumulo dei dati in più fotogrammi, aumentando l’accuratezza della localizzazione dei punti di riferimento.

6. Registrazione e calcolo della trasformazione rigida

L’ultimo passo consiste nel calcolare la trasformazione rigida ottimale che allinea i punti di riferimento della camera stereo con quelli del LiDAR utilizzando il metodo di Umeyama. Il risultato finale è la trasformazione rigida che definisce la posizione e l’orientamento relativo tra la camera stereo e il LiDAR.

Calcolo della trasformazione

Una volta ottenuti i punti di riferimento sia dalla camera stereo che dal LiDAR, il passaggio successivo è la registrazione (*registration*) dei due set di punti per calcolare la trasformazione rigida che definisce la posizione relativa tra i sensori. Il processo di registrazione consiste nel trovare la trasformazione ottimale che minimizza la distanza tra i punti corrispondenti nei due set, ovvero tra i punti 3D rilevati dalla camera e quelli rilevati dal LiDAR.

Nel processo di registrazione dei punti, è possibile modificare la posizione del pattern rispetto ai sensori, a condizione che ciò venga specificato nei parametri di configurazione. Questo approccio consente di acquisire i

punti per una certa posizione del pattern, interrompere temporaneamente l’acquisizione, spostare il pattern e poi riavviare l’acquisizione.

Questa procedura di registrazione consente di ottenere nuvole di punti provenienti da diverse posizioni all’interno dell’ambiente. Tale strategia non solo arricchisce il set di dati acquisiti, ma migliora anche significativamente l’accuratezza della trasformata.

Il metodo utilizzato per la registrazione è l’algoritmo di Umeyama, che ottimizza la trasformazione rigida tridimensionale cercando di allineare i due set di punti attraverso una combinazione di rotazione e traslazione. La trasformazione è descritta da una matrice \mathbf{T} , che contiene una matrice di rotazione \mathbf{R} e un vettore di traslazione \mathbf{t} :

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix}$$

dove $\mathbf{R} \in \mathbb{R}^{3 \times 3}$ rappresenta la matrice di rotazione e $\mathbf{t} \in \mathbb{R}^3$ è il vettore di traslazione.

Il criterio di ottimizzazione utilizzato minimizza l’errore quadratico medio tra i punti corrispondenti nei due set. L’errore è dato dalla somma delle distanze al quadrato tra i punti $\mathbf{p}_i^{\text{cam}}$ (punti della camera) e $\mathbf{p}_i^{\text{LiDAR}}$ (punti del LiDAR) dopo l’applicazione della trasformazione:

$$\text{errore} = \frac{1}{N} \sum_{i=1}^N \| \mathbf{R} \mathbf{p}_i^{\text{cam}} + \mathbf{t} - \mathbf{p}_i^{\text{LiDAR}} \|^2$$

dove N è il numero di punti di corrispondenza tra i due sensori.

L’algoritmo di Umeyama è particolarmente adatto per situazioni in cui i punti di riferimento sono coplanari, come nel caso della calibrazione tra camera stereo e LiDAR, evitando ambiguità nelle configurazioni con punti 2D o 3D allineati. Una volta ottenuta la matrice di trasformazione \mathbf{T} , questa

può essere applicata per convertire i punti rilevati dalla camera nel sistema di riferimento del LiDAR.

3.4 Criticità riscontrate nell'articolo

L'idea iniziale era di applicare il metodo descritto nell'articolo per calcolare la trasformazione tra il LiDAR e la camera stereo. Tuttavia, non è stato possibile replicare i risultati ottenuti nello studio originale. La difficoltà principale è stata nell'acquisizione dei punti dalla camera stereo: la nuvola di punti generata conteneva un'elevata quantità di rumore, rendendo estremamente complessa l'identificazione di un piano che attraversasse correttamente la nuvola stessa. Nelle rare occasioni in cui il piano veniva individuato, una volta riportati i punti su di esso, era comunque impossibile segmentare correttamente i cerchi necessari per la calibrazione (fig. 3.5).

Questo problema si accentuava ulteriormente quando il pattern di calibrazione si trovava a più di un metro e mezzo di distanza dalla camera. Tale limitazione è particolarmente critica nella configurazione in esame, dove la camera è posizionata a più di 1,5 metri dietro al LiDAR. Inoltre, poiché il LiDAR a 16 canali non riesce a rilevare l'intero pattern a distanze inferiori a 3 metri, la camera deve essere in grado di riconoscere i cerchi a una distanza di almeno 4,5 metri.

