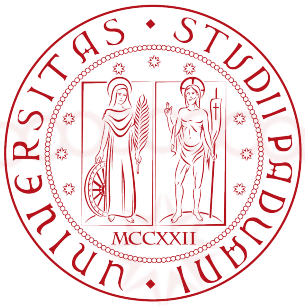


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di laurea magistrale in INGEGNERIA INFORMATICA

TESI DI LAUREA MAGISTRALE

ALGORITMO DI RILEVAZIONE DI PERSONE A TERRA DA DATI 3D PER ROBOT DI TELEPRESENZA

Relatore: Prof. EMANUELE MENEGATTI

Laureando: MARCO PIEROBON

Correlatori: Dott. MORRIS ANTONELLO

Dott. MARCO CARRARO

ANNO ACCADEMICO 2015-2016

Abstract

Questa tesi si costituisce come parte del progetto di un robot di telepresenza per l'assistenza agli anziani.

La prima tematica affrontata riguarderà il miglioramento delle prestazioni di mappatura e navigazione in ambienti interni. A causa della bassa altezza del sensore Laser Range Finder (LRF), il robot rischia di collidere con alcuni componenti comuni dell'arredamento, quali tavoli e sedie. Per garantire l'assenza di collisioni sarà quindi necessario integrare le informazioni provenienti dal sensore LRF con quelle tridimensionali ottenute da una telecamera Kinect One, montata nel punto più alto del robot.

Quindi si provvederà ad ideare un algoritmo che consenta la rilevazione di persone cadute, sfruttando i dati 3D ottenuti dalla Kinect One. La soluzione individuata consiste in un sistema intelligente costituito da due classificatori operanti in serie. In particolare saranno discusse approfonditamente alcune scelte di importanza centrale effettuate nella messa a punto del secondo classificatore. L'algoritmo ottenuto si dimostra efficace ed applicabile anche in scenari scarsamente illuminati o popolati da numerosi ostacoli.

Sarà inoltre presentato un metodo per effettuare automaticamente l'annotazione delle performance basato sulla mappa dell'ambiente. Il vantaggio di tale approccio è quello di evitare il ricorso all'annotazione manuale di ogni singolo frame.

Infine si cercherà di ridurre il numero di falsi allarmi sfruttando la conoscenza a priori della mappa dell'ambiente nonché i molti frame acquisiti da diversi punti di vista durante la navigazione del robot. I rilevamenti ottenuti nei singoli frame saranno quindi rappresentati sulla mappa e raggruppati in base alla distanza. Sfruttando l'informazione derivante dal numero dei componenti del gruppo e dall'istante in cui ogni componente è stato generato, si deciderà se confermare i rilevamenti o considerarli un falso allarme.

Indice

Capitolo 1

Introduzione

Nel corso degli ultimi 25 anni si è registrato un costante aumento dell'età media della popolazione del pianeta, fenomeno che tutt'ora non accenna a fermarsi. Studi di settore prevedono infatti una crescita della percentuale di popolazione di età superiore ai 65 anni dall'attuale 8% al 16% entro il 2050 [1].

In un simile scenario, sta ottenendo importanza sempre più crescente la ricerca di infrastrutture tecnologiche che permettano agli anziani di vivere autonomamente a casa loro. Lo scopo di tali sistemi, detti di "Ambient Assisted Living" (o AAL), è quello di evitare il costoso ricorso ad assistenti umani o al trasferimento in case di riposo. L'importanza della ricerca di soluzioni di tal sorta è stata sottolineata anche nel report del President's Council of Advisors on Science and Technology (PCAST) del 15 marzo 2016 [2]; in tale documento sono analizzati ed esposti al presidente degli Stati Uniti la situazione attuale e i progetti di ricerca riguardo a indipendenza, tecnologia e connessione delle persone in età avanzata.

Il principale problema da affrontare per permettere a persone anziane o disabili una vita il più possibile autonoma e sicura riguarda la gestione delle emergenze. Un sistema di AAL deve innanzitutto poter garantire la capacità di riconoscere il maggior numero possibile di situazioni di pericolo di vita in cui possono trovarsi le persone monitorate. Quindi deve provvedere efficacemente ad allertare i soccorsi ed i familiari.

Una possibile soluzione di tal sorta, citata anche nel report PCAST, prevede di ricorrere ad un robot di servizio, il quale monitora la situazione all'interno della casa, verificando periodicamente lo stato dei suoi occupanti. Una piattaforma robotica può essere preferibile ad altre soluzioni AAL per diversi motivi. Innanzitutto non richiede che vengano apportate modifiche radicali all'ambiente domestico, né vincola le persone sotto controllo ad indossare dispositivi di monitoraggio. Inoltre può offrire agli occupanti della casa non solo un fondamentale supporto in caso di emergenza, ma anche un utile strumento di comunicazione facilitata con familiari lontani; può offrire servizi di notifica ad orari prestabiliti per l'assunzione di medicine e lo svolgimento di routine quotidiane; può essere sfruttato come supporto alla deambulazione.

Questa tesi affronta principalmente due problematiche connesse all'utilizzo di un robot di telepresenza progettato per svolgere alcune delle funzionalità sopra elencate.

La prima riguarda l'effettiva capacità di navigare in sicurezza in un ambiente domestico. Per fare ciò il robot fornito implementa una funzione di mappatura, che gli permette di ricreare la mappa dell'area dove andrà ad operare; tale mappa viene quindi sfruttata per evitare gli ostacoli circostanti e orientarsi tra le diverse stanze della casa. In questa sede sarà proposto un metodo per migliorare le capacità di mappatura e navigazione iniziali del robot fornito, ancora troppo inaffidabili per consentirne l'uso in ambienti dinamici e variegati come quelli domestici. Queste funzionalità, infatti, inizialmente si basano solo sull'utilizzo dei dati provenienti dal sensore laser. A causa dell'altezza alla quale è montato quest'ultimo e della forma del robot, le prestazioni di navigazione risultano insoddisfacenti, in quanto il dispositivo tende a collidere con diversi oggetti presenti nell'ambiente in cui opera, quali tavoli e sedie. Sarà quindi necessario sfruttare anche i dati tridimensionali ottenibili dal sensore RGB-D montato sul robot. In questo modo sarà possibile creare mappe comprendenti tutti gli ostacoli a rischio di collisione e, di conseguenza, garantire la sicurezza della navigazione.

La seconda problematica affrontata riguarda il compito di individuare le cadute, una

situazione di emergenza ricorrente nella maggior parte dei casi di ricovero di persone di età superiore ai 65 anni [3]. Viene quindi proposto un algoritmo ideato appositamente per svolgere compiti di rilevamento di persone a terra con il robot in dotazione. Questo si costituisce come la sintesi tra due algoritmi precedentemente sperimentati, entrambi troppo poco efficaci per essere candidabili all'uso in scenari reali.

Il punto di incontro tra i due ambiti descritti è rappresentato dall'utilizzo della mappa e della navigazione del robot per individuare e sopprimere, tra tutti i rilevamenti effettuati nei singoli frame, i falsi positivi.

1.1 Struttura della tesi

Nel capitolo "Stato dell'arte" è presentata una breve panoramica del lavoro esistente in termini di rilevamento di persone a terra e di robot per l'assistenza di anziani.

In "Descrizione del sistema" viene descritto il robot utilizzato, sia per quanto riguarda l'hardware che per quanto concerne il software che lo controlla.

"Mappatura e navigazione" descrive come è stata migliorata la capacità del robot di mappare ambienti interni e navigare al loro interno.

Nel capitolo "Rilevamento di persone a terra" sono descritte le tecniche sperimentate per l'individuazione frame per frame di eventuali persone cadute.

In "Localizzazione delle persone a terra" è presentato il metodo ideato per segnalare la posizione delle persone a terra individuate, il quale al contempo migliora ulteriormente la capacità del robot di riconoscimento delle emergenze.

Infine in "Conclusioni" sono ricapitolati i risultati raggiunti da questo lavoro e sono fornite alcune indicazioni su possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

In questa sezione verrà presentata una rapida panoramica delle soluzioni esistenti per l'assistenza agli anziani. Sarà inoltre fornita una breve introduzione per quanto riguarda lo stato dell'arte di mappatura e navigazione.

Nella sezione "Tecniche di mappatura e navigazione" si fornirà una breve panoramica delle tecniche esistenti riguardo a mappatura e navigazione.

Nella sezione "Sistemi per il rilevamento di cadute" verranno descritte alcune tecniche non basate su robot per il rilevamento di cadute.

Nella sezione "Robot per Ambient Assisted Living", invece, ci si concentrerà sui robot e sugli algoritmi per questi ultimi dedicati al tema del rilevamento di persone a terra.

2.1 Tecniche di mappatura e navigazione

Una tematica di importanza centrale nel campo della robotica è sicuramente quella della navigazione, sia essa in mappe note [4, 5] o in ambienti ignoti [6], con presenza di ostacoli statici (la cui posizione non cambia nel tempo, ad esempio il mobilio di una stanza) o dinamici (la cui posizione varia, come nel caso di persone ed animali). Molti sforzi su questo frangente sono volti al miglioramento della robustezza e dell'affidabilità della navigazione [7]: il robot deve essere in grado di raggiungere i goal prefissati evitando di collidere con gli oggetti presenti nell'ambiente, di qualsiasi natura essi siano, idealmente in tutte le differenti condizioni ambientali in cui è pensato per operare.

Diverse tecniche di mappatura recenti sono basate su un metodo noto ed affermato da tempo per rappresentare l'informazione sull'ambiente circostante: la "occupancy grid" [8]. Secondo tale sistema, lo spazio, bidimensionale o tridimensionale che sia, viene diviso in celle di lato noto. In ogni cella è contenuto un valore che rappresenta la probabilità che questa sia occupata. Tali stime sono quindi aggiornate ad ogni nuovo input proveniente dai sensori del robot.

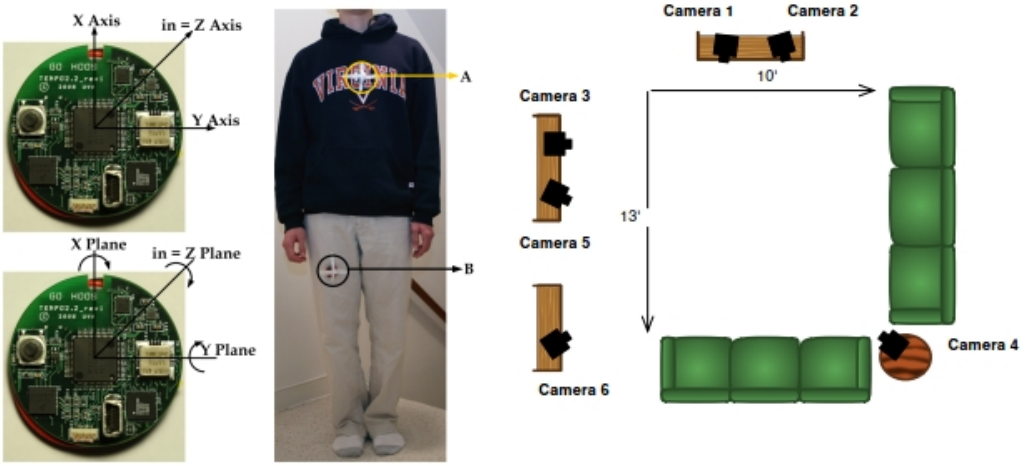
Per quanto riguarda la creazione di mappe in due dimensioni, una delle soluzioni più utilizzate è GMapping [38], appartenete al progetto OpenSLAM[1]. L'algoritmo alla base di GMapping implementa un Rao-Blackwellized Particle Filter [39, 40] per risolvere il problema detto SLAM ("Simultaneous Localization And Mapping" [41, 42]), permettendo di ricostruire una occupancy grid da dati provenienti da un sensore Laser Range Finder.

Una volta ottenuta la mappa dell'area, è fondamentale, ai fini della navigazione, che il robot sia in grado di localizzarsi efficacemente all'interno di essa. "Adaptive Monte Carlo Localization" (AMCL) [43] è un sistema di localizzazione probabilistico su mappe bidimensionali. Utilizza un particle filter per rappresentare tutti i possibili stati del robot e sfrutta i dati provenienti dai sensori, confrontandoli con la mappa nota, per stimare l'attuale posizione del robot.

Per quanto riguarda i sensori utilizzati per percepire l'ambiente circostante, l'incremento nelle capacità di calcolo e memorizzazione dei calcolatori sta favorendo sempre più l'adozione di sensori di tipo RGB-D [9]; questi ultimi sono utilizzati tanto per ricostruire una mappa dell'area circostante, quanto per ottenere informazioni su posizione del robot e ostacoli non mappati in fase di navigazione autonoma. Soluzioni alternative per la localizzazione di robot all'interno di edifici ricorrono all'uso di tag RFID [10] o tag visivi [11].

[1] Link: `https://openslam.org`

2.2 Sistemi per il rilevamento di cadute



(a) Sensore indossabile TEMPO 3.0, basato su accelerometro e giroscopio. (Fonte: [12])

(b) Disposizione delle telecamere per il rilevamento di cadute in una stanza. (Fonte: [13])

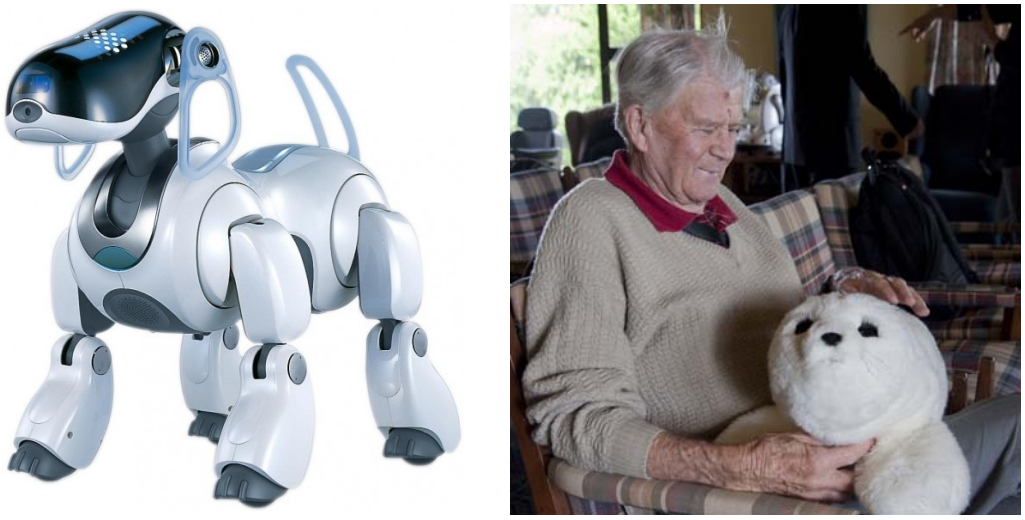
Figura 2.1: Esempi di sistemi di rilevamento di cadute.

In letteraura è possibile trovare una gran varietà di proposte per sistemi di Ambient-Assisted Living, a seconda delle necessità di spazio, privacy, efficacia ed economia che ci si impone [14]. Per quanto riguarda il problema della rilevazione di cadute, sono stati presentati numerosi possibili approcci, a seconda del tipo di informazioni e dispositivi che si intende utilizzare.

Una categoria che recentemente ha conosciuto una grande crescita è quella dei dispositivi indossabili. Parte della popolarità di queste soluzioni è dovuta alla diffusione di piattaforme open-source sempre più performanti e ridotte nelle dimensioni, interfacciabili con una vasta gamma di sensori a basso costo [15]. Nella maggior parte dei casi questi ultimi sono accelerometri [12, 16, 17], eventualmente combinati con altri tipi di sensori. La principale criticità consiste nel rendere gli algoritmi di apprendimento intelligente capaci di distinguere cadute da altre azioni ricorrenti, come il sedersi o il distendersi a letto. L'esempio di un dispositivo di tal sorta è riportato in Figura 2.1a. Una delle principali critiche mosse a questo tipo di periferiche riguarda il fatto che non tutte le persone sono disposte ad accettare di indossarle; è peraltro possibile che talvolta non siano indossate anche solo a causa di una dimenticanza.

Altre tecniche per la rilevazione in tempo reale di cadute si affidano a dispositivi montati nell'ambiente, come array di microfoni [18] e telecamere che operano il tracking delle persone [13, 19], o alla combinazione di PIR e altri sensori economici che rilevano vibrazioni [20]. Queste ultime tuttavia presentano generalmente prestazioni inferiori rispetto al grado di affidabilità ottenuto con dispositivi indossabili. La scelta di utilizzare telecamere fisse è generalmente dispendiosa, dal momento che per ottenere una copertura di tutta la casa è necessario installare diversi dispositivi (si veda, ad esempio, Figura 2.1b) e prevedere un calcolatore all'altezza per gestirne la grande mole di dati. Molte persone, inoltre, sono diffidenti nei confronti delle telecamere perché le vedono come un'invasione nella loro privacy.

2.3 Robot per Ambient Assisted Living



(a) (Fonte: `http://www.sony-aibo.com`) (b) (Fonte: `http://wiki.auckland.ac.nz`)

Figura 2.2: Esempi di robot progettati per suscitare una risposta affettiva: Aibo (sinistra) e Paro (destra).

I robot per assistenza sociale hanno il compito di interagire con gli utenti al fine di migliorarne la qualità di vita [21]. In particolare, i settori in cui tali robot sono maggiormente utili riguardano l'assistenza di persone anziane, il supporto alla riabilitazione di persone colpite da un trauma fisico e il trattamento terapeutico di persone con disabilità cognitive. I robot progettati per AAL possono essere intesi per agire su due frangenti: quello affettivo e quello funzionale [22].