Queste problematiche hanno portato a cercare una soluzione alternativa, modificando il metodo di acquisizione dei punti da parte della camera, al fine di ottenere una maggiore robustezza anche a distanze maggiori.

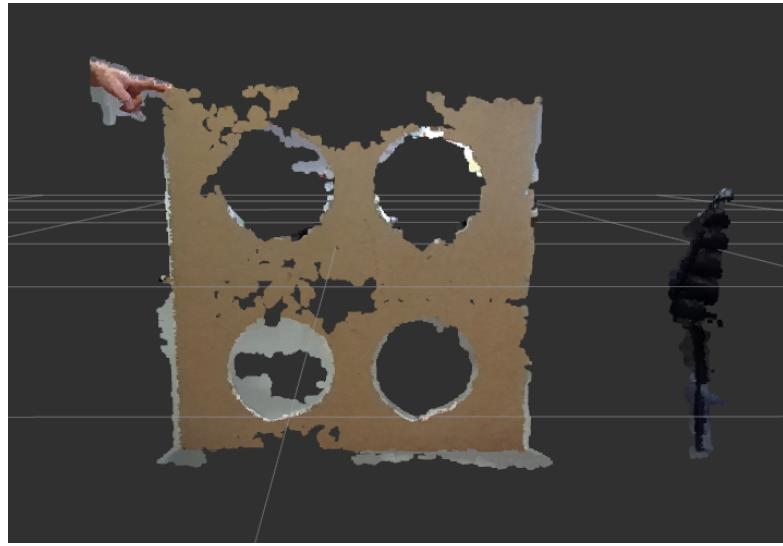


Figura 3.5: Pattern riconosciuto dalla camera a 1,5m

3.5 La soluzione proposta

Per risolvere le problematiche emerse durante la calibrazione, sono state apportate due modifiche principali al processo originale. La prima riguarda il pattern di calibrazione: è stato scelto di allargare i buchi presenti nel pattern per permettere a un numero maggiore di fasci del LiDAR di attraversarli, anche con uno a bassa risoluzione, come quello a 16 canali impiegato in questo lavoro.

La seconda modifica ha riguardato il processo di acquisizione dei punti da parte della camera stereo, che è stato completamente ripensato per garantire un migliore riconoscimento dei cerchi. Il nuovo metodo si sviluppa attraverso i seguenti passaggi:

- 1. Acquisizione delle immagini:** Inizialmente, vengono acquisite le immagini sia dalla camera sinistra che dalla camera destra del sistema stereo. Queste immagini rappresentano la base da cui estrarre i cerchi presenti nel pattern di calibrazione.



Figura 3.6: Immagine dopo l'applicazione del filtro blur e di Canny

2. **Applicazione di un filtro *blur*:** Per ridurre il rumore presente nelle immagini e facilitare la rilevazione dei bordi, viene applicato un filtro di *blur* (sfocatura) su entrambe le immagini. Questo passaggio è essenziale per migliorare la qualità dell'immagine e ottenere contorni più definiti, riducendo il rischio di falsi positivi nella fase di rilevazione dei cerchi.
3. **Rilevazione dei bordi con il filtro di Canny:** Dopo aver sfocato l'immagine, viene utilizzato il filtro di Canny per rilevare i bordi (fig. 3.6). Questo algoritmo consente di identificare i contorni più netti all'interno dell'immagine, che saranno poi utilizzati per la successiva rilevazione dei cerchi.
4. **Trasformata di Hough per la rilevazione dei cerchi:** Una volta ottenuti i contorni tramite il filtro di Canny, viene applicata la Trasformata di Hough per identificare i cerchi nelle immagini. Questa tecnica è particolarmente efficace per individuare le forme circolari all'interno dell'immagine, poiché rileva i cerchi basandosi sulle curve individuate nei bordi (fig. 3.7).