Per quanto riguarda l'ambito affettivo, si va a ricercare un robot che vada a suscitare una risposta emozionale nelle persone con cui interagisce. Un esempio di robot di questo tipo è Aibo [23], un robot dalle sembianze canine (ritratto in Figura 2.2a) prodotto da Sony. Tale dispositivo, prima di uscire di produzione, è stato utilizzato in diversi studi per stimare gli effetti che l'interazione del robot con gli anziani ha sulla loro qualità di vita e sulla gestione dello stress. Un altro robot oggetto di simili studi è Paro [24], sviluppato dall'istituito giapponese ISRI, avente le sembianze di una foca peluche (visibile in Figura 2.2b).
I robot progettati per fornire dei servizi hanno invece un design maggiormente dettato dalle funzioni che devono svolgere. Di questa categoria, Pearl [25] (Figura 2.3a), ideato alla Carnegie-Mellon e parte del progetto "Nursebot", è uno tra i più studiati e citati. I compiti per i quali è stato inteso riguardano l'agevolazione degli spostamenti degli anziani nelle case di riposo e l'assistenza nello svolgimento di attività di routine. Astro [26] (visibile in Figura 2.3b), realizzato dalla Scuola Superiore Sant'Anna di Pisa, è un robot sviluppato per svolgere un'ampia varietà di mansioni, che presenta caratteristiche e tecnologie compatibili con la produzione in serie su larga scala. Un simile progetto, con base in Austria, riguarda il robot di servizio Hobbit [27]. Interessante è anche il progetto Giraffplus [28], che prevede un sistema integrato comprendente, oltre a numerosi sensori domotici, anche un robot di telepresenza (riportato in Figura 2.3c). Tale sistema è già stato estensivamente testato in casi d'uso reali.

Il campo della ricerca di algoritmi per l'identificazione di persone a terra mediante videocamere è un settore molto attivo. Il denominatore comune di tali studi riguarda l'ideazione di soluzioni performanti e allo stesso tempo compatibili con le limitate risorse computazionali a disposizione del robot. Wang, Zabir e Leibe [29] ad esempio propongono

(a) (Fonte: `http://www.cs.cmu.edu`)

(b) (Fonte: `http://www.robot-era.eu`)

(c) (Fonte: `http://www.mdpi.com`)

Figura 2.3: Esempi di robot di servizio: da sinistra, Pearl, Astro e Giraff.

un efficace algoritmo a pipeline, inteso per essere usato su robot di servizio dotati di sensori RGB. Molto simile al caso di studio di questa tesi è invece la proposta di Volkhardt, Schneemann e Gross [30], che sfrutta dati tridimensionali ottenuti da un sensore Kinect per rilevare persone a terra mediante un'analisi su più livelli. Il principale vantaggio di questo approccio consiste nella possibilità di far funzionare il robot anche in condizioni di scarsa o nulla illuminazione, cosa non praticabile nel caso dell'utilizzo di dati RGB. Inoltre gli algoritmi pensati per funzionare su dati bidimensionali sono generalmente meno robusti in scenari dove sono presenti molti ostacoli.

Infine, una volta rilevata la persona a terra, può essere opportuno che il robot conduca ulteriori operazioni per accertarsi del tipo di emergenza in corso e fornire ulteriori dettagli per un soccorso più pronto. Nishi e Miura [31] per tali scopi suggeriscono un algoritmo per individuare la posizione della testa di una persona distesa, basato sull'uso di un sensore RGB-D. Capire la posizione del capo è infatti importante per permettere al robot di testare alcuni segni vitali dell'individuo, come il colorito facciale o la concentrazione di diossido di carbonio nel fiato.

Capitolo 3

Descrizione del sistema

La scelta delle componenti meccaniche e la progettazione del software di un robot sono fortemente dipendenti dalle funzioni che deve svolgere e dall'ambiente nel quale si vorrà utilizzarlo.

Per quanto riguarda l'hardware, la scelta di sensori più adeguata per lo scenario di utilizzo del robot è fondamentale [32]; la maggiore convenienza di un sensore rispetto all'altro è dettata dalle caratteristiche fisiche e dalle capacità computazionali del robot stesso, oltre che dallo studio di come esso può interagire con l'ambiente in cui si muove.

Per quanto riguarda il software, si va a scegliere il modo in cui rappresentare, elaborare ed integrare le informazioni provenienti dal mondo e percepite dai sensori, al fine di ottenere il miglior risultato possibile con le componenti a disposizione.

Il lavoro esposto in queste pagine è incentrato su un robot dotato di un sensore Laser Range Finder (LRF) ed un sensore RGB-D. I compiti per i quali è stato progettato prevedono la navigazione autonoma in ambienti interni, il rilevamento di persone a terra, il riconoscimento di esseri umani, l'interazione con essi e la telepresenza [33].

Di seguito, in "Robot" verranno forniti maggiori dettagli di carattere tecnico sul robot utilizzato e sui sensori principali sui quali fa affidamento.

In "Software" verranno invece presentate le principali caratteristiche del software impiegato per permettere il funzionamento del robot.

3.1 Robot

Il robot consiste principalmente in un Turtlebot 2[1] con base Kobuki della iClebo, illustrato in Figura 3.3a, al quale sono state apportate alcune modifiche ed aggiunte, visibili in Figura 3.3b oltre che nel modello rappresentato in Figura 3.1c e Figura 3.1d:

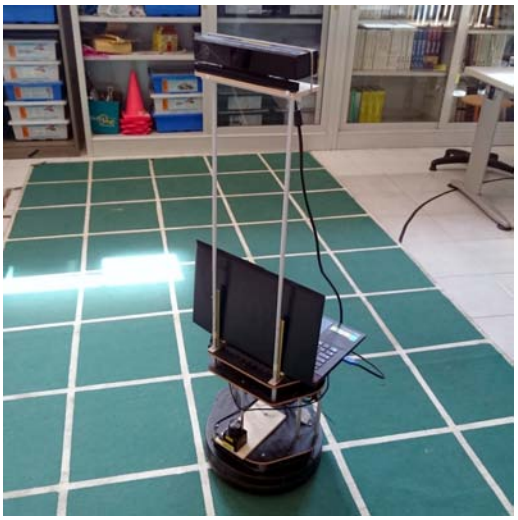
- Una struttura metallica soprastante il robot, la quale porta l'altezza massima dell'intero dispositivo a 1.18 m.
- Un sensore Microsoft Kinect One (Kinect v2) montato in cima alla struttura metallica sopra descritta. Tale dispositivo ha, per quanto riguarda il sensore a colori, una risoluzione di 1920×1080 pixel ed un FoV pari a $84.1° \times 53.8°$, mentre, per quanto riguarda il sensore ad infrarossi con tecnologia ToF, una risoluzione di 512×424 pixel con un FoV pari a $70.6° \times 60°$ e range 0.5 m - 4.5 m; entrambi i dati sono aggiornati con una frequenza massima di 30 Hz.
- Un sensore LRF Hokuyo URG-04LX-UG01 montato su un supporto soprastante la base, in posizione frontale (sopra il bumper). Tale posizione fa sì che il piano di acquisizione dei dati laser sia parallelo al suolo, dal quale dista 21 cm.

  Il sensore ha risoluzione angolare (angolo minimo tra due raggi contigui, o "step angle") pari a 0.36°, FoV 240°, range 2cm - 5.6m con accuratezza di ± 3 cm entro 1 m di distanza e frequenza di aggiornamento 10 Hz.

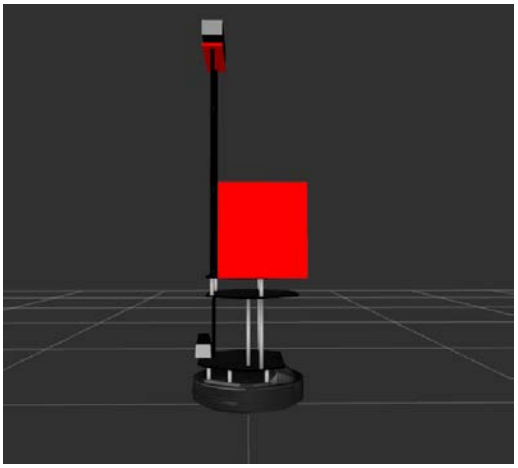
Per la parte di elaborazione il robot si affida ad un computer portatile Lenovo Y50-70 appoggiato sulla superficie superiore della struttura, avente le seguenti specifiche:

[1] Per maggiori informazioni: `http://www.turtlebot.com/`

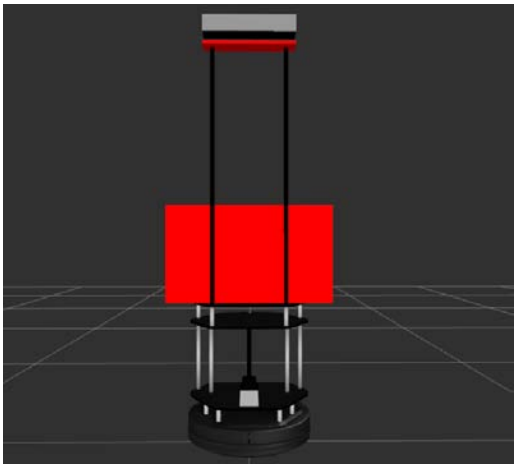
(a) Turtlebot 2 originale.



(b) Foto del robot in dotazione.



(c) Modello URDF utilizzato per rappresentare il robot, vista laterale.



(d) Modello URDF utilizzato per rappresentare il robot, vista frontale.

Figura 3.1: Il robot.

- Processore: Intel Core i7-4710HQ
- RAM: 16 GB DDR3L
- GPU: NVIDIA GeForce GTX 860M con 4 GB di memoria dedicata
- Spazio di archiviazione: Samsung SSD 840 EVO 250 GB
- Sistema operativo: Ubuntu 14.04 64 bit

Il progetto del quale fa parte questa tesi è finanziato da Omitech[2], che ha fornito robot e computer.

[2]Link: http://www.omitech.it

3.2 Software

Di seguito saranno descritti brevemente le caratteristiche delle principali librerie software utilizzate. Saranno quindi analizzati nello specifico alcuni pacchetti di importanza centrale nel corso di questa tesi.

3.2.1 Il meta-sistema operativo ROS

Figura 3.2: Logo di ROS. (Fonte: `http://www.ros.org`)

"Robot Operating System" (ROS)[3] [34] è un set di librerie software e strumenti pensati per agevolare la creazione di applicazioni per la robotica. È un progetto open-source, supportato dalla "Open Source Robotics Foundation", che può contare su una comunità di utenti molto attiva. Il software ROS è strutturato per essere distribuito e modulare, e può contare su più di 3000 pacchetti. I componenti fondamentali di ROS (detti "core") sono distribuiti sotto licenza BSD, che ne permette l'utilizzo anche in prodotti di natura commerciale. I pacchetti della comunità tuttavia possono essere regolati da un'ampia gamma di licenze, tra le quali Apache 2.0, GPL, MIT o licenze proprietarie.

Il core di ROS è in sostanza un middleware, costituito da un elemento di coordinazione ("ROS Master") presso il quale si registrano i vari moduli attivi, detti "nodi". I nodi possono essere dedicati alla gestione di particolari componenti robotici o semplicemente pensati per assolvere a funzioni specifiche. La comunicazione diretta tra i vari nodi è regolata mediante l'uso di messaggi standard. In ROS sono stati specificati differenti tipi di messaggi standard, per coprire ogni sorta di informazione legata alla gestione del robot: letture dei sensori, informazioni di carattere geometrico e dati di navigazione e posizionamento.

ROS può inoltre contare su una buona base di strumenti pensati per la diagnosi e il debug del codice sviluppato. Tra questi vi è il visualizzatore RViz, che tramite un'interfaccia grafica permette di visualizzare dati provenienti dai sensori, sistemi di riferimento e modelli tridimensionali.

Infine il software ROS offre l'integrazione nativa di altre librerie software connesse al mondo della robotica; tra queste si possono annoverare il software di simulazione Gazebo[4], la libreria di motion planning MoveIt![5] e le librerie di computer vision OpenCV e PCL (descritte nella sezione che segue).

La versione di ROS utilizzata in questo progetto di tesi è "Indigo Igloo".

3.2.2 Librerie per computer vision

Nel corso di questa tesi sono stati utilizzati metodi e tecniche provenienti da due affermate librerie di computer vision distribuite con licenza BSD: PCL e OpenCV.

[3] Link: `http://www.ros.org`
[4] Link: `http://gazebosim.org/`
[5] Link: `http://moveit.ros.org/`

(a) (Fonte: `http://pointclouds.org`) (b) (Fonte: `http://opencv.org`)

Figura 3.3: Logo di PCL (a sinistra) e di OpenCV (a destra).

"Point Cloud Library" (PCL)[6] [35] è un progetto open-source volto a fornire gli strumenti necessari ad elaborare nuvole di punti tridimensionali. Include l'implementazione di numerosi algoritmi allo stato dell'arte per compiere un'ampia gamma di operazioni, tra cui filtraggio, calcolo di descrittori, ricostruzione di superfici, registrazione di nuvole di punti e segmentazione. Offre inoltre l'accesso a tecniche di alto livello per la mappatura di ambienti e il riconoscimento di oggetti.

In questo lavoro di tesi sono utilizzate PCL 1.7 e PCL 1.8.

"Open Source Computer Vision" (OpenCV)[7] [36] è una libreria open-source pensata per offrire un'infrastruttura comune per applicazioni di computer vision. Il software in essa disponibile è ottimizzato per sfruttare piattaforme multi-core e l'accelerazione hardware quando possibile, ed è pensato principalmente per l'uso in tempo reale. I più di 2500 algoritmi che ne fanno parte comprendono sia tecniche allo stato dell'arte che appartenenti alla letteratura classica. Vi è inoltre una nutrita sezione di algoritmi di machine learning per la computer vision. Utilizzi tipici del software di OpenCV includono rilevazione e riconoscimento di volti, individuazione di oggetti, classificazione di azioni umane nei video, eye-tracking e creazione di nuvole di punti tridimensionali usando fotocamere stereo.

In questo lavoro di tesi sarà utilizzato OpenCV 2.4.

3.2.3 La repository `orobot`

Tutto il software finora sviluppato per il robot in uso è raccolto in una repository open-source denominata `orobot`[8]. Di seguito saranno descritti con maggior dettaglio i pacchetti ROS appartenenti ad essa dedicati ai compiti di mappatura e navigazione.

Mappatura

Il compito di mappare un ambiente è eseguito sotto il comando di un agente umano, il quale si occupa di fare navigare il robot mediante un joypad. L'obiettivo è quello di ottenere una mappa il più possibile completa e corretta della zona nella quale il robot dovrà operare. In questa fase, quindi, parte della qualità del risultato finale è anche dipendente dalla perizia con cui l'operatore al comando del robot si preoccupa di coprire tutta l'area di interesse. Eventuali zone ricostruite male possono essere individuate, in tempo reale, sullo schermo del portatile; passandovi a più riprese con il robot è possibile cercare di migliorarne la qualità.

[6] Link: `http://pointclouds.org`
[7] Link: `http://opencv.org`
[8] Reperibile al link: `https:/bitbucket.org/iaslab-unipd/orobot`

Il pacchetto che permette di avviare l'operazione di mappatura è `orobot_mapping`[9]. Tale pacchetto è costituito solamente da un launcher, il cui compito è quello di invocare, a sua volta, altri quattro launcher:

- **orobot_bringup/minimal.launch:** si occupa dell'inizializzazione di tutti i servizi "base" del robot, tra cui la pubblicazione delle informazioni sulle trasformate[10] tra i vari frames (topic `/tf` e `/tf_static`), il controllo del movimento e della posizione (topic `/odom`, `/cmd_vel_mux`) ed in genere tutte le informazioni provenienti dai pulsanti ed i sensori interni della base.

  Inoltre, un importante compito svolto da questo launcher è quello di avviare anche il driver del sensore laser (mediante l'avvio del nodo `urg_node` del pacchetto `urg_node`) ed effettuare la prima operazione di filtraggio dei dati "grezzi" (utilizzando un nodo `scan_to_scan_filter_chain` del pacchetto `laser_filters`) con un filtro interpolatore; gli scan così processati vengono ripubblicati nel topic `/laser_scan`.

- **orobot_navigation/gmapping_demo.launch:** si occupa della vera e propria mappatura dell'ambiente circostante. Il pacchetto `gmapping` di ROS è di fatto un wrapper dell'omonima libreria del progetto OpenSLAM. Come precedentemente spiegato, l'algoritmo alla base di GMapping permette di ricostruire una grid map da dati provenienti da un sensore LRF. Di conseguenza, il nodo ROS si sottoscrive ai topic nei quali sono pubblicate le trasformate ed al topic dove sono pubblicate le informazioni laser, mentre pubblica nel topic `/map` la mappa finora elaborata.

- **orobot_teleop/logitech.launch:** permette il controllo del robot tramite un joypad wireless Logitech.

- **orobot_rviz_launchers/view_navigation.launch:** apre il visualizzatore RViz permettendo di vedere in tempo reale l'andamento della creazione della mappa sullo schermo del PC.

Una volta raggiunto l'obiettivo preposto, la mappa corrente può essere salvata in un file .pgm avviando il nodo `map_saver` del pacchetto `map_server`, il quale crea anche un file .yaml contenente i principali parametri che la descrivono.

Navigazione

Per quanto riguarda la navigazione, il robot si può muovere all'interno di un ambiente, del quale è nota la mappa, in modo semi-autonomo o totalmente autonomo. Nel primo caso il tragitto viene pianificato per raggiungere una serie di tappe specificate a priori, in RViz, da un operatore umano. Nel secondo caso il robot segue un percorso casuale.

Le due possibili modalità corrispondono rispettivamente ai launcher `manual.launch` ed `automatic.launch` del pacchetto `orobot_coverage_launchers`. Ai fini dell'attuale esperienza è stata utilizzata, in fase di test, la prima modalità.
Come nel caso precedente, il launcher utilizzato rimanda alla chiamata di altri launcher, tra cui i già discussi **minimal.launch**, **logitech.launch** e **view_navigation.launch**. Il compito vero e proprio di gestire navigazione e localizzazione del robot è invece affidato a:

- **orobot_navigation/amcl_demo.launch:** avvia tutte le funzionalità fondamentali per attuare la navigazione, ossia i nodi `map_server`, `amcl` e `move_base`.

[9] Mediante il comando `roslaunch orobot_mapping make_map.launch`

[10] Per maggiori informazioni riguardo al modo in cui ROS gestisce i frame di riferimento e le rispettive trasformate, si veda [37]

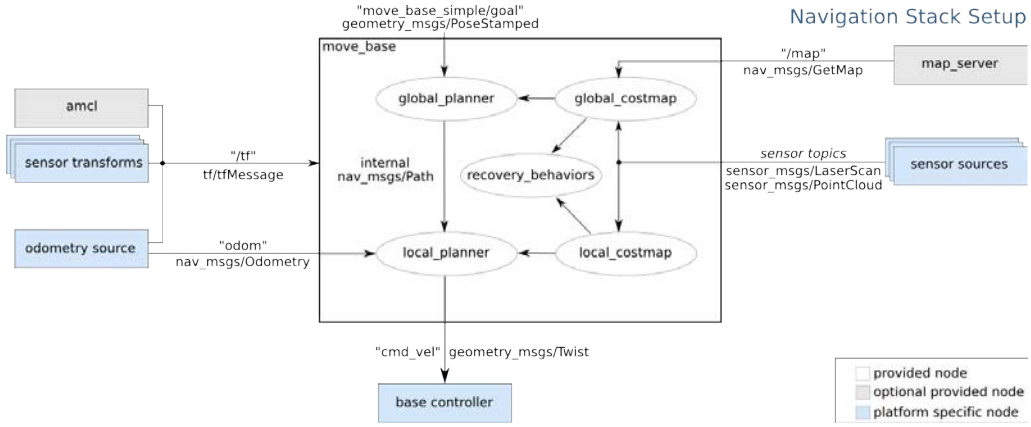


Figura 3.4: Schema riassuntivo del funzionamento della navigazione in ROS. (Fonte: `http://wiki.ros.org/move_base`)

`map_server` si occupa di rendere disponibile una mappa precedentemente salvata, pubblicandola nel topic `/map`.

`amcl` si occupa di gestire la localizzazione del robot nella mappa. L'implementazione disponibile in ROS è predisposta per accettare in input solo dati di tipo `sensor_msgs/LaserScan`, tipicamente ottenuti da sensori LRF.

I topic che richiede in ingresso sono pertanto le trasformate, i dati del laser, e la mappa. È richiesta inoltre la posa iniziale (approssimativa) del robot, pubblicata in `/initialpose`; questa è fornita ogni volta dall'operatore, il quale, dopo l'avvio di `manual.launch`, deve specificare posizione ed orientamento del robot nella mappa, facendo uso dello strumento apposito presente in RViz. Alternativamente, si può impostare una posa prestabilita dalla quale il robot verrà avviato (in un caso d'uso reale, questa potrebbe corrispondere alla locazione in cui è situata la base di ricarica).

`move_base` svolge la funzione del planner vero e proprio, oltre che quella di gestore del movimento del robot. Secondo i dettami del paradigma ROS per la gestione del navigation stack (schematizzati in Figura 3.4), questo nodo accetta in input le informazioni sulla mappa, il goal da raggiungere ed i dati sull'attuale localizzazione del robot; pubblica quindi direttamente sul topic `cmd_vel` i comandi per dirigere il robot lungo il percorso calcolato. La gestione della pianificazione è organizzata su due livelli, uno globale ("global plan") ed uno locale ("local plan") [5].

Il plan globale si occupa di trovare il percorso migliore dalla posizione attuale all'obiettivo sulla mappa nota, utilizzando l'algoritmo A* [44]. Il planner locale ha un'area d'azione limitata alle vicinanze del robot (agisce su un'area quadrata di lato 4 metri centrata sul turtlebot) e considera anche ostacoli dinamici non presenti nella mappa preimpostata ai fini del calcolo della traiettoria da seguire. La valutazione degli spostamenti da fare viene svolta servendosi di una "costmap" in entrambi i casi.

- **orobot_manual_path_planner/plan.launch:** si sottoscrive al topic di RViz `/clicked_point`, dove sono pubblicate le tappe di navigazione che l'utente ha impostato da interfaccia grafica. Il suo compito è quello di renderle disponibili, nel formato richiesto dagli altri nodi, tramite il servizio `/goals_array`, oltre a pubblicare in `/manual_path_marker` il percorso calcolato dal planner in un formato che rende possibile visualizzarlo con RViz.

L'avvio della navigazione, una volta specificati posa iniziale e obiettivi, avviene invocando il comando `rosrun simple_navigation_goal mod_simple_navigation_goal`.

Il nodo così avviato si sottoscrive a `goals_array` ed invia una dopo l'altra le tappe di navigazione al nodo `move_base`. Oltre a ciò, verifica il successo od il fallimento del tentativo di trovare un percorso verso la posizione desiderata; al raggiungimento di un dato numero di tentativi con esito negativo, mette in atto una procedura di recovery molto basilare, consistente nel fare indietreggiare il robot di una piccola distanza prefissata. Quest'ultima procedura permette di sbloccare il robot da una frequente situazione di stallo causata dalle discrepanze tra modello del robot e robot reale.

3.2.4 Il pacchetto `kinect2_bridge`

Oltre a navigazione e mappatura, il robot dispone anche di un pacchetto dedicato all'interazione con la telecamera Kinect v2, `kinect2_bridge`, appartenente alla repository di ROS `iai_kinect2`.

Avviando il launcher `kinect2_bridge_ir.launch` viene infatti avviato il driver e sono pubblicati tutti i topic contenenti i dati messi a disposizione dal dispositivo.

Di particolare interesse sono i topic `/kinect2_head/depth_ir/points`, dove è possibile ottenere la point cloud elaborata a partire dalle rilevazioni del sensore ToF infrarosso, e `/kinect2_head/depth/image`, dalla quale si può ottenere la depth-image associata alla visuale corrente.

3.2.5 Il pacchetto `sensor_msgs`

Il pacchetto `sensor_msgs` di ROS definisce i formati di messaggio standard per i sensori più comunemente usati. Di seguito sono descritte le caratteristiche dei principali tipi di messaggio con i quali si è avuto a che fare durante l'esperienza.

`LaserScan` è il formato usato per comunicare dati provenienti da sensori LRF. Ogni messaggio rappresenta una singola acquisizione (o più semplicemente "scan"), ed è costituito dai campi:

- `Header header`, header standard ROS, comprende un timestamp (che nel caso specifico è relativo all'istante di acquisizione del primo raggio), l'indicazione del frame di riferimento (nel nostro caso, *base_laser_link*) ed un sequence ID.
- `float32 angle_min`, angolo in radianti da cui è iniziata la scansione. Per convenzione, l'angolo di scansione è misurato attorno all'asse z, perpendicolare al piano di scansione e rivolto verso l'alto, in senso antiorario. La direzione corrispondente all'angolo nullo coincide con l'asse x [11].
- `float32 angle_max`, angolo in radianti in corrispondenza del quale termina la scansione.
- `float32 angle_increment`, step angle.
- `float32 time_increment`, tempo in secondi tra due misure consecutive .
- `float32 scan_time`, tempo in secondi tra due scan consecutivi.
- `float32 range_min`, valore minimo del range.
- `float32 range_max`, valore massimo del range.

[11] Per una descrizione più approfondita della convenzione comunemente utilizzata per la nomenclatura degli assi di riferimento in ROS si rimanda alla sezione "Implementazione".

- `float32[] ranges`, vettore contenente i ranges relativi ad ogni raggio. Tale array avrà n celle, ognuna contenente la distanza (range) misurata lungo la direzione di angolo α, con:

$$n = \frac{angle\_max - angle\_min}{angle\_increment} \tag{3.2.1}$$

$$\alpha = angle\_min + (cell\_index * angle\_increment) \tag{3.2.2}$$

- `float32[] intensities`, vettore contenente l'intensità misurata relativa ad ogni raggio.

`PointCloud2` è il formato standard dei messaggi contenenti point cloud; è utilizzato per i messaggi pubblicati nel topic `/kinect2_head/depth_ir/points`. Esso è costituito dai seguenti campi:

- `Header header`, header standard ROS. Il frame id dei punti pubblicati da `/kinect2_head/depth_ir/points` è *kinect2_head_ir_optical_frame*.
- `uint32 height`, una delle due dimensioni della point cloud, nel caso questa sia "2D-organized". Nel caso invece in cui la nuvola di punti sia "unordered", questo campo ha valore 1.
- `uint32 width`, una delle due dimensioni della point cloud. Il numero totale dei punti della nuvola è ricavabile come `height` $*$ `width`.
- `PointField[] fields`, descrive i canali usati per rappresentare i dati contenuti nel messaggio ed i rispettivi layout.
- `bool is_bigendian`, specifica se è usata la convenzione "big endian".
- `uint32 point_step`, dimensione in byte di un punto.
- `uint32 row_step`, dimensione in byte di una riga.
- `uint8[] data`, i veri e propri dati point cloud. Occupano uno spazio pari a `row_step*height` byte.
- `bool is_dense`, specifica se ci sono punti non validi.

Capitolo 4

Mappatura e navigazione

L'obiettivo che ci si propone in questo capitolo è quello di individuare ed implementare un modo per integrare le informazioni provenienti dai dati laser con quelle provenienti dal sensore Kinect. Ciò contribuirà a rendere più dettagliata la mappa e più robusta la navigazione, specialmente nel caso di ostacoli la cui forma articolata impedirebbe al solo laser di svolgere il compito in maniera affidabile.

Nella sezione "Descrizione del problema" è introdotto con maggiore dettaglio il problema che ci si propone di risolvere.

Nella sezione "Implementazione" sono discussi i metodi utilizzati e le soluzioni ideate per raggiungere l'obiettivo.

Nella sezione "Risultati sperimentali" infine si andranno ad illustrare brevemente alcuni test eseguiti per testimoniare i progressi fatti.

4.1 Descrizione del problema

Nel capitolo precedente si è sottolineato come pacchetti di importanza centrale per i compiti di mappatura e navigazione, ovvero `gmapping` e `amcl`, siano predisposti per funzionare con messaggi `sensor_msgs/LaserScan`, ottenuti dal sensore LRF montato sul robot.

L'utilizzo di tale dispositivo permette di ottenere informazioni di tipo esclusivamente bidimensionale sugli ostacoli nelle vicinanze, in quanto restituisce tutti e soli i punti in cui un ben precisato piano interseca gli oggetti all'interno del FoV del sensore. Tipicamente, per ovviare a tale problema, si può ricorrere ad espedienti esterni, come l'introduzione di attuatori che permettano di variare nel tempo direzione ed inclinazione del fascio laser. Nel caso in esame, invece, si è già visto che il piano di acquisizione degli scan è fisso, parallelo al suolo, situato a 21 cm d'altezza.

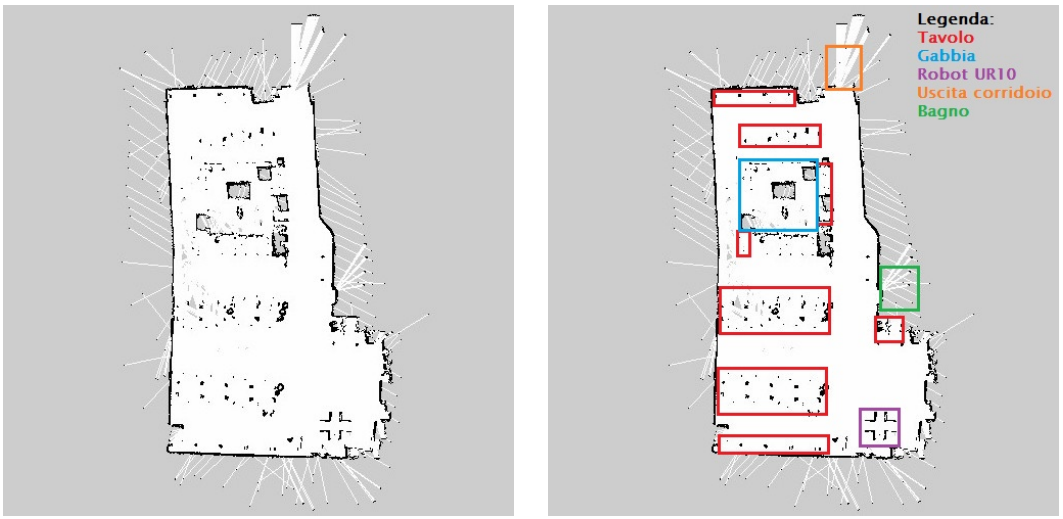
Il problema che ci si propone di risolvere è originato dal fatto che gli ingombri verticali del robot raggiungono una quota quasi un metro al di sopra del fascio laser. La realtà bidimensionale ricostruita dal sensore, infatti, si rivela assolutamente insufficiente a garantire il corretto funzionamento del robot in un ambiente indoor senza l'introduzione di pesanti vincoli sugli ostacoli presenti in tale scenario.

Oggetti ampiamente diffusi come i tavoli costituiscono un esempio lampante di quanto appena affermato.

I tavoli sono caratterizzati da una parte piana, posta ad un'altezza superiore a quella del sensore laser ma inferiore al punto più alto del robot. Questa poggia su piedistalli dalla sezione orizzontale tipicamente modesta e, soprattutto, distanziati tra loro di una misura generalmente superiore al massimo ingombro orizzontale del robot. È facilmente intuibile quindi che, data una mappa come quella di Figura 4.1, il planner possa restituire una traiettoria per il percorso verso il goal passante tra le gambe dei tavoli; avviata la navigazione, la collisione della parte superiore del robot con il bordo del tavolo è inevitabile.

Ovviamente i tavoli sono solo uno dei casi di collisione con oggetti mal rappresentati rilevati in fase di sperimentazione, tra cui vi sono anche le sedie girevoli (delle quali viene visto solo il piedistallo centrale) o la gabbia rettangolare segnalata in azzurro nella mappa di Figura 4.1.

Si è quindi deciso di sfruttare anche il sensore Kinect montato sul robot per ovviare a tali problemi. Le due principali strade percorribili in tal senso sono:



(a) (b)

Figura 4.1: Esempio di mappa di un ambiente indoor con tavoli ottenuta utilizzando il solo sensore LRF. Si noti come i tavoli (individuati dai rettangoli rossi) siano visti come sequenze di piccoli ostacoli circolari.

- passare ad una rappresentazione tridimensionale della mappa, ricostruibile grazie ai dati tridimensionali ottenuti dalla Kinect v2 ed all'utilizzo di appositi pacchetti disponibili per ROS, quali `octomap` per la mappatura e `3d_navigation` per la navigazione;

- mantenere la struttura esistente trovando un modo per integrare nei dati di tipo `sensor_msgs/LaserScan` forniti in ingresso ai nodi di mappatura e navigazione le informazioni sugli ostacoli circostanti ricavate grazie alla Kinect v2.

Si è optato per la seconda strada, la quale richiede l'introduzione di modifiche meno radicali ed invasive nella gestione del robot.

4.2 Implementazione

In questa sezione si andranno ad esaminare i principali metodi utilizzati per risolvere i problemi incontrati, fino a giungere alla soluzione finale, schematizzata in Figura 4.2.

Ogni sottosezione affronta un preciso aspetto del progetto, ne segue gli sviluppi e motiva le scelte fatte lungo tutto il corso dell'esperienza.

In particolare, "Proiezione dei dati Kinect sul piano laser" esamina le possibilità offerte dalla repository ufficiale ROS per ottenere uno scan dai dati ricavabili da un sensore Kinect. "Trasformazione dei dati in un unico frame di riferimento" evidenzia alcune questioni legate alla gestione dei frame utilizzati per rappresentare i dati. "Filtraggio" esamina e motiva l'importanza del filtraggio dei dati provenienti dai sensori; verranno qui presentate le successive strategie adottate ai fini dell'ottenimento di un buon risultato per quanto riguarda sia la mappatura che la navigazione. Analogamente, "Fusione degli scan" mostrerà le tecniche sperimentate per effettuare l'integrazione tra i dati laser e le proiezioni ottenute dalla Kinect in modo efficace. Le sezioni "Mappatura" e "Navigazione", infine, trattano i problemi specifici incontrati nei rispettivi ambiti.