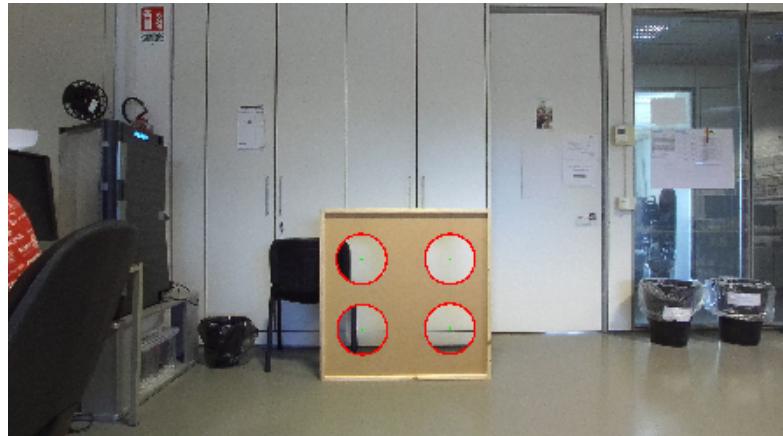


Figura 3.7: Cerchi rilevati dalla Trasformata di Hough

5. **Verifica dei cerchi rilevati:** Dopo aver individuato i cerchi in entrambe le immagini, vengono confrontati i risultati per verificare che i cerchi trovati appartengano effettivamente al pattern di calibrazione. A questo scopo, controllo la coerenza geometrica tra i cerchi rilevati e le dimensioni attese del pattern.
6. **Creazione delle corrispondenze tra le immagini:** Una volta verificati i cerchi, vengono ordinati i centri rilevati e stabilite le corrispondenze tra i cerchi individuati nell'immagine di sinistra e quelli nell'immagine di destra. Questo passaggio è cruciale per garantire che le coppie di cerchi si corrispondano esattamente tra le due immagini, requisito fondamentale per la successiva ricostruzione 3D.
7. **Calcolo delle coordinate 3D:** Una volta individuati i centri dei cerchi nelle immagini della camera stereo e ordinati, è possibile calcolare le coordinate tridimensionali di tali punti utilizzando la triangolazione. Per eseguire questo calcolo, si parte dalle coordinate dei centri dei cerchi in entrambe le immagini (sinistra e destra) e si utilizza la geometria del sistema stereo.

Il processo di triangolazione si basa sulla differenza di posizione dei centri dei cerchi nelle due immagini, chiamata *disparità*. Dato un punto (u_L, v_L) nell'immagine di sinistra e un punto corrispondente (u_R, v_R) nell'immagine di destra, la disparità d è data dalla differenza tra le coordinate orizzontali dei due punti:

$$d = u_L - u_R$$

La profondità Z del punto nel mondo reale può essere calcolata utilizzando la disparità, la lunghezza focale f e la distanza tra le due telecamere (detta *baseline* B):

$$Z = \frac{f \cdot B}{d}$$

Una volta nota la profondità Z , le coordinate tridimensionali del punto possono essere calcolate in base alle sue coordinate nelle immagini e alla matrice intrinseca della camera \mathbf{K} . Le coordinate tridimensionali (X, Y, Z) del punto sono ottenute come segue:

$$X = \frac{(u_L - c_x) \cdot Z}{f}, \quad Y = \frac{(v_L - c_y) \cdot Z}{f}$$

dove (c_x, c_y) sono le coordinate del centro ottico della camera, ottenute dalla matrice intrinseca \mathbf{K} , e f è la lunghezza focale, anch'essa presente in \mathbf{K} . Questo calcolo consente di trasformare i punti trovati nelle immagini in coordinate tridimensionali nel sistema di riferimento della camera stereo, pronte per essere utilizzate nel processo di calibrazione con il LiDAR.

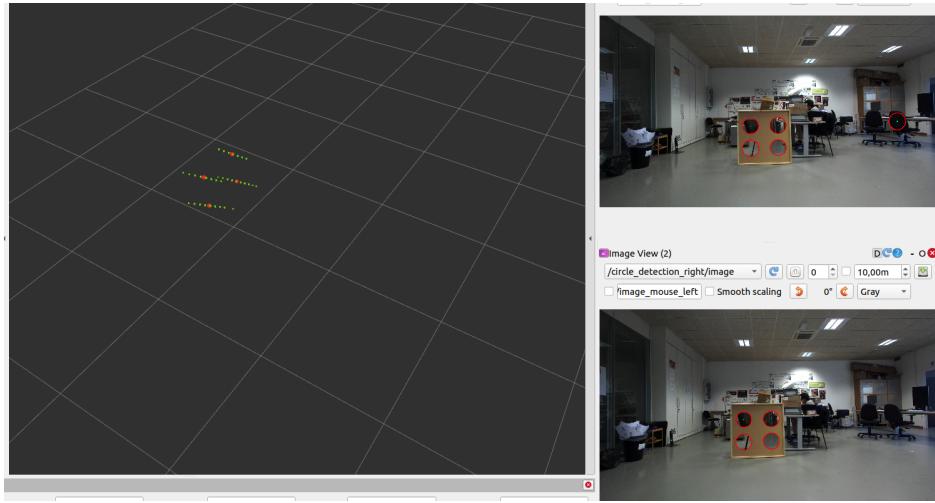


Figura 3.8: Visualizzazione su RQT dell’immagine destra e sinistra e le nuvole di punti trovate nel 3D