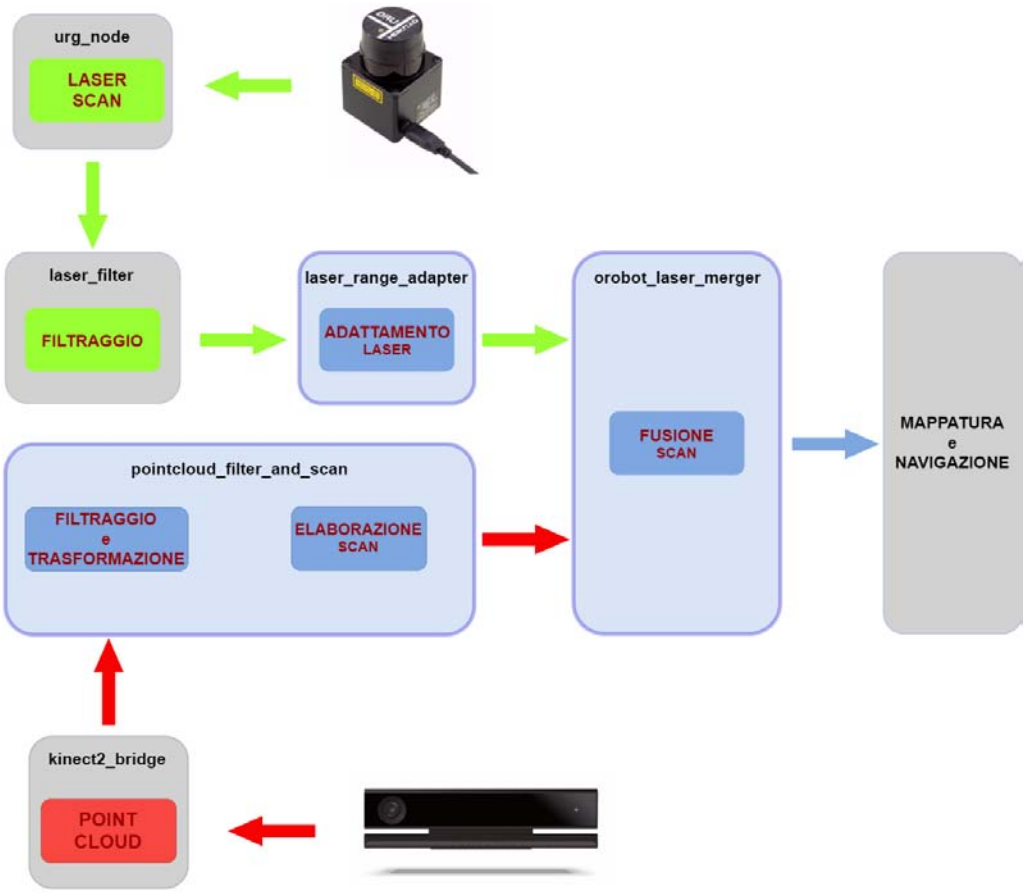


Figura 4.2: Schema riassuntivo delle funzioni svolte dai nodi sviluppati (in azzurro) e delle interazioni tra essi.

4.2.1 Proiezione dei dati Kinect sul piano laser

Il primo approccio seguito è stato quello di verificare se vi fossero pacchetti ROS preesistenti che permettessero, una volta combinati, di raggiungere il risultato desiderato. In particolare, per quanto riguarda l'ottenimento di uno scan laser significativo dai dati provenienti dalla Kinect, esistono due pacchetti (correntemente mantenuti) già nella repository ufficiale di ROS: `depthimage_to_laserscan` e `pointcloud_to_laserscan`. Come facilmente intuibile dai nomi, questi nodi sono studiati per sottoscriversi rispettivamente ai topic `/kinect2_head/depth/image` e `/kinect2_head/depth_ir/points` e pubblicare in output messaggi di tipo `sensor_msgs/LaserScan`. Il compito svolto da questi nodi è esattamente inteso per risolvere il problema in esame, in quanto il risultato ottenuto è assimilabile a ciò che si otterrebbe proiettando verticalmente sul piano di un sensore laser tutti gli oggetti inquadrati dalla telecamera.

Per ragioni di carico computazionale si è inizialmente preferito testare il funzionamento di `depthimage_to_laserscan`. Il risultato, riportato in Figura 4.3, evidenzia come il rumore presente nella depth-image andasse pesantemente a degradare la qualità dell'output. La quantità di rumore proiettata e rappresentata nel formato LaserScan si è rivelata talmente ingente da risultare un ostacolo pressoché insormontabile anche utilizzando i filtri messi a disposizione dalla community ufficiale ROS nel pacchetto `laser_filters`, pensati appositamente per filtrare dati provenienti da sensori laser. In questa fase si è anche rilevato che, generalmente, le operazioni di filtraggio più elaborate vengono effettuate trasformando i dati laser in una point cloud equivalente. In tal modo è possibile avere accesso ai numerosi e più potenti strumenti di filtraggio disponibili per queste ultime, dopo l'applicazione dei

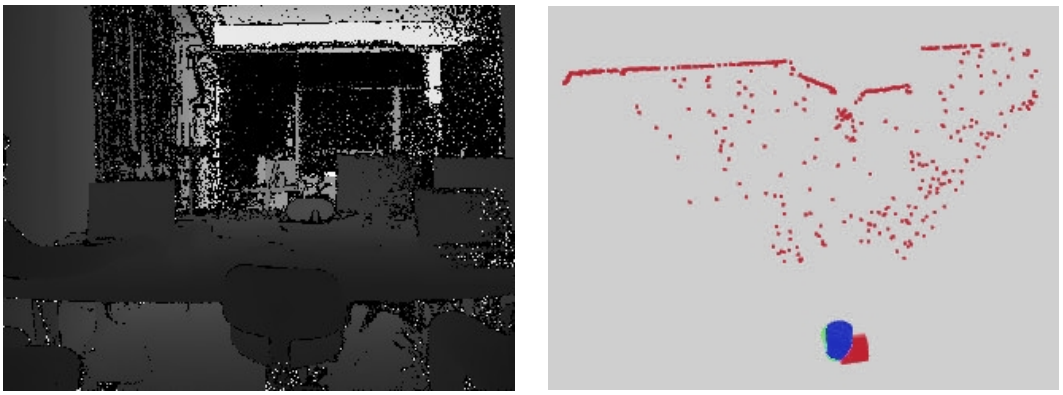


Figura 4.3: Una depth-image (sinistra) ed il rispettivo scan equivalente (destra).

quali è necessario ritrasformare il tutto nel formato iniziale. Tale strada è stata tuttavia considerata non conveniente, in quanto richiede due passaggi di trasformazione aggiuntivi che, di fatto, avrebbero annullato i vantaggi dell'uso della depth-image rispetto alla point cloud come fonte di informazione.

Essendo quindi necessario effettuare il filtraggio dei dati a monte del nodo `depthimage_to_laserscan`, si sono ricercati in letteratura e tra le repository di software di computer vision metodi per filtrare le depth-image. Dati i risultati inconsistenti delle ricerche fatte, si è preferito passare all'utilizzo di `pointcloud_to_laserscan`, consci del fatto che operare sulle point cloud avrebbe probabilmente avuto un impatto negativo sulle prestazioni.

Operare con le point cloud ha permesso di sfruttare le numerose tecniche di filtraggio disponibili nelle repository di software ROS (in particolare tali strumenti sono raccolti nel pacchetto `pcl_ros/Filters`) oppure reperibili direttamente dalle repository di PointCloud.org[1]. Come viene approfonditamente spiegato nella sezione "Filtraggio", un'opportuna combinazione di filtri di diversa natura ha permesso di ottenere in output da `pointcloud_to_laserscan` uno scan avente la qualità necessaria per ottenere mappe prive di rumore e complete.

Un'azione più radicale, invece, è stata operata in seguito ai test riguardanti la navigazione. La necessità di prestazioni maggiormente ottimizzate ha infatti spinto ad intervenire direttamente sul codice di `pointcloud_to_laserscan`.

Maggiori dettagli in merito a queste modifiche finali sono forniti nella sezione "Navigazione".

4.2.2 Trasformazione dei dati in un unico frame di riferimento

Fin dai primi test del pacchetto `pointcloud_to_laserscan`, è emerso un problema dato dal fatto che la Kinect possiede una convenzione, riguardante il frame rispetto al quale pubblica i dati, differente da quella del laser Hokuyo.

Come è possibile notare in Figura 4.4, infatti, i frame di riferimento per i dati laser e per le point cloud sono rispettivamente */base_laser_link* e */kinect2_head_ir_optical_frame*.

/base_laser_link ha l'asse x (verde) rivolto verso il senso di marcia, l'asse y (blu) parallelo al terreno e crescente verso la sinistra rispetto ad x e l'asse z (rosso) rivolto verso l'alto. */kinect2_head_ir_optical_frame*, invece, vede l'asse x rappresentare la larghezza, l'asse y rappresentare l'altezza ed infine l'asse z rappresentare la profondità.

`pointcloud_to_laserscan` è preimpostato per lavorare con dati in input riferiti a frame strutturati come */base_laser_link*. Fornendo in input point cloud riferite a */ki-*

[1]Link: `http://www.pointclouds.org/`

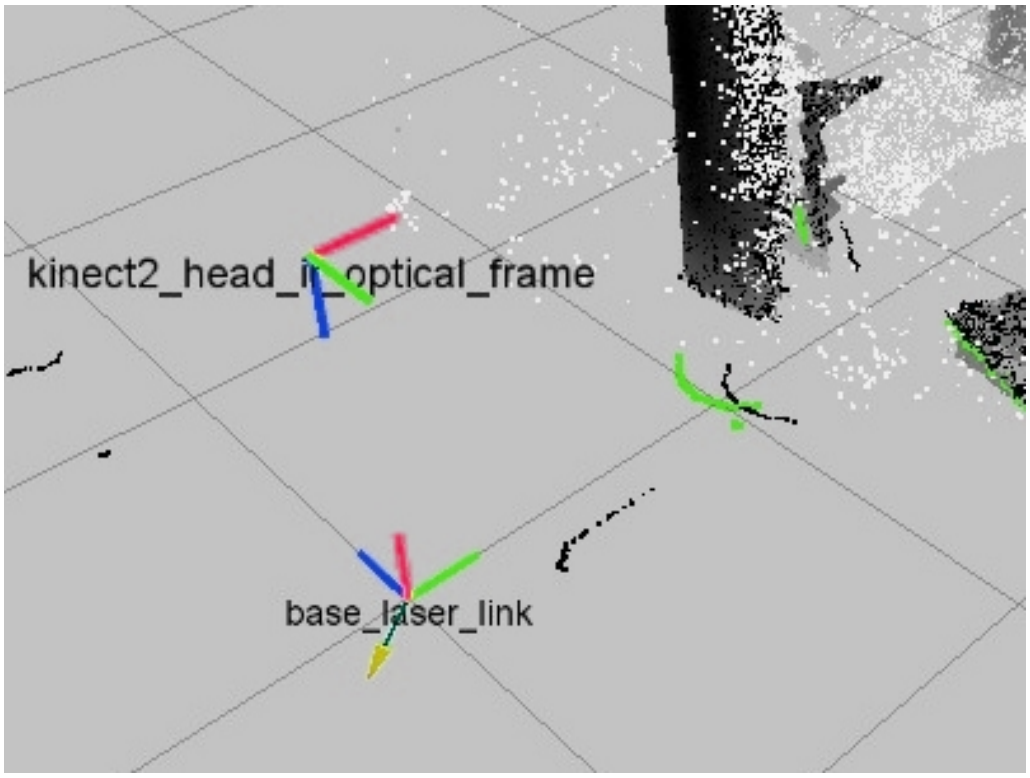


Figura 4.4: Esempio di schermata di RViz dove è possibile vedere chiaramente la differente convenzione utilizzata per il frame del sensore Kinect (in alto) ed il sensore laser (in basso). L'asse x è rappresentato in verde, l'asse y in blu, l'asse z in rosso.

nect2_ head_ ir_ optical_ frame, la procedura di creazione dello scan non va a buon fine a meno che, prima di processare i punti, non sia operata una trasformazione opportuna sull'intera nuvola di punti.

Inoltre, nonostante sia presente in `pointcloud_to_laserscan` un parametro che permette di specificare, da launch file, il frame di destinazione desiderato per i dati in ingresso, un'incompatibilità causata dall'utilizzo della libreria `tf2` rende impossibile sfruttare tale possibilità.

Quest'ultimo problema è stato inizialmente aggirato grazie alle caratteristiche del nodo `VoxelGrid` del pacchetto `pcl_ros/Filters`, descritto più approfonditamente nella sezione "Filtraggio".

Come conseguenza di successive scelte implementative, si è dovuti ricorrere direttamente al reperimento della trasformata e alla successiva trasformazione della point cloud. Tale operazione è stata inserita all'interno del nodo `pointcloud_filter`, per la discussione del quale si rimanda sempre alla sezione "Filtraggio".

Andando successivamente ad indagare su quali operazioni influiscano maggiormente sulle prestazioni del nodo appena menzionato in fase di mappatura, si è rilevato come la trasformazione della point cloud costituisca uno dei maggiori colli di bottiglia in tal senso. Disattivando tale operazione, infatti, si riesce a guadagnare fino a quasi 2 Hz nella frequenza con cui sono pubblicati i messaggi nel topic di output del nodo.

Inizialmente non sono state adottate contromisure troppo radicali in tal senso, se non l'accortezza di rendere la trasformazione della nuvola di punti l'ultima operazione svolta dal nodo prima di ripubblicare il messaggio. Tale espediente assicura che la trasformazione sia eseguita coinvolgendo il minor numero possibile di punti.

In seguito, la richiesta di maggiori prestazioni ha spinto ad intervenire direttamente sul codice di `pointcloud_to_laserscan`. Ciò ha permesso di risolvere i problemi legati

all'incompatibilità tra le convenzioni dei frame descritte sopra e di fornire l'occasione per spostare le operazioni di trasformazione più in basso nella catena di elaborazione dei dati sensoriali.

Una trattazione più approfondita di tali cambiamenti tuttavia necessita di una maggiore comprensione degli altri nodi sviluppati e delle problematiche ad essi connesse. Si rinvia pertanto, per avere ulteriori dettagli su queste modifiche finali, alla sezione "Navigazione".

4.2.3 Filtraggio

L'operazione di pre-filtraggio della point cloud ha il compito di eliminare il rumore presente nei dati reperiti da `/kinect2_head/depth_ir/points`, al fine di ottenere uno scan equivalente di buona qualità.

Un altro compito importante affidato ai filtri è quello di ridurre il numero di punti presenti nella point cloud, in modo da alleggerire il carico computazionale richiesto per elaborarla.

La strategia d'azione scelta, pertanto, si articola nei seguenti tre punti:

1. Selezione di un'area di interesse nella nuvola di punti. Tipicamente per fare tale operazione è definito un intervallo di coordinate tra due valori "soglia" lungo un asse del frame; tutti i punti aventi coordinate al di fuori di tale intervallo sono esclusi dal risultato del filtraggio.

2. Sottocampionamento della point cloud. Viene solitamente effettuato dividendo lo spazio in piccole porzioni di ugual volume, dette "voxel". Tutti i punti compresi in un voxel sono sostituiti dal loro centroide.

3. Filtraggio di rumore ed outliers. Consiste nel rimuovere i fenomeni rumorosi presenti nei dati, ed è il passaggio che in genere ha un impatto maggiore sulle prestazioni. Per effettuare tali operazioni si può ricorrere a tecniche euristiche, più leggere ma meno precise ed adattabili, o statistiche, che tendenzialmente offrono risultati migliori ad un prezzo più alto in termini di prestazioni.

Utilizzo di nodi preesistenti

Seguendo le linee guida appena enumerate, si sono andati ad utilizzare i seguenti nodi/nodelet presenti in `pcl_ros/Filters`:

1. `PassThrough` filter, permette di delimitare il range dei punti di interesse, specificando da launch file asse di riferimento, soglia inferiore e soglia superiore. I punti restituiti dal nodo sono solo quelli all'interno (o all'esterno, se si agisce su un flag apposito da launch file) dell'intervallo specificato. Inizialmente si è deciso di tagliare tutti i punti entro la distanza minima di 0.3 m dal sensore, poiché maggiormente interessata da disturbi rumorosi, oltre ad essere di importanza trascurabile ai fini pratici. Sono stati esclusi anche i punti oltre la distanza massima di 3 m, con l'intento di ridurre considerevolmente il numero di punti da processare nei passaggi successivi.

2. `VoxelGrid` filter, permette di effettuare il sottocampionamento, specificando da launch file il lato (parametro `leaf_size`) dei voxel usati per partizionare lo spazio. La scelta della `leaf_size` deve essere ponderata anche in funzione delle caratteristiche del filtro di rimozione degli outliers adottato. All'aumentare della dimensione dei voxel, in genere, si riscontra un generale aumento della velocità di filtraggio ma anche una minore capacità del filtro di individuare efficacemente gli outliers. Dopo diverse prove, si è optato per impostare la `leaf_size` a 0.01 m.

VoxelGrid offre inoltre la possibilità di specificare da launch file, mediante il parametro `output_frame`, il frame di destinazione rispetto al quale trasformare i punti processati. Tale funzionalità è risultata inizialmente di grande utilità, in quanto permette di operare la trasformazione dal frame della Kinect al frame del LRF senza ricorrere ad un nodo apposito.

3. `StatisticalOutlierRemoval` filter, implementa un filtro di tipo statistico per effettuare la rimozione del rumore. Gli unici parametri sui quali è possibile agire sono solamente due, `mean_k`, ovvero il numero di vicini considerati, e `stddev`, un moltiplicatore della deviazione standard che permette di allargare o restringere la soglia oltre la quale un punto è considerato un outlier.