8. Calcolo del centro delle nuvole di punti: Nella soluzione proposta, per migliorare la precisione del calcolo delle coordinate 3D dei cerchi del pattern, vengono acquisiti più frame consecutivi della scena. Per ogni frame, vengono estratti i punti 3D corrispondenti ai cerchi rilevati nel pattern. Questo approccio consente di generare una nuvola di punti per ciascun cerchio.

Successivamente, viene calcolata una media ponderata delle nuvole di punti ottenute dai diversi frame. Il risultato finale è un singolo punto 3D che rappresenta il centro della nuvola di punti per ciascun cerchio del pattern. Questo processo riduce l’effetto del rumore e di eventuali errori nelle singole stime dei punti (fig. 3.8).

3.6 Risultati del metodo di calibrazione

L’obiettivo principale della calibrazione tra camera e LiDAR è quello di proiettare i punti 3D acquisiti dal LiDAR nel piano immagine della ca-

mera, rendendo così possibile l'integrazione dei dati LiDAR in algoritmi di visual-SLAM. Per verificare l'accuratezza della proiezione, sono stati eseguiti i seguenti test:

- Inizialmente, sono state determinate le coordinate tridimensionali dei centri dei cerchi del pattern e i rispettivi raggi nella nuvola di punti generata dal LiDAR.
- Successivamente, i centri individuati sono stati proiettati nel piano immagine della camera sinistra utilizzando la matrice estrinseca trovata tramite la calibrazione e la matrice intrinseca della camera.
- Parallelamente, tramite uno script Python, è stato possibile selezionare manualmente i cerchi del pattern nell'immagine della camera utilizzando il mouse. Questo ha permesso di rilevare direttamente i centri e i raggi dei cerchi nel piano immagine, offrendo un confronto diretto con i dati proiettati dal LiDAR.
- Una volta ottenute le coordinate e i raggi dei cerchi sia dalla proiezione LiDAR che dall'immagine della camera, è stata eseguita un'analisi comparativa. I cerchi rilevati con entrambi i metodi sono stati sovrapposti (fig. 3.9), e per valutarne la congruenza è stato calcolato il valore di *Intersection over Union* (IoU) tra i cerchi. Questo parametro fornisce una misura quantitativa della sovrapposizione tra i cerchi, offrendo così una valutazione dell'accuratezza della proiezione dei punti 3D nel piano immagine.

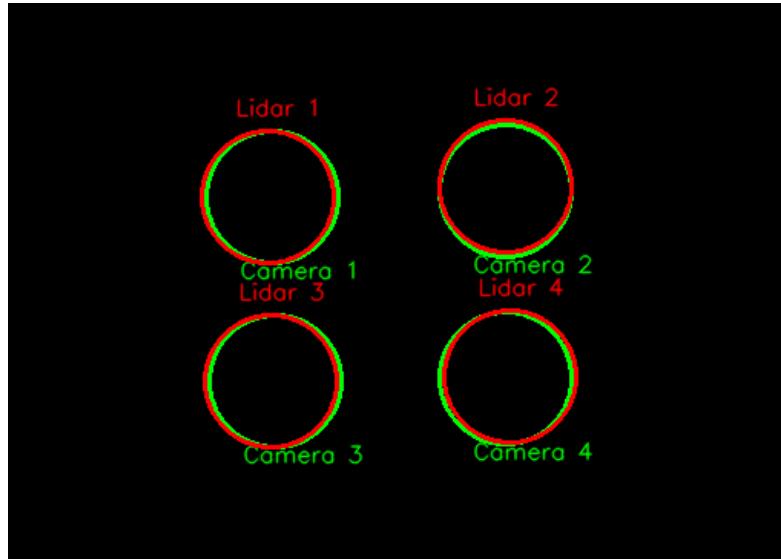


Figura 3.9: Esempio della sovrapposizione dei cerchi con un IoU del 93%

3.7 Miglioramento dell'Intersection over Union con l'aumentare delle posizioni del pattern

Per valutare l'accuratezza della calibrazione all'aumentare del numero di posizioni del pattern, è stato tracciato un grafico (fig. 3.10) che mostra l'evoluzione dell'*Intersection over Union* (IoU) in funzione del numero di posizioni rilevate. Dai risultati emerge che dopo aver considerato almeno tre diverse posizioni del pattern, la trasformata risulta più accurata. In particolare, si nota che a partire dalla terza posizione l'IoU si stabilizza.

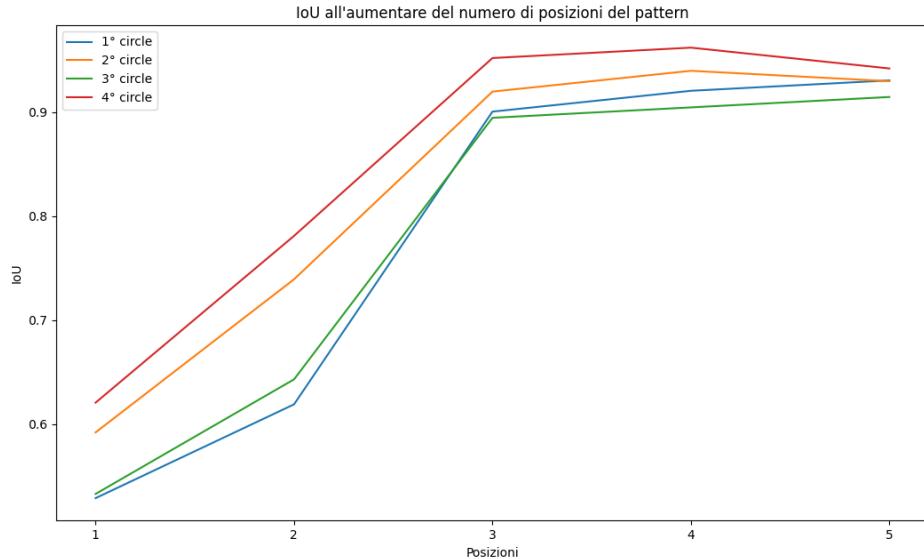


Figura 3.10: Miglioramento dell’IoU all’aumentare del numero di posizioni del pattern

3.8 Confronto con misurazioni manuali

Per valutare ulteriormente l’efficacia del metodo di calibrazione automatica proposto, è stato condotto un test comparativo con una trasformazione ottenuta tramite misurazioni manuali dei parametri di traslazione e rotazione tra i due sensori. I risultati, riportati nella tabella seguente (tab. 3.1), mostrano chiaramente che l’approccio manuale porta a risultati significativamente inferiori rispetto alla calibrazione automatica (fig. 3.11).

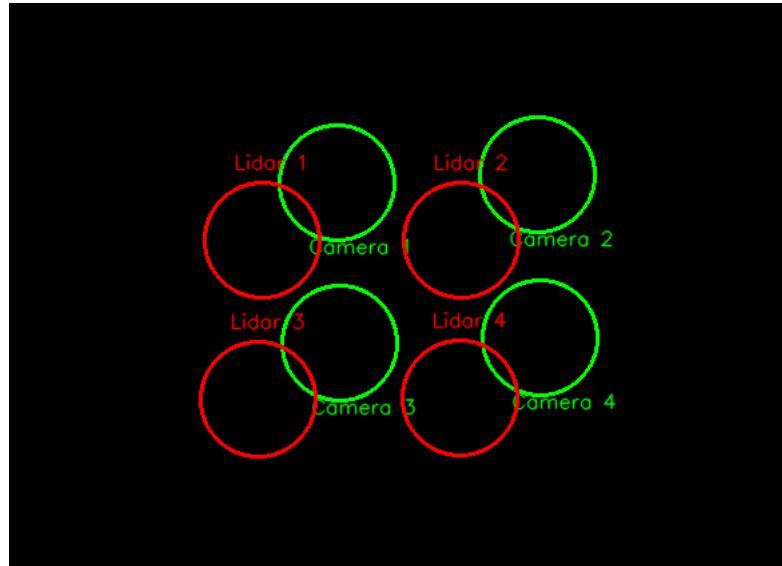


Figura 3.11: Esempio di sovrapposizione dei cerchi con un IoU del 4%

| Cerchio | Trasformata automatica(IoU) | Trasformata con valori reali(IoU) |
|---------|-----------------------------|-----------------------------------|
| 1 | 0.920 | 0.055 |
| 2 | 0.939 | 0.038 |
| 3 | 0.904 | 0.035 |
| 4 | 0.961 | 0.034 |

Tabella 3.1: Comparazione dei risultati tra la trasformata trovata con il metodo proposto e la trasformata utilizzando i valori reali della posizione dei due sensori.