   Sono state necessarie diverse sessioni di tuning dei parametri, in cui si è ricercato anche il connubio ideale con il valore della `leaf_size` del voxel filter, al fine di trovare una soluzione che restituisca un livello di pulizia del rumore sufficiente senza abbassare eccessivamente il frame rate. Si è verificato che un incremento del numero di vicini considerati porta rapidamente ad un peggioramento troppo drastico delle prestazioni per giustificare i moderati miglioramenti ottenuti sul frangente della pulizia. Alla fine si è optato per i seguenti valori: `mean_k` $= 10$ e `stddev` $= 0.1$.

Per aumentare ulteriormente l'efficienza di queste operazioni concatenate, si è deciso di ricorrere all'uso dei nodelet. Questi permettono di evitare le operazioni di copiatura dei dati che normalmente avvengono durante il passaggio di messaggi tra nodi di tipo tradizionale posti in successione. La configurazione realizzata prevede quindi l'utilizzo di un singolo nodelet manager per gestire i tre nodi di filtraggio sopra elencati ed il nodo `pointcloud_to_laserscan`.

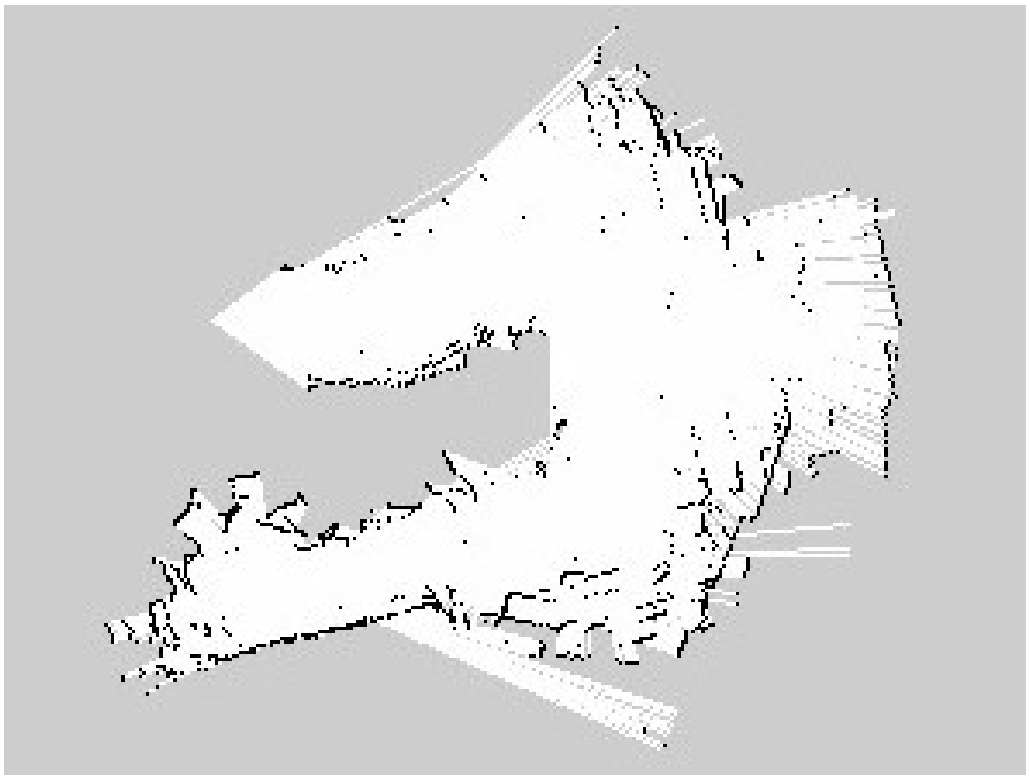


Figura 4.5: Prima prova di mappatura eseguita su un breve percorso in laboratorio. Si notano chiaramente i numerosi segmenti che si ripetono lungo i bordi del percorso seguito dal robot, causati dalla mancata rimozione del terreno dalla point cloud.

Tale messa a punto è stata quindi testata avviando alcune operazioni di mappatura, sfruttando al momento il solo laserscan ottenuto dai dati Kinect (trascurando quindi quello del sensore laser). Fin da subito si sono notati difetti consistenti, quali l'introduzione nella mappa di muri inesistenti, dovuti alla mancata rimozione del terreno dalla point cloud. Un esempio di tale fenomeno, riportato in Figura 4.5, lascia intendere come questi appaiano, durante il movimento, ad una distanza apparentemente costante di fronte al robot.

Creazione di un nodo di filtraggio unificato

Una strategia alternativa all'uso di combinazioni di nodi offerti dalle repository di ROS consiste nel creare un unico nodo, nel quale sono utilizzati tutti i filtri ritenuti necessari.

Tale modo di procedere ha i seguenti vantaggi:

- Permette di utilizzare filtri disponibili nelle librerie di PointClouds.org.
- Garantisce il controllo diretto su tutti i parametri dei filtri della catena, consentendo di adottare azioni più mirate al risolvimento delle criticità.
- Permette di adottare misure volte a migliorare l'efficienza computazionale delle operazioni effettuate.

La scelta di passare a tale strategia è stata effettuata dopo aver indagato sulla natura del problema dei muri inesistenti introdotto precedentemente. Si è velocemente rilevato che questo è dovuto al fatto che, non essendo stati imposti limiti all'altezza dei punti della nuvola, l'algoritmo rappresenta nello scan anche i punti del pavimento. La distanza del muro dal robot infatti corrisponde al punto in cui il FoV della Kinect interseca il piano del suolo.

È stato quindi necessario ricorrere al taglio della point cloud anche lungo l'asse verticale; in particolare, si è scelto di considerare solo i punti in grado di rappresentare un ostacolo per il robot, ovvero da 0.1 m sopra il sensore Kinect a 0.9 m sotto di esso. 0.9 m è inferiore all'effettiva altezza del sensore rispetto al suolo. Si è utilizzato tale valore precauzionale dopo aver osservato che, a causa del beccheggio del robot in movimento, con un valore dell'altezza troppo esatto si verificava la comparsa intermittente di ostacoli inesistenti a distanza variabile dal robot. Una soluzione alternativa, consistente nell'utilizzo di un algoritmo di ground subtraction, avrebbe sicuramente garantito una maggiore robustezza al beccheggio del robot; non è stata tuttavia implementata per non appesantire ulteriormente il già consistente carico computazionale.

La necessità di rimuovere anche la parte inferiore della point cloud ha motivato l'abbandono dell'uso del PassThrough filter in favore di un filtro di tipo ConditionAnd (disponibile nelle repository ufficiali di PointClouds.org), il quale permette di applicare in un unico passaggio di filtraggio tutte le soglie desiderate.

Si è quindi creato il pacchetto `pointcloud_filter`, il quale contiene un nodo omonimo che applica in successione il filtro appena descritto, il voxel filter, trasforma la point cloud ed infine opera uno statistical outlier removal. Il risultato è infine ripubblicato nel topic `/kinect2_filtered`, dove sarà in ascolto il nodo `pointcloud_to_laserscan`.

Dopo le prime fasi di test dell'intero sistema durante la mappatura è emersa la necessità migliorare alcuni dettagli riguardanti il funzionamento del nodo.

Figura 4.6 illustra un interessante risultato ottenuto da alcune prove effettuate utilizzando il solo sensore Kinect. Esse infatti evidenziano come la ricostruzione dell'ambiente venga più precisa ed accurata adottando un range massimo per i punti delle point cloud ben oltre i 4.5 metri di range massimo dichiarato dal costruttore. A causa infatti del ridotto FoV del dispositivo, effettuando la mappatura con un range ridotto spesso l'algoritmo non trova sufficienti match tra i precedenti ostacoli rilevati e gli attuali dati sensoriali, con il risultato che la mappa calcolata contiene numerose aree "vuote" (in grigio).

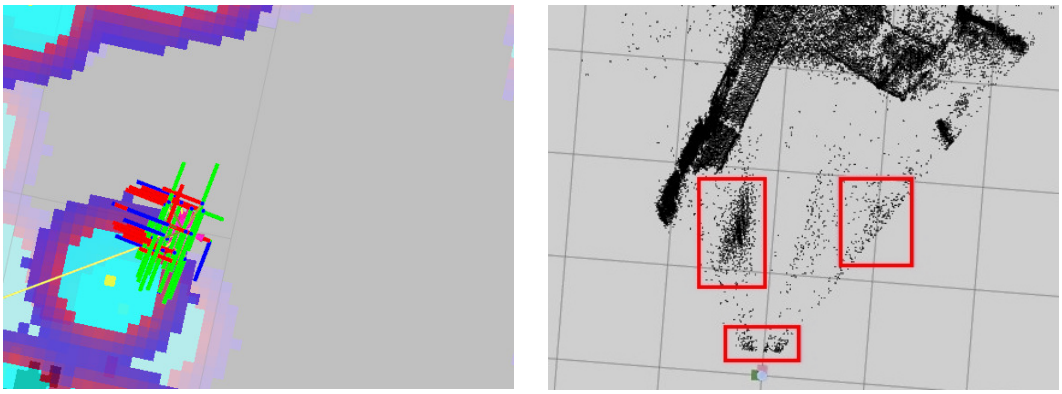
(a) (b) (c)

Figura 4.6: Da sinistra verso destra, le mappe ottenute dalla stessa bag di test utilizzando la sola Kinect con range, rispettivamente: 3, 5 e 10 m.

Dopo essersi accertati che l'incidenza sulle prestazioni derivante dall'utilizzo di un range più elevato è poco significativa, sia il range massimo fornito al filtro ConditionAnd di `pointcloud_filter` sia il massimo range fornito a `pointcloud_to_laserscan` sono stati impostati a 10 m.

Il filtraggio ha avuto un ruolo chiave anche per quanto riguarda l'efficacia della navigazione.

Nei primi test di tal sorta, infatti, il robot ha dimostrato serie difficoltà a raggiungere le posizioni specificate sulla mappa. Anche nei casi di successo, le traiettorie scelte non si sono rivelate ottimali; al contrario, il robot spesso ha deviato dal percorso più breve, fermandosi frequentemente per ricalcolare il percorso ed occasionalmente avviare le procedure di recovery.



(a) (b)

Figura 4.7: Esempio di ostacolo puntiforme creatosi a distanza ravvicinata dal robot (a sinistra) e dei blob rumorosi che ne sono la causa (a destra).

Esaminando la mappa locale da RViz, si è appurato che ciò è dovuto alla rilevazione di ostacoli, per lo più puntiformi, non realmente presenti di fronte al robot. Un esempio di tale fenomeno è riportato in Figura 4.7a. Simili disturbi non hanno mai avuto alcuna incidenza nei test di mappatura a causa del loro carattere intermittente e dell'esiguo numero di punti che li costituiscono.

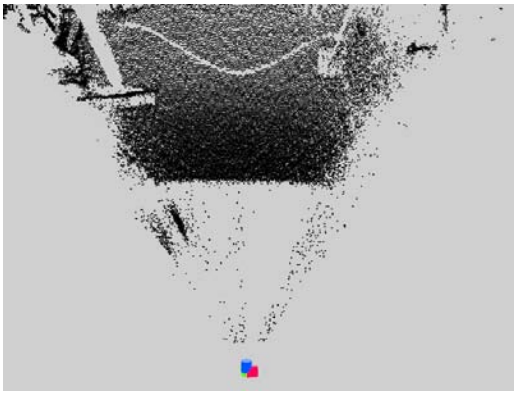
Andando così a condurre test in diverse aree della mappa, si è rilevato come le zone maggiormente interessate da tali effetti rumorosi siano l'area libera di fronte al robot UR10 e la zona nei pressi della porta che dal laboratorio dà sul corridoio. Si è quindi proceduto

con l'analisi delle point cloud filtrate per capire quali tecniche adottare per eliminare tali ostacoli.

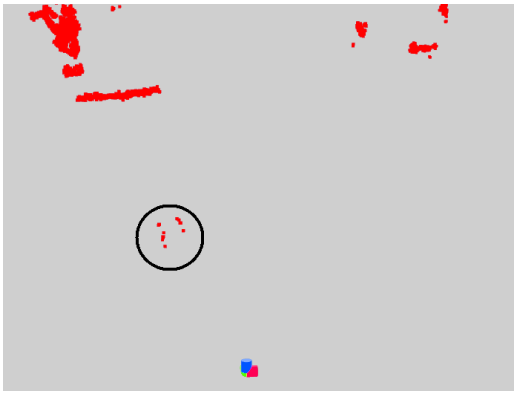
Come si può vedere in Figura 4.7b, il carattere di tali disturbi, specie in particolari periodi della giornata, assume un'incidenza non indifferente, presentandosi in forma di veri e propri "blob" di punti. Numerosi tentativi hanno dimostrato l'inefficacia del solo filtro StatisticalOutlierRemoval nel rimuoverli, anche adottando configurazioni dei parametri più aggressive, a spese del frame rate.

A questo punto sono state pensate e, successivamente, implementate e testate le seguenti contromisure:

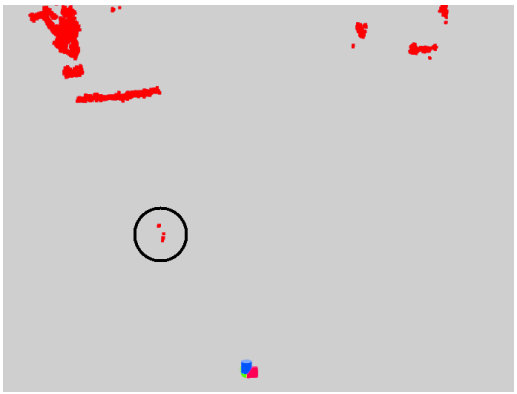
1. Divisione della point cloud in due "fasce" di punti, discriminate introducendo una soglia sui valori dell'asse z del frame *kinect2_head_ir_optical_frame*. Ciò permette di adottare politiche di filtraggio più aggressive per i punti più prossimi al robot, ovvero quelli rientranti nella mappa locale dell'algoritmo di navigazione. Al contempo viene attenuata l'aggressività del filtraggio dei punti più distanti, di interesse solamente ai fini della localizzazione, dove l'eventuale rumore non influisce sulle sorti del planning del percorso. Per la soglia in questione è stato scelto il valore di 2 m. Le due porzioni di point cloud sono ricongiunte subito dopo le operazioni di filtraggio.



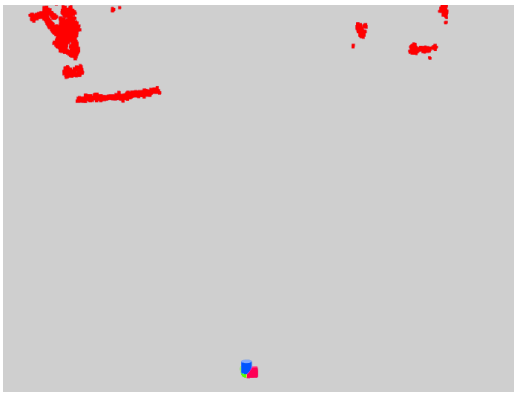
(a) Point cloud originale.



(b) Primo filtro: ROR.



(c) Secondo filtro: SOR.



(d) Terzo filtro: ROR.

Figura 4.8: Effetto della catena di filtri utilizzata su una point cloud rumorosa.

2. Utilizzo del filtro **RadiusOutlierRemoval**, in congiunta con il filtro StatisticalOutlierRemoval, nella porzione di point cloud più prossima al robot. Tale filtro permette di rimuovere tutti i punti non aventi, entro un dato raggio (impostato mediante il metodo `setRadiusSearch`) un numero minimo di vicini (impostato mediante il metodo `setMinNeighborsInRadius`). Diverse combinazioni dei due filtri sono state provate, arrivando ad applicare anche più passaggi di ogni singolo filtro, fino a giungere ad

una strategia in grado di eliminare efficacemente tutti i punti la cui distribuzione nello spazio è talmente poco densa da farne intuire la natura rumorosa.

I punti sono quindi processati usando la seguente sequenza di filtri, della quale Figura 4.8 illustra i risultati parziali applicati alla cloud rumorosa di Figura 4.8a:

- RadiusOutlierRemoval, `radius`: 0.01, `neighbors`: 3. (Figura 4.8b)
- StatisticalOutlierRemoval, `mean_k`: 10, `stddevmulthresh`: 0.01. (Figura 4.8c)
- RadiusOutlierRemoval, `radius`:0.05, `neighbors`: 25. (Figura 4.8d)

Per la point cloud più distante, invece, si è optato per un più leggero singolo passaggio di StatisticalOutlierRemoval, con un `mean_k` dimezzato rispetto al parametro indicato sopra e `stddevmulthresh` meno restrittivo di un fattore 10.

3. Irrobustimento ulteriore del filtraggio dei blob rumorosi, intervenendo anche direttamente sullo scan derivato dalla point cloud. Inizialmente si è provato concatenando all'output di `pointcloud_to_laserscan` i nodi di filtraggio dei dati laser offerti dal pacchetto ROS `laser_filters`. Questi tuttavia si sono dimostrati totalmente inefficaci nel nostro caso: in alcune situazioni il rumore veniva addirittura amplificato, ad esempio utilizzando il filtro interpolatore sfruttato anche in `minimal.launch`.