Questo test evidenzia un aspetto cruciale: il calcolo automatico della trasformazione risulta non solo più preciso, ma anche più pratico, specialmente in situazioni in cui il setup dei sensori (camera e LiDAR) viene frequentemente smontato e rimontato. In questi casi, l'uso di una trasformazione calcolata direttamente dai sensori permette di ottenere una configurazione accurata in modo più rapido ed efficiente.

3.9 Proiezione dei punti LiDAR nell’immagine della camera sinistra

Per verificare visivamente i risultati della calibrazione estrinseca tra LiDAR e camera, è stato implementato un metodo di proiezione dei punti LiDAR nel piano immagine della camera sinistra (fig. 3.12). Il processo si articola in due fasi principali:

- **Trasformazione dei punti LiDAR nel sistema di riferimento della camera sinistra:** I punti 3D del LiDAR, inizialmente definiti nel sistema di riferimento del sensore LiDAR, vengono trasformati nel sistema di riferimento della camera sinistra. Questa trasformazione è ottenuta utilizzando la matrice estrinseca $\mathbf{T}_{\text{LiDAR,Camera}}$ prodotta tramite il processo di calibrazione automatica.
- **Proiezione nel piano immagine della camera:** Successivamente, i punti trasformati vengono proiettati nel piano immagine della camera sinistra. Questo avviene moltiplicando i punti 3D trasformati per la matrice intrinseca della camera $\mathbf{K}_{\text{Camera}}$, che mappa i punti 3D dalle coordinate del mondo alle coordinate dell’immagine. La relazione è data dalla seguente equazione:

$$\mathbf{p}_{\text{immagine}} = \mathbf{K}_{\text{Camera}} \cdot \mathbf{T}_{\text{LiDAR,Camera}} \cdot \mathbf{p}_{\text{LiDAR}}$$

dove $\mathbf{p}_{\text{LiDAR}}$ sono le coordinate dei punti nel sistema di riferimento del LiDAR, $\mathbf{T}_{\text{LiDAR,Camera}}$ è la matrice estrinseca (rototraslazione) ottenuta dalla calibrazione, $\mathbf{K}_{\text{Camera}}$ è la matrice intrinseca della camera, $\mathbf{p}_{\text{immagine}}$ sono le coordinate proiettate nel piano immagine.



Figura 3.12: Punti del LiDAR proiettati sul piano dell’immagine, il colore rappresenta la profondità dal rosso al blu.

3.10 Comparazione tra le due soluzioni

Non è stato possibile eseguire una comparazione diretta tra il metodo proposto nell’articolo [1] di riferimento e la soluzione sviluppata.

Ciò è avvenuto a causa delle limitazioni imposte dall’hardware disponibile e dalle specifiche esigenze del caso d’uso, che hanno reso impraticabile la replicazione dei risultati ottenuti con il metodo originale. La natura dei sensori utilizzati e le condizioni operative hanno reso l’approccio dell’articolo [1] meno applicabile.

Tuttavia, il metodo sviluppato ha dimostrato una maggiore robustezza nel rilevare i centri dei cerchi nelle immagini della camera, in particolare a distanze maggiori. L’impiego della media delle nuvole di punti ha permesso di ottenere una stima più stabile e precisa, contribuendo a ridurre l’impatto del rumore e delle variazioni dovute al movimento o alle incertezze nei ri-

levamenti. Questo ha portato a risultati più consistenti rispetto al metodo descritto in [1], specialmente in queste specifiche condizioni d'uso.

Capitolo 4

Idee e Sviluppi futuri: Integrazione dei punti LiDAR su ORB_SLAM3

In questo capitolo viene illustrato come il lavoro svolto fino a questo punto possa essere utilizzato per integrare i dati del LiDAR all'interno di un algoritmo di visual-SLAM, come ORB_SLAM3 [2]. L'obiettivo è sfruttare i vantaggi offerti dalla combinazione di dati visivi e LiDAR, ottenendo una mappatura e una localizzazione più accurate e robuste, specialmente in ambienti complessi o poco strutturati. L'integrazione dei punti LiDAR permette di arricchire le informazioni di profondità, migliorando la stima della posizione e riducendo l'impatto di condizioni ambientali difficili, come scarsa illuminazione o texture ripetitive, dove gli algoritmi basati unicamente sulla visione possono incontrare difficoltà.