L'insuccesso registrato solo all'ultimo punto ha portato ad optare per un'azione più radicale, consistente nella modifica ed integrazione in `pointcloud_filter` del codice del nodo `pointcloud_to_laserscan`. Il nodo ottenuto, chiamato `pointcloud_filter_and_scan`, rende possibile introdurre il passaggio di filtraggio all'interno del processo di proiezione della point cloud in un messaggio di tipo `sensor_msgs/Laserscan`, con evidenti risultati sia in termini di efficacia che di efficienza computazionale. Maggiori approfondimenti riguardanti quest'ultima operazione sono riportati nella sezione "Navigazione".

4.2.4 Fusione degli scan

Il passaggio di fusione tra lo scan ottenuto dal sensore Hokuyo e quello ottenuto dalla Kinect è ritenuto importante a causa dell'altezza alla quale è montata la telecamera. Complice il ridotto FoV verticale, usando la sola Kinect è infatti impossibile per il robot individuare ostacoli dalla ridotta altezza rispetto al terreno posti a meno di 2 m di fronte ad esso.

Fin dai primi test con il visualizzatore RViz, si è notato come lo scan derivato dalla point cloud presentasse un visibile disallineamento dallo scan derivato dal sensore Hokuyo, seppur dell'ordine di pochi centimetri. Indagando ulteriormente sulla natura di questo problema, si è arrivati a ritenere che ciò sia dovuto ad un insieme di cause. La principale è l'imprecisa specifica della trasformazione tra i frame di riferimento dei due sensori, soggetta a variazioni sul lungo periodo di utilizzo a causa del supporto sul quale è montata la Kinect, non in grado di garantire il mantenimento della posa esatta del dispositivo in presenza di sollecitazioni esterne. Altre cause concorrenti sono da ricercarsi nel beccheggio del robot in movimento e nel grado di precisione offerto dagli strumenti stessi.

Tale problema è tuttavia risolvibile mediante opportune procedure di calibrazione dei sensori, le quali portano ad ottenere una trasformata corretta tra i due frame. La discussione di tali tecniche esula tuttavia dallo scopo di questa esperienza.

Utilizzo di nodi preesistenti

Come prima operazione si è effettuata una ricerca di pacchetti e metodi preesistenti adatti a svolgere il compito di fondere coppie di messaggi di tipo `sensor_msgs/LaserScan`.

È stato così individuato un pacchetto ROS, `iralabdisco/ira_laser_tools`, sviluppato dal laboratorio di informatica e robotica per l'automazione dell'università di Milano-Bicocca (IRALAB) e reso disponibile su GitHub [2]. Uno dei due eseguibili disponibili in tale pacchetto, `laserscan_multi_merger`, è essenzialmente un nodo pensato per fondere diversi topic laser, dei quali è nota la trasformata che relaziona i frame. Nel caso in esame, tuttavia, la trasformazione della point cloud avviene in fase di filtraggio, pertanto i dati che giungono al nodo di fusione sono già riferiti al frame *base_laser_link*.

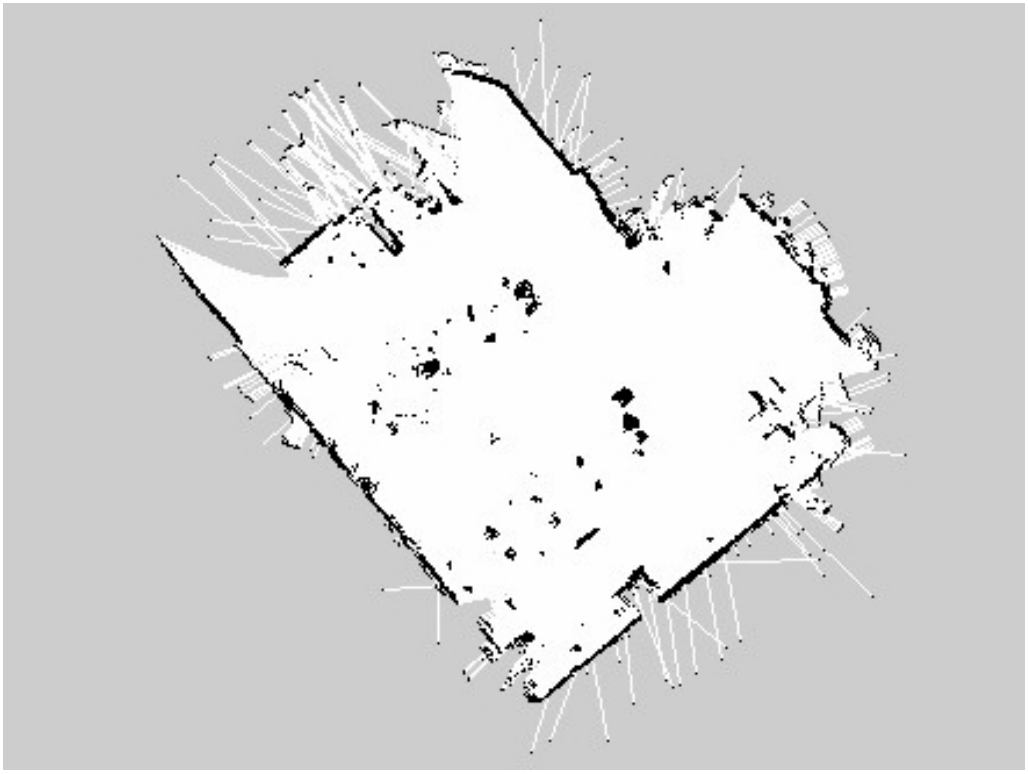


Figura 4.9: Esempio di mappa ottenuta integrando i dati Kinect con quelli del sensore LRF, mantenendo però le impostazioni di default per angolo minimo e massimo dei dati laser. Si nota chiaramente come in queste condizioni il contenuto informativo della mappa sia essenzialmente lo stesso di una mappa ottenuta sfruttando il solo laser.

La ripetizione di alcuni test di mappatura ha evidenziato una qualità delle mappe ottenute ancora eccessivamente deludente. Oltre al persistente problema del basso frame rate con cui sono pubblicati i dati fusi (dell'ordine di 2-3 Hz), si è constatato il più grave fatto che non tutti gli ostacoli nel circondario del robot vengono rappresentati nella mappa. In particolare, molti ostacoli, inizialmente rilevati, sono successivamente cancellati nei successivi aggiornamenti causati dal movimento del robot, con il risultato che la mappa finale è simile a quelle ottenute con il solo laser (un esempio è riportato in Figura 4.9).

La causa di ciò è stata individuata nel fatto che vengono di volta in volta fusi due messaggi laser provenienti da sensori con FoV differenti: ogni volta che il robot ruota, alcuni ostacoli escono dalla ristretta inquadratura della Kinect senza persistere nella finestra di visione dello scan fuso, pari a quella, molto più ampia, del sensore laser. L'algoritmo GMapping pertanto rimuove tali ostacoli in quanto, al pari di disturbi luminosi di breve durata o ostacoli dinamici di vario genere (persone di passaggio), la loro presenza in quella determinata posizione è supportata da relativamente pochi frame, uscendo pertanto penalizzata dalla logica di funzionamento del particle filter.

[2] link al repository: `https://github.com/iralabdisco/ira_laser_tools`

La soluzione trovata a questo problema è stata quella di ridurre il FoV del laser, tenendo solamente l'area frontale avente apertura angolare 70.6°, ovvero quella esattamente in sovrapposizione con il FoV del sensore Kinect. Per fare ciò è stato sviluppato il pacchetto `laser_range_adapter`, il quale prende in input il topic `laser_scan` e permette di ripubblicare messaggi di tipo `sensor_msgs/LaserScan` ridimensionati a precisati valori per angolo minimo, angolo massimo, range minimo e range massimo.

La maggiore critica sollevata nei confronti di `laserscan_multi_merger` riguarda l'apparente lentezza necessaria a svolgere il compito preposto. La causa di ciò è data dall'ambito di utilizzo più ampio per il quale è stato pensato il nodo, comprendente anche casi di fusione di dati laser riferiti a frame differenti.

Non essendo possibile trasformare il tipo di dato `sensor_msgs/LaserScan` utilizzando le librerie `tf` o `tf2` di ROS, è necessario convertire ogni scan in una nuvola di punti, quindi trasformare quest'ultima in modo da ottenerla riferita al medesimo frame, ed infine ritrasformarla in dato laser con un processo uguale a quello usato da `pointcloud_to_laserscan`. Queste ultime operazioni costituiscono solo un motivo di perdita di frame rate, poiché non necessarie: nel caso in esame, infatti, al nodo vengono forniti dati laser già riferiti al medesimo frame, perfettamente sovrapponibili (quindi i vettori contenenti range dei punti laser hanno uguali dimensione, angolo minimo ed angolo massimo).

Creazione di un nodo di fusione

I motivi appena elencati hanno giustificato l'abbandono del pacchetto `ira_laser_tools` in favore di un pacchetto creato specificatamente per questa determinata situazione, chiamato `orobot_laser_merger`. Questo si limita ad iterare lungo le celle di uguale indice dei due messaggi `sensor_msgs/LaserScan` ottenuti dai due topic ai quali si sottoscrive, salvando nella corrispondente cella dell'array del messaggio che verrà restituito il range di valore più basso tra quello delle due celle considerate.

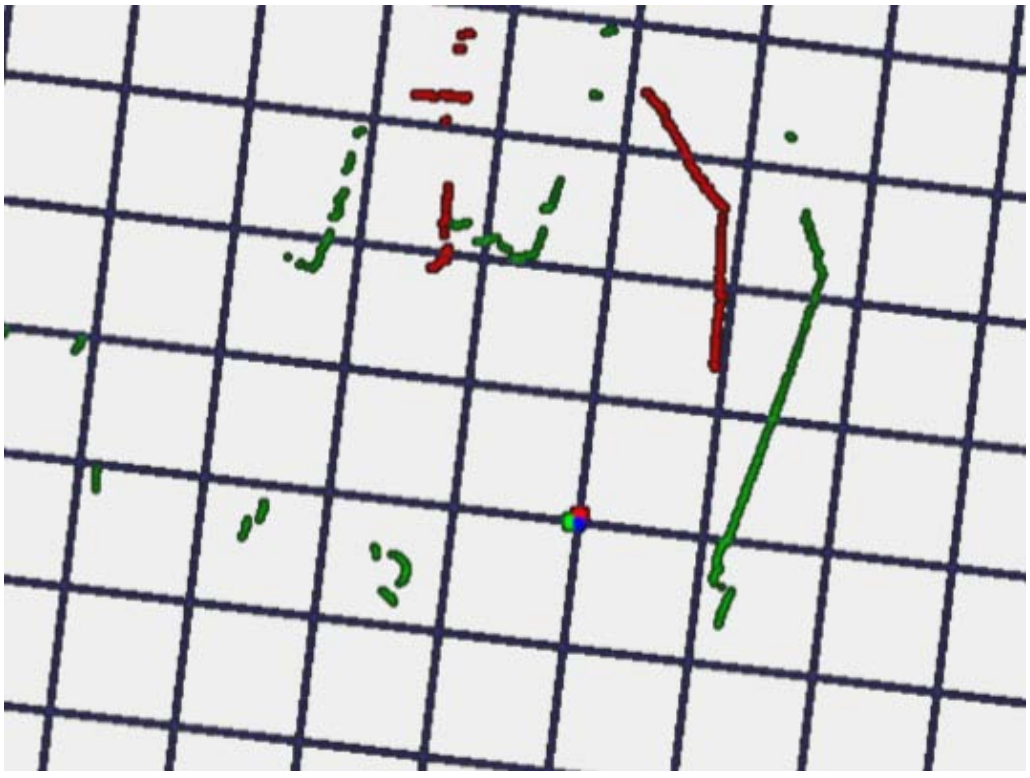


Figura 4.10: Esempio di disallineamento (durante una rotazione del robot) tra lo scan ricavato dal sensore LRF (verde) e quello ottenuto dal processing dei dati kinect (rosso).

In questa fase la sincronizzazione tra i messaggi dei due topic in ingresso ha costituito una criticità. Gestire i messaggi senza operare alcuna forma di sincronizzazione (vengono fusi i due messaggi più recenti pubblicati nei due topic) è infatti deleterio a causa del delay introdotto dalle operazioni che permettono di trasformare una point cloud in uno scan. Il risultato che si ottiene presenta un frame rate pari a quello del topic con maggiore frequenza di aggiornamento, tuttavia descrive male la realtà dei fatti. Figura 4.10 ritrae un'istantanea degli ultimi due scan pubblicati nei topic `kinect2_scan` e `laser_scan` mentre robot ruota verso destra. Si intuisce chiaramente che il ritardo del topic ottenuto dalla Kinect (rosso) rispetto a quello pubblicato dal laser (verde) rende impossibile sovrapporre i due scan senza causare la comparsa di occlusioni e barriere inesistenti nella mappa ricostruita da GMapping.

L'introduzione di una condizione che impone al nodo di aggiornare i messaggi con la stessa frequenza del topic più lento ha apportato migliorie troppo lievi per potersi ritenere soddisfatti.

Fortunatamente è disponibile nei repository ROS uno strumento utile alla sincronizzazione di messaggi provenienti da diversi topic, basato sul campo `timestamp` dell'header dei messaggi. Il timestamp viene tramandato immutato da tutti i metodi che operano sulla point cloud, pertanto i timestamp che giungono al nodo di fusione dei due scan sono relativi all'istante di pubblicazione della point cloud e del messaggio laser da parte dei relativi sensori. Lo strumento di sincronizzazione, implementato come un oggetto `message_filters::Synchronizer`, richiede che sia specificata una callback da invocare ogni volta che viene pubblicata una coppia di messaggi "corrispondenti" secondo la policy di sincronizzazione scelta.

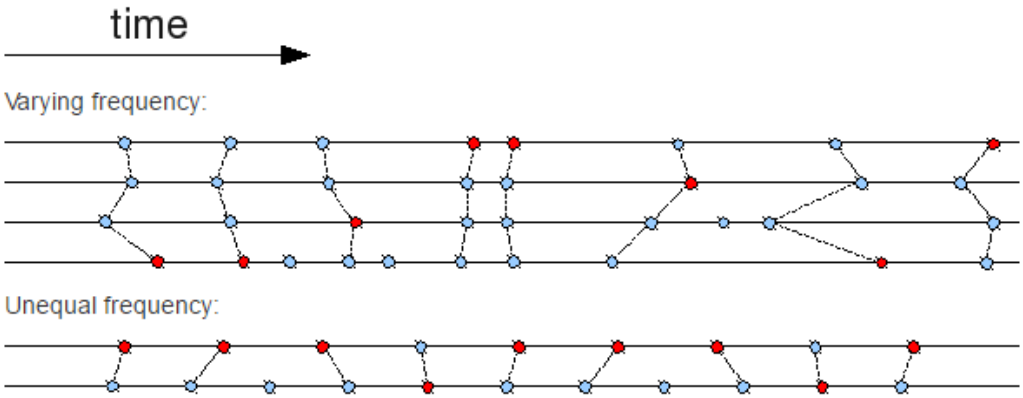
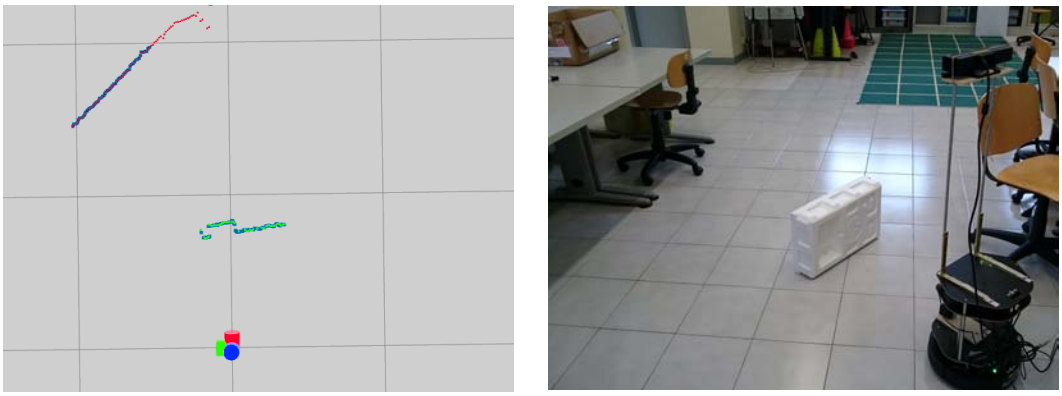


Figura 4.11: Funzionamento della policy `ApproximateTime` del sincronizzatore di tipo `message_filters::Synchronizer`. Ogni linea rappresenta un diverso topic, i punti sono i messaggi pubblicati. I punti di topic differenti collegati tra loro sono utilizzati dalla medesima callback.
(Fonte: `http://wiki.ros.org/message_filters/ApproximateTime`).

Riguardo alla modalità con cui vengono associati i messaggi, si può scegliere se adottare una `ExactTime` policy oppure una più adattiva `ApproximateTime` policy. Nel primo caso la callback è invocata solo in occasione di messaggi aventi il medesimo timestamp; nel secondo invece sono accettati anche lievi sfasamenti temporali tra i messaggi pubblicati, come illustra l'infografica di Figura 4.11. Dopo differenti prove, si è rilevato come per i nostri scopi convenga scegliere quest'ultima policy.

Proseguendo nei test, si è riscontrato il raggiungimento dell'obiettivo di rappresentare correttamente nelle mappe gli ostacoli più prossimi individuati dai sensori montati. Un



(a) (b)

Figura 4.12: Esempio di funzionamento della fusione (azzurro) tra lo scan ottenuto dalla kinect (rosso) e quello ottenuto dal laser (verde). Nella figura di destra è riportata la scena che il robot sta inquadrando.

esempio dell'output fuso, sovrapposto agli scan ottenuti da sensore Hokuyo e Kinect, è riportato in Figura 4.12.

(a) (b) (c)

Figura 4.13: Da sinistra verso destra, le mappe di test ottenute utilizzando rispettivamente: solo laser, solo Kinect ed entrambi.

Figura 4.13 paragona tre mappe, ottenute sfruttando rispettivamente solo il laser, solo la Kinect e lo scan fuso. Tale prova ha permesso di individuare un altro problema, avente la particolare tendenza a manifestarsi dopo alcuni minuti dall'inizio della sessione di mappatura.

Il cattivo risultato della mappa relativa allo scan fuso (Figura 4.13c), infatti, è dovuto ad una ricorrente interruzione totale, per diversi secondi, della pubblicazione di messaggi da parte del nodo `orobot_laser_merger`. Dopo aver indagato ed escluso differenti possibili cause di questo comportamento, si è scoperto che il bug era dovuto ad un valore troppo basso impostato per il parametro che determina, nell'istanza di `message_filters::Synchronizer`, la lunghezza massima della coda di messaggi in ingresso: dopo aver portato la dimensione di tale coda e delle code in ingresso ai due subscriber da 10 a 50 messaggi, le performance del nodo di fusione hanno dimostrato un vistoso miglioramento ed il malfunzionamento non si è più manifestato.

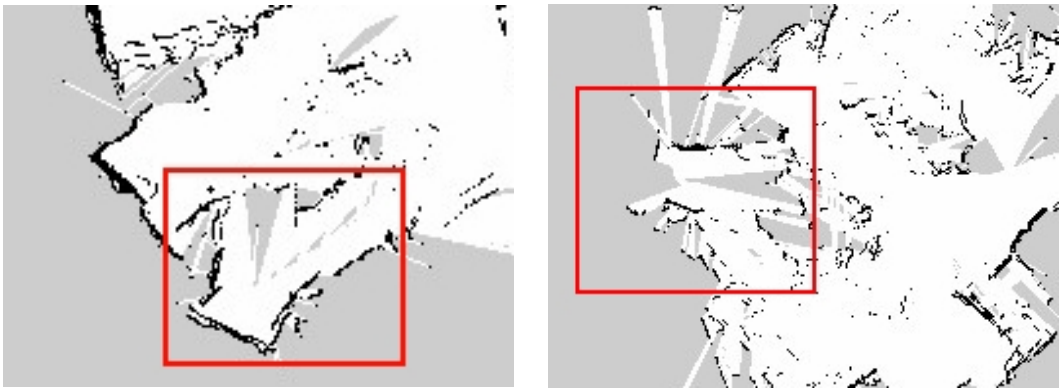
Un altro aspetto critico rilevato grazie a questi test riguarda la scelta di ridurre il FoV di entrambi i dispositivi: se ora gli ostacoli precedentemente rilevati non vengono più cancellati dalla mappa, è anche vero che la costruzione della mappa si rivela maggiormente difficoltosa, a causa del ristretto campo di visione. Ciò è dimostrato dalla bassa qualità

della mappa ottenuta sfruttando il solo laser (Figura 4.13a). Per ottenere una copertura efficace di tutta l'area è quindi opportuno far passare il robot più volte per zone già visitate, preferendo una traiettoria a zig-zag piuttosto che una rettilinea, al fine di favorire la miglior rappresentazione degli ingombri posti ai lati del robot, i quali altrimenti andrebbero inevitabilmente persi.

Un'ultima correzione nell'ambito della fusione tra i due flussi di dati riguarda il range dei dati laser. È stato constatato che il raggio d'azione entro il quale il sensore laser svolge un ruolo fondamentale corrisponde alla distanza entro la quale il piano del laser interseca il FoV verticale della Kinect. Oltre tale distanza l'informazione fornita è ridondante o sovrascritta da rilevazioni più prossime fatte dalla Kinect. Mantenere tuttavia il range originale degli scan può essere fonte di disturbi nelle mappe, a causa di raggi recanti valori errati o di un'eccessiva sparsità dei punti della point cloud in un'area. Da ciò consegue che alcuni raggi laser possono erroneamente contribuire allo scan finale del nodo merger nonostante si riferiscano a ostacoli più distanti.

Come conseguenza di ciò si è scelto di limitare il range del laser (sfruttando il nodo `laser_range_adapter`) a 1.625 m, distanza alla quale il FoV della Kinect teoricamente interseca il piano del laser.

4.2.5 Mappatura



(a) (b)

Figura 4.14: Esempio di errore di ricostruzione della mappa in seguito ad una rotazione del robot (a sinistra è stata utilizzata la sola Kinect, a destra sia Kinect che Hokuyo).

Oltre alle già discusse modifiche apportate ai nodi di filtraggio e fusione, i risultati di alcuni test di mappatura hanno sollevato alcuni problemi non strettamente legati a tali nodi. Ad esempio, esaminando sempre il confronto riportato in Figura 4.13, si nota come la rotazione del robot costituisca uno dei maggiori problemi per il nodo di mappatura. Figura 4.14 riporta due dettagli riferiti alla stessa area ripresa rispettivamente in Figura 4.13b (a sinistra) e Figura 4.13c (a destra), dove è visibile una sorta di "biforcazione" nel corridoio inferiore della mappa. In tale posizione al robot è stato fatto compiere un giro di 360° su se stesso. L'errata sovrapposizione tra l'inquadratura ottenuta dall'acquisizione iniziale e l'inquadratura rilevata a rotazione terminata dà origine a questo fastidioso effetto.

Si è dedotto poi che gran parte della colpa dei cattivi risultati ottenuti in Figura 4.13b fosse imputabile al frame rate ancora troppo basso, che si attesta (con computer alimentato a batteria) attorno ai 4.5 Hz.

La prima e più semplice contromisura adottata per far fronte a tali problemi è stata quella di impostare al minimo valore possibile le velocità massime (lineare ed angolare) del robot. Si è avviato quindi un lavoro di ottimizzazione del codice, volto ad individuare ed eliminare le inefficienze introdotte nei nodi creati.

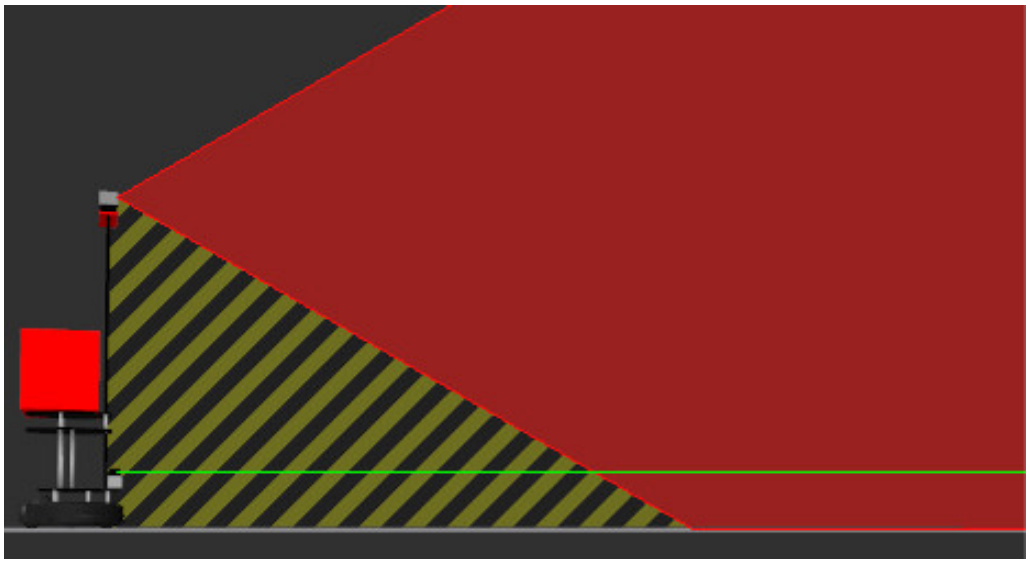


Figura 4.15: Area "cieca" del robot (pattern giallo-nero). Tutti gli oggetti compresi in quest'area che non intersecano il piano del sensore laser (verde) non sono rilevati.

Un altro dato rilevato durante i test riguarda, invece, un punto di debolezza "sistematico", ovvero dipendente dalla struttura del robot e pertanto non immediatamente risolvibile se non con l'adozione di particolari precauzioni in fase di mappatura dell'ambiente. Passando eccessivamente vicini ad un tavolo o ad una sedia, infatti, GMapping tende a correggere la mappa spostando il precedentemente riconosciuto bordo del tavolo ad una maggiore distanza dal robot, vanificando così le precedenti rilevazioni (corrette). Ciò è causato dall'esistenza di un'area cieca per entrambi i sensori, rappresentata in Figura 4.15.

La migliore soluzione in questo caso rimane quella di evitare il passaggio troppo vicino a questi oggetti in fase di mappatura, cercando, laddove si noti un degrado della mappa dovuto a tale difetto, di ri-inquadrare l'area interessata da una maggior distanza per permetterne la correzione.

Le migliorie introdotte nel codice, il miglior tuning dei parametri, la riduzione della velocità del robot e le precauzioni adottate hanno consentito l'ottenimento della mappa rappresentata in Figura 4.16. Durante tutta la mappatura il topic contenente lo scan fuso ha pubblicato messaggi con una frequenza sempre superiore ai 5 Hz.

4.2.6 Navigazione

Giudicati soddisfacenti i risultati ottenuti con la mappatura, si è ricavata una mappa dell'intero laboratorio ed utilizzando questa si è proceduto al test del funzionamento della navigazione del robot verso goal prefissati.

Si è così constatato che sporadicamente si verificano episodi di de-localizzazione: situazioni in cui il robot ricalcola la sua posizione sulla mappa, sfruttando le informazioni sugli ostacoli circostanti, riposizionandosi poi in un luogo sbagliato.

Indagando ulteriormente sulle cause di tale fenomeno, si è scoperto che la de-localizzazione si manifesta solo in scenari in cui l'ambiente dal quale viene avviata la navigazione è cambiato sensibilmente rispetto a quando è stata acquisita la mappa (situazione ricorrente a causa della presenza di altre persone nell'ambiente di test).

Unificazione di `pointcloud_filter` e `pointcloud_to_laserscan`

La necessità di reagire con prontezza ad eventuali ostacoli dinamici e la difficoltà dell'elaboratore nel gestire il carico computazionale hanno motivato la ricerca di migliori prestazioni

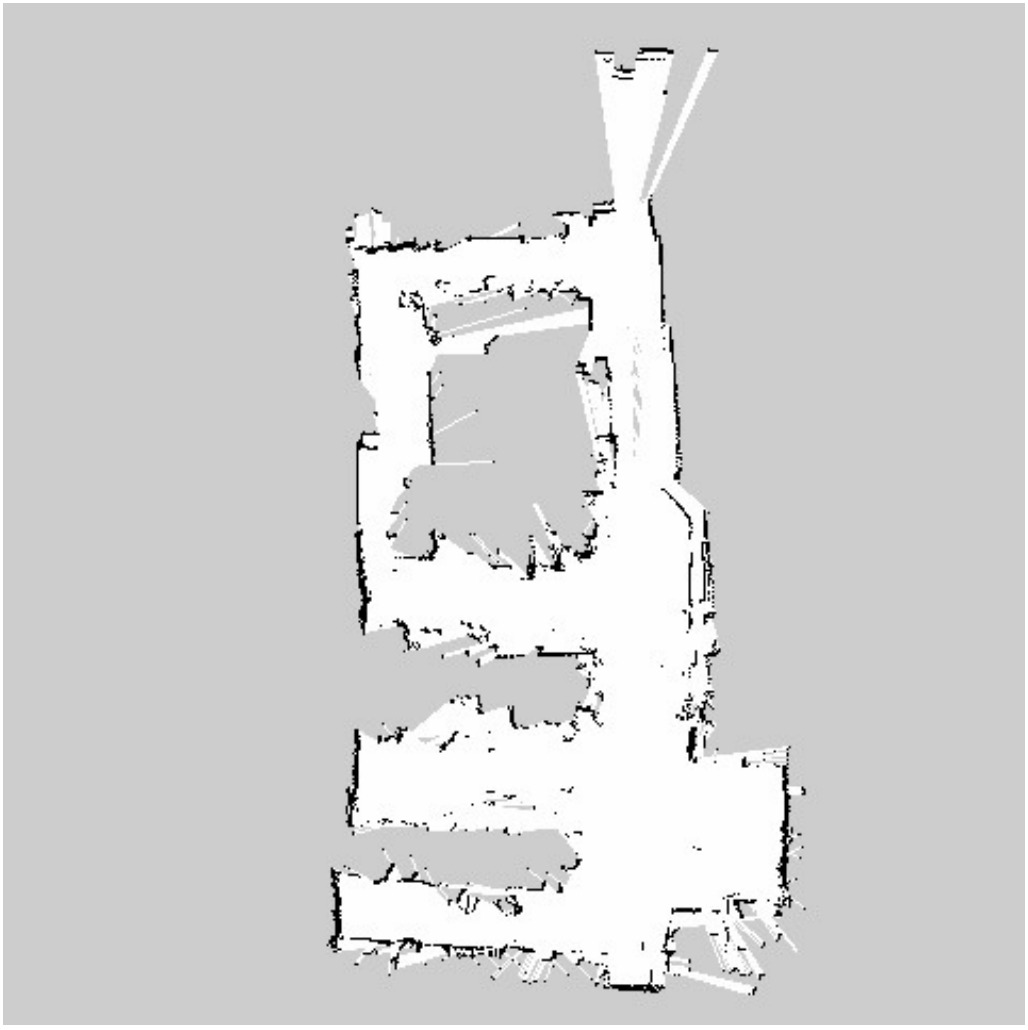


Figura 4.16: Mappa ottenuta dopo aver allungato le code dei messaggi in ingresso al nodo `orobot_laser_merger`.

in termini di frame rate dello scan fuso.

Tali ragioni hanno spinto a modificare il codice di un nodo finora non interessato da alcuna operazione di ottimizzazione: `pointcloud_to_laserscan`. I vantaggi che ne sono derivati sono molteplici:

- Come già spiegato, per adottare tecniche di filtraggio efficienti su dati di tipo `sensor_msgs/LaserScan`, è necessario convertire i dati nel formato utilizzato per le point cloud, `sensor_msgs/PointCloud2`. Agire su `pointcloud_to_laserscan` permette di evitare tale inefficienza, in quanto si può introdurre il filtraggio direttamente all'interno del processo di conversione da point cloud a dato laser.

  Il funzionamento di `pointcloud_to_laserscan` infatti è basato su un loop di iterazione su tutti i punti della nuvola. Per ogni punto è calcolata la componente orizzontale della distanza dal sensore (tramite la funzione `hypot`) e l'angolo del raggio corrispondente (usando la funzione `atan2`), usato per risalire alla cella corretta del vettore `ranges` dello scan in output. In ogni cella di quest'ultimo vettore (inizializzata a range infinito) viene salvato via via il più piccolo range calcolato relativo a quel determinato raggio.

  Facendo sì che tale ciclo salvi in una nuova point cloud i punti relativi ai raggi minimi individuati, si ottiene una nuvola di punti dalle dimensioni ridotte sulla quale

è possibile applicare qualsiasi filtro si desideri. Quindi, iterando su tutti i punti in uscita dal filtro, si ottiene un messaggio LaserScan vero e proprio.

Nel caso in esame il filtro scelto è RadiusOutlierRemoval; empiricamente sono stati individuati come valori ottimali 0.03 e 2 rispettivamente per raggio e numero di vicini.

- È già stato discusso il fatto che il peso computazionale del cambiamento del frame di riferimento è un fattore limitante per le prestazioni.

  Procedere come descritto al punto precedente porta tuttavia un altro vantaggio: si possono cambiare le convenzioni degli assi lungo i quali calcolare le distanze dei punti per lo scan, permettendo così di operare direttamente su dati riferiti al frame *kinect2_head_ir_optical_frame*. Ciò permette di spostare il cambio di frame (comunque necessario per operare correttamente la fusione dei due topic laser) più in basso nella catena di operazioni sulla cloud, per la precisione subito dopo al passaggio di filtraggio descritto al punto precedente.

  La drastica riduzione del numero di punti da trasformare permette un sensibile miglioramento delle prestazioni, con la frequenza dei messaggi pubblicati nel topic `/scan` stabilmente tra i 5.5 ed i 6 Hz. Tali valori costituiscono un buon risultato, considerati anche i diversi passaggi di filtraggio aggiuntivi introdotti per risolvere i problemi di navigazione (discussi alla fine della sezione "Filtraggio").

- Dover intervenire sul codice del pacchetto permette di cogliere l'occasione per eliminare un'altra inefficienza del sistema adottato, ovvero il passaggio di messaggi di tipo `sensor_msgs/PointCloud2` tra il nodo di filtraggio ed il nodo di conversione in dati scan. Si è quindi deciso di introdurre il codice del nodo in esame in coda a quello di `pointcloud_filter`, generando così un nuovo nodo appartenente al pacchetto `pointcloud_filter`, chiamato `pointcloud_filter_and_scan`.

L'adozione di questi espedienti ha sortito gli effetti desiderati, permettendo al robot di seguire in maniera lineare e stabile il path calcolato inizialmente anche passando per i punti più critici della mappa.