L'integrazione dei dati del LiDAR con quelli della telecamera consente di ottenere informazioni più complete: mentre la telecamera fornisce dettagli di colore e texture, il LiDAR offre una visione geometrica dettagliata

dell’ambiente circostante. Questo approccio combinato riduce gli errori di localizzazione e migliora la qualità della mappa generata, specialmente in ambienti complessi o scarsamente illuminati.

4.1 Introduzione a ORB_{_}SLAM3

ORB_{_}SLAM3 [2] è un sistema di SLAM (Simultaneous Localization and Mapping) avanzato che supporta vari tipi di sensori, tra cui telecamere monoculari, stereo e RGB-D. Quando viene utilizzato con una camera stereo, ORB_{_}SLAM3 [2] sfrutta le informazioni di profondità direttamente ricavate dalla disparità tra le immagini catturate dalle due lenti.

Una delle principali caratteristiche di ORB_{_}SLAM3 [2] è l’uso di caratteristiche ORB (Oriented FAST and Rotated BRIEF) per il rilevamento e la descrizione dei punti chiave nell’immagine. Questi punti chiave sono robusti alle variazioni di illuminazione e rotazione, rendendo il sistema adatto per ambienti reali complessi. ORB_{_}SLAM3 [2] utilizza una pipeline di tre principali fasi:

- **Tracking:** I punti chiave vengono rilevati in tempo reale e associati tra le immagini stereo. Questa fase è cruciale per stimare la posizione e l’orientamento della camera in ogni istante.
- **Local Mapping:** Le nuove osservazioni vengono integrate nella mappa locale. La triangolazione stereo viene sfruttata per calcolare con precisione le coordinate 3D dei punti chiave rilevati, permettendo di arricchire la mappa dell’ambiente circostante.
- **Loop Closure:** Questa fase permette di ridurre la deriva del sistema, rilevando e correggendo eventuali errori quando la camera ritorna in

una posizione precedentemente esplorata. Ciò garantisce che la mappa globale rimanga consistente e accurata nel tempo.

4.2 Idea per integrare i punti del LiDAR nell'algoritmo

Per integrare i punti del LiDAR in ORB_SLAM3 [2], è stato deciso di adottare un metodo semplice e veloce che consentisse l'impiego dell'algoritmo in breve tempo. L'idea principale consiste nell'utilizzare i punti del LiDAR per migliorare la stima della profondità delle feature ORB. A un certo istante di tempo, si dispone delle due immagini stereo (destra e sinistra) e di una scansione del LiDAR. ORB_SLAM3 [2] analizza le immagini e calcola le feature ORB, identificando i KeyPoints all'interno delle immagini. La profondità dei KeyPoints viene inizialmente calcolata tramite la triangolazione stereo delle camere. Contemporaneamente, i punti del LiDAR vengono trasformati nel frame ottico della camera utilizzando la matrice estrinseca \mathbf{T}_{CL} . Questa trasformazione avviene secondo l'equazione

$$\mathbf{P}_C = \mathbf{T}_{CL} \cdot \mathbf{P}_L$$

dove \mathbf{P}_L è il punto 3D nel frame del LiDAR e \mathbf{P}_C è il punto trasformato nel frame della camera. Una volta trasformati, i punti del LiDAR vengono proiettati sul piano dell'immagine della telecamera sinistra attraverso la matrice intrinseca \mathbf{K} , seguendo la formula $\mathbf{p}_i = \mathbf{K} \cdot \mathbf{P}_C$. La profondità z_i del punto nel frame della camera viene memorizzata come z_{LiDAR} .

Il processo di miglioramento della stima della profondità avviene nel seguente modo:

- Per ogni KeyPoint, si controlla se esiste un punto del LiDAR all'interno di una finestra di $N \times N$ pixel centrata sul KeyPoint.
- Se la finestra non contiene alcun punto del LiDAR, viene mantenuta la distanza calcolata tramite la camera.
- Se la finestra contiene uno o più punti del LiDAR, viene calcolato l'errore sulla profondità tra la distanza calcolata dalla camera z_{Camera} e quella del LiDAR z_{LiDAR} tramite la seguente formula:

$$e_z = |z_{Camera} - z_{LiDAR}|$$

dove e_z rappresenta l'errore assoluto sulla profondità.

- Se l'errore e_z è inferiore a una soglia prestabilita T_{depth} , la profondità calcolata dalla camera viene sostituita con quella calcolata dal LiDAR. Altrimenti, si continua a usare la profondità calcolata dalla camera.
- Se sono presenti più punti del LiDAR all'interno della finestra, si seleziona quello con l'errore e_z minore tra tutti quelli che soddisfano il *threshold*.