4.3 Risultati sperimentali

4.3.1 Mappatura

In fase di implementazione, al fine di individuare e risolvere i problemi che via via si sono presentati, sono stati effettuati differenti test sul campo. Tuttavia, per ottenere confronti maggiormente oggettivi sulla qualità delle scelte fatte, si è anche utilizzato un utile strumento, messo a disposizione dalla libreria standard ROS, chiamato "bag". Con tale termine si indica un file contenente la registrazione della sequenza di tutti i messaggi pubblicati in alcuni topic di interesse (indicati a priori dall'operatore del robot) in un certo lasso di tempo. Tale file permette quindi di simulare un determinato percorso seguito dal robot, con i relativi input sensoriali, senza dover utilizzare il robot stesso tutte le volte che si desidera effettuare un test comparativo tra più strategie.

Una volta risolti i problemi iniziali che caratterizzano i nodi di filtraggio e fusione creati, si è ritenuto opportuno registrare una bag relativa ad un percorso contenente diversi punti giudicati di particolare interesse, al fine di valutare la bontà della ricostruzione fatta.

In particolare, con riferimento a Figura 4.17:

- Il robot inizia (punto "A") compiendo un giro di 360° su se stesso, dando la possibilità di valutare l'abilità dell'algoritmo di ricostruire la mappa anche in fase di rotazione prolungata. Tale operazione, come si è già discusso, è particolarmente critica a causa

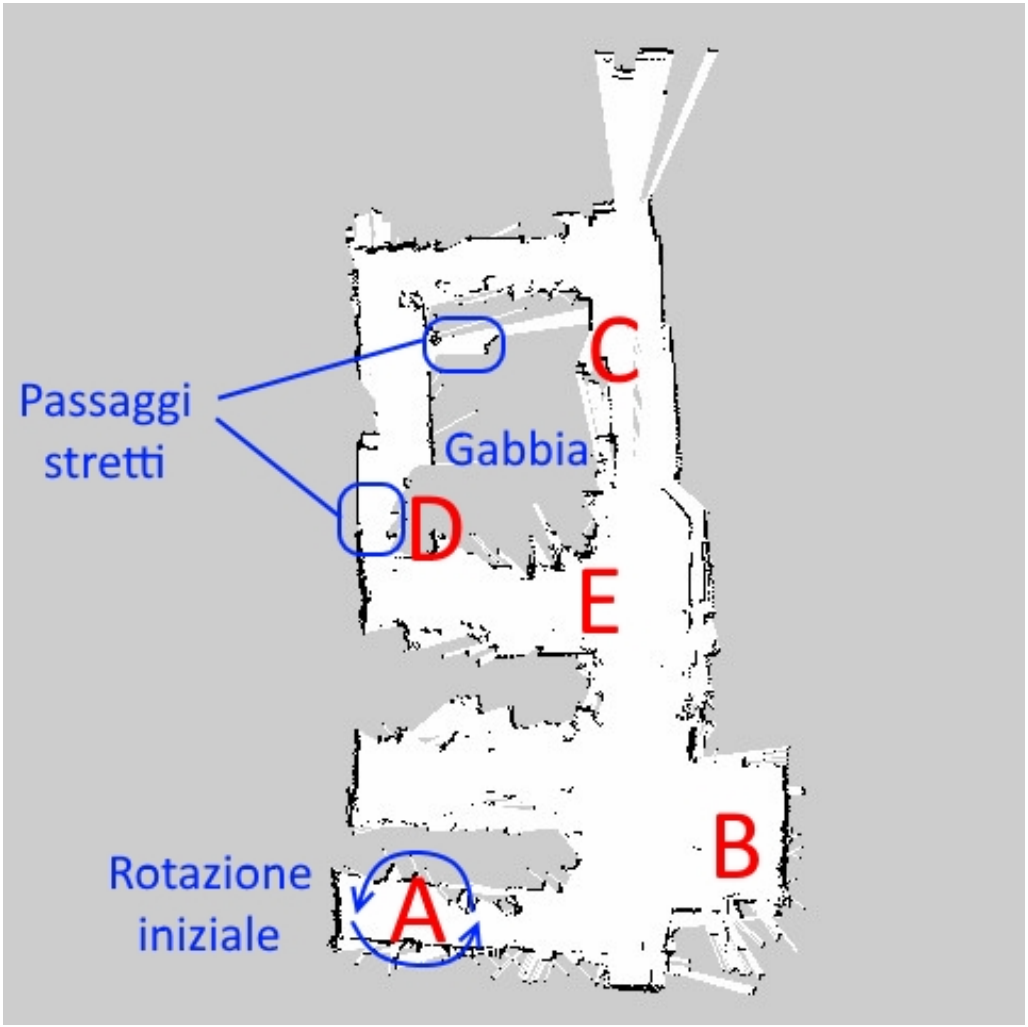


Figura 4.17: Mappa con riportate le principali tappe del percorso di test.

dell'imperfetta sovrapposizione dei dati laser-Kinect. Serve inoltre per verificare se il frame rate riesce ad essere all'altezza di una situazione dove sono richiesti frequenti aggiornamenti dai sensori.

- Dopo un breve tratto rettilineo delimitato da tavoli, il cui corretto riconoscimento è prerogativa della Kinect, il robot viene diretto in un'area ampia e sgombra da ostacoli (punto "B"). Tale zona è tuttavia nota per essere particolarmente soggetta a riflessi e disturbi luminosi, causati da finestre e superfici riflettenti di varia natura.

- Più volte viene inquadrata la gabbia (spostandosi verso "C"), oggetto particolarmente critico per il solo laser da riconoscere nel modo corretto. In altre occasioni lungo il percorso vengono inquadrati dettagli riconoscibili solo dall'uno o dall'altro sensore, per essere certi che la mappa tragga effettivamente vantaggio dall'uso di entrambi i sensori.

- Il robot percorre quindi diversi passaggi sempre più stretti, spesso delimitati da tavoli e sedie, il più stretto dei quali è corrispondente all'angolo in basso a destra della "gabbia" (punto "D"). Ciò è utile per capire se i sensori sono in grado di rappresentare, e con quale qualità, gli ostacoli molto vicini e situati ai limiti dei loro FoV.

- Il robot finisce il test completando un percorso circolare attorno alla gabbia (fermandosi in "E"). La ricostruzione di percorsi chiusi è infatti sovente usata come benchmark

della bontà della mappatura, in quanto la chiusura di un loop è una delle operazioni più critiche per GMapping.

Affinché questa venga effettuata in modo corretto è necessario un buon livello di precisione dei dati in input ed una corretta taratura dei parametri interessati.

Un ottimo esempio di bag ottenuta seguendo questi dettami è stato utilizzato per ottenere la comparativa precedentemente mostrata in Figura 4.13.

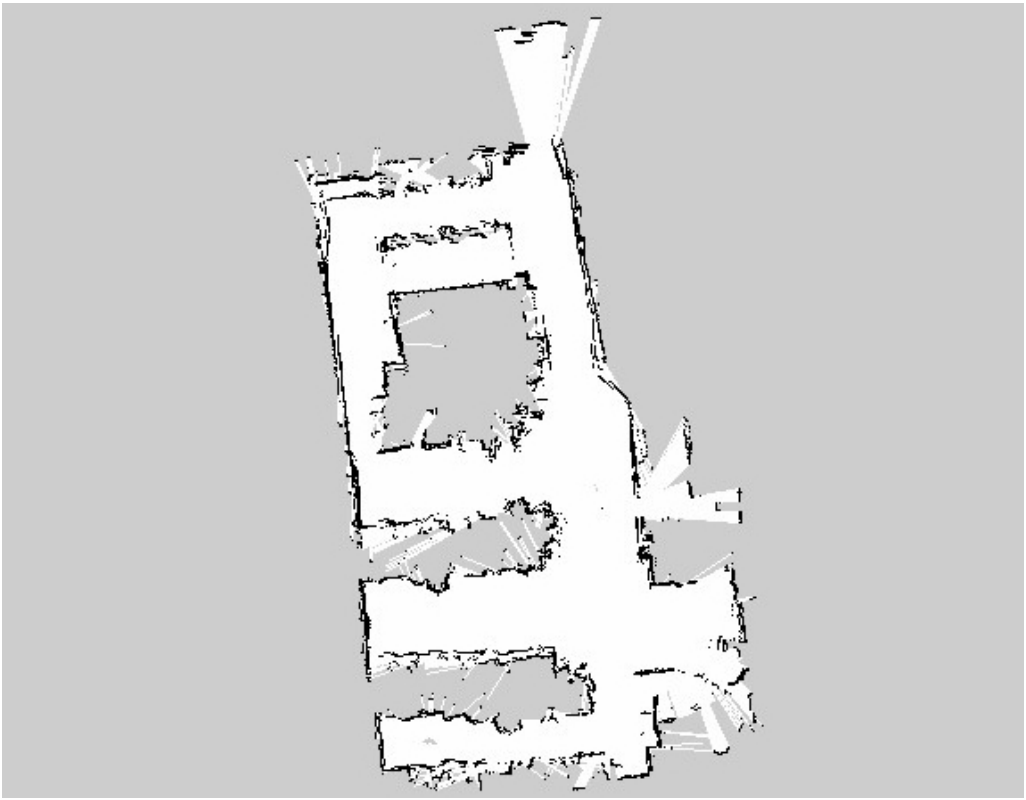


Figura 4.18: Mappa completa dell'aula CAD.

Figura 4.18 e Figura 4.19 raffigurano le mappe complete ottenute per due ambienti di test, ovvero la (già nota) aula CAD e la più piccola aula LAB.

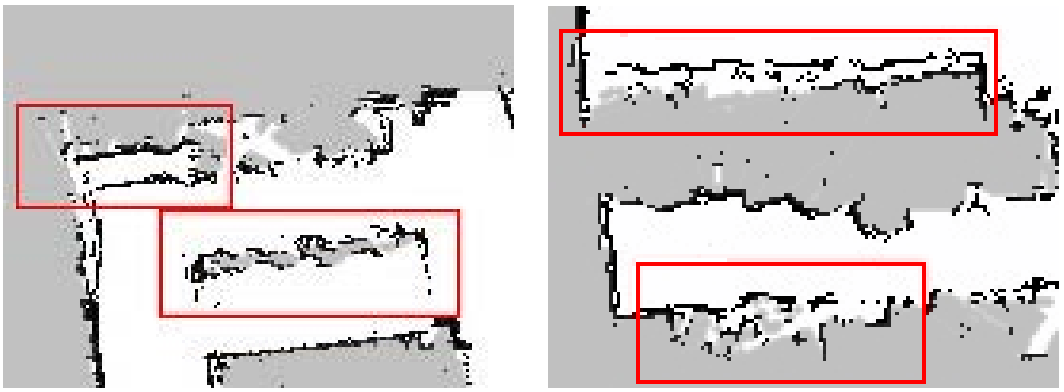
In tali mappe, ottenute alla fine dell'intero processo implementativo, è possibile rilevare i diversi punti di forza e debolezze dell'approccio scelto.

Ad esempio, in Figura 4.20 è possibile notare i già citati difetti di ricostruzione che si presentano quando il robot passa eccessivamente vicino ai tavoli. In particolare si nota che, mentre per il difetto evidenziato in alto a sinistra una successiva acquisizione dell'area da una maggiore distanza ha permesso di porre rimedio, per la ricostruzione del tavolo subito sotto non è stato possibile fare nulla, a causa della presenza ravvicinata della gabbia.

Nella medesima mappa è possibile anche notare come il robot sia stato in grado di rappresentare ostacoli ben più impegnativi dei semplici tavoli.

In Figura 4.21, ad esempio, è evidenziato come nella mappa compaia una barriera in corrispondenza dei piedistalli, uniti dal nastro, posti a protezione del robot UR10. Qualora ci si fosse affidati all'uso del solo laser, di tale struttura sarebbe stata trasposta nella mappa solo la parte inferiore, ovvero la sezione orizzontale dei singoli piedistalli.

Figura 4.19: Mappa completa dell'aula LAB.



(a) (b)

Figura 4.20: Dettagli della mappa dell'aula CAD, con evidenziati i difetti che emergono qualora si passi troppo vicino ai tavoli.

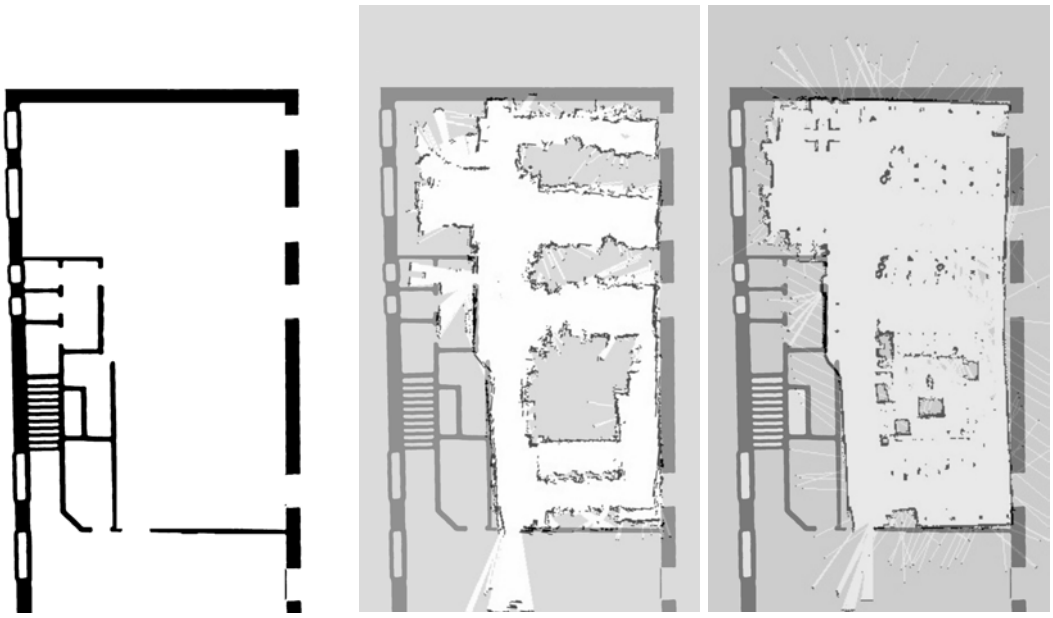
Per concludere in Figura 4.22 viene presentato un confronto fotografico tra la planimetria dell'aula CAD (Figura 4.22a) e la mappa di Figura 4.19.

Come dimostra Figura 4.22b, l'utilizzo del sensore Kinect in abbinata al sensore LRF non va a pregiudicare l'affidabilità con cui è ricostruito l'ambiente. Le proporzioni tra le varie parti della stanza sono infatti corrette, e le distorsioni sono tutte di lieve entità, non tali cioè da pregiudicare il buon andamento delle operazioni per le quali è stato progettato il robot.

Come termine di paragone, in Figura 4.22c è presentata la sovrapposizione tra la planimetria e Figura 4.1.

(a) (b)

Figura 4.21: Esempio della capacità di ricostruire ostacoli di difficile individuazione per il solo laser.



(a) Planimetria. (b) Mappa finale. (c) Mappa laser.

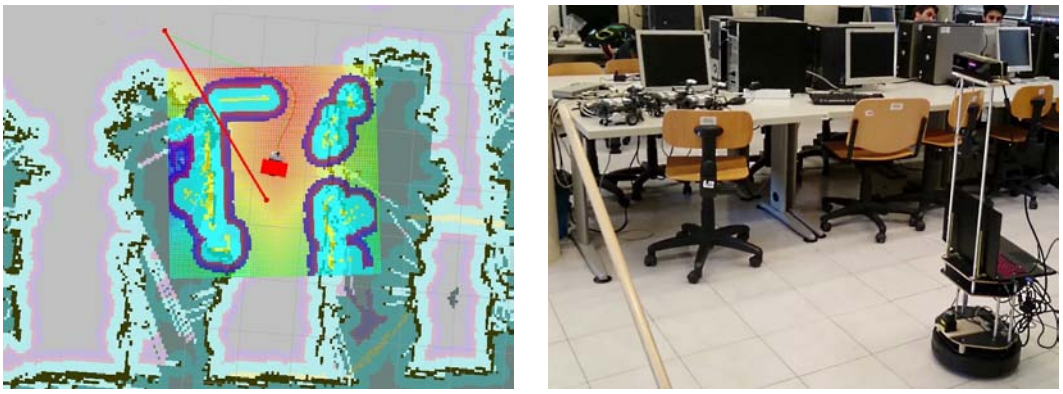
Figura 4.22: Test di sovrapposizione mappa - planimetria.

4.3.2 Navigazione

Diversi test sono stati quindi condotti, usando le mappe appena presentate, per essere certi che ostacoli di diversa natura e forma fossero correttamente individuati anche in fase di navigazione.

Figura 4.23 illustra una situazione simile a quella di Figura 4.21, presa da un test specifico condotto durante la navigazione. Al robot è stato dato il comando di dirigersi verso un goal (in alto a sinistra nell'immagine) e quindi ritornare verso il punto di partenza; il percorso, nella mappa nota, non presenta ostacoli. Nell'ambiente reale tuttavia è stato posto un manico di scopa, in modo da ostruire parzialmente l'uscita del corridoio, ad altezza intermedia tra laser e sensore Kinect.

Il test è stato pensato per verificare la capacità del robot di individuare un ostacolo dallo spessore ridotto ed evitarlo, nonostante questo a distanza ravvicinata ricada nella "zona d'ombra" tra i due sensori. In simili situazioni infatti vi è il rischio che il pesante filtraggio utilizzato per eliminare i disturbi luminosi