4.3 Sincronizzazione dei frame

Un aspetto cruciale da considerare nell'integrazione dei sensori è la sincronizzazione dei frame, poiché la camera stereo e il LiDAR operano a frequenze diverse. Tipicamente, il LiDAR ha un frame rate più basso rispetto alla camera stereo. Nella sezione precedente è stato ipotizzato di avere contemporaneamente a disposizione le immagini stereo e la scansione del LiDAR, ma in realtà ciò non accade frequentemente.

Nel contesto preso in analisi, ORB_SLAM3 [2] funziona con un frame rate di circa 30 FPS (Frame per Second), mentre il LiDAR, configurato per operare alla massima velocità, raggiunge un massimo di 20 FPS. Per sincronizzare i due sensori è necessario trovare una corrispondenza tra i frame generati dai due dispositivi. A tale scopo, è possibile sfruttare l'header dei messaggi ROS, che contiene il timestamp preciso in cui ogni messaggio è stato creato. Questa strategia sarebbe efficace se i due sensori operassero con lo stesso frame rate, ma poiché le frequenze sono diverse, è necessario che il sensore più veloce si adegui al ritmo del sensore più lento.

Tuttavia, questo approccio presenta un'importante limitazione: i frame generati dal sensore più veloce, che non vengono utilizzati, rappresentano dati potenzialmente utili che vengono persi. Per massimizzare l'utilizzo dei dati disponibili, è stata ideata una soluzione che permette alla camera di operare al massimo frame rate possibile. La sincronizzazione avviene sempre tramite il timestamp dell'header dei messaggi ROS, ma i punti del LiDAR vengono utilizzati solo quando è disponibile un frame del LiDAR che differisce di un intervallo di tempo inferiore a una soglia prestabilita rispetto al frame della camera. Se non c'è una corrispondenza adeguata, l'algoritmo continua a funzionare utilizzando solo i dati della camera, senza l'integrazione dei punti del LiDAR.

4.4 Conclusioni

In questo capitolo è stato presentato un metodo preliminare per l'integrazione dei dati LiDAR nell'algoritmo ORB_SLAM3 [2], con l'obiettivo di migliorare la stima della profondità delle feature ORB. Il procedimento sfrutta i punti LiDAR per affinare le stime di profondità derivate dalla trian-

golazione stereo, trasformando i punti nel frame ottico della camera, come descritto nella calibrazione del capitolo precedente (cap. 3), e successivamente proiettandoli nel piano dell'immagine.

Il metodo di sincronizzazione tra i sensori, basato sui timestamp ROS, così come l'integrazione dei dati LiDAR, richiede ancora una validazione sperimentale e un'analisi dettagliata per valutarne l'efficacia complessiva.

Questo lavoro costituisce un primo passo verso l'integrazione dei sensori LiDAR in ORB_SLAM3, ma ulteriori sviluppi e verifiche sono necessari per ottimizzare la sincronizzazione e per determinare l'impatto dei dati LiDAR sulla mappatura e il tracking in scenari dinamici.

In conclusione, la calibrazione tra LiDAR e camera effettuata nel capitolo tre (cap. 3) ha fornito una base solida per lo sviluppo futuro dell'integrazione sensoriale, sebbene il metodo proposto sia ancora in fase di sperimentazione e richieda ulteriori verifiche.

Bibliografia

- [1] Jorge Beltrán, Carlos Guindel, Arturo de la Escalera, and Fernando García. Automatic extrinsic calibration method for lidar and camera sensor setups. *IEEE Transactions on Intelligent Transportation Systems*, 2022.
- [2] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.
- [3] Zhen Chen, Liang Zhuo, Kaiqiong Sun, and Congxuan Zhang. Extrinsic calibration of a camera and a laser range finder using point to line constraint. *Procedia Engineering*, 29:4348–4352, 2012. 2012 International Workshop on Information and Electronics Engineering.
- [4] Emanuele Nencioni. Sviluppo e valutazione di un sistema slam basato su orb_slam3 su ros, 2022. https://github.com/emanuelenencioni/ORB_SLAM3_ROS/blob/main/doc/documentation.pdf.
- [5] Lipu Zhou, Zimo Li, and Michael Kaess. Automatic extrinsic calibration of a camera and a 3d lidar using line and plane correspondences. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5562–5569, 2018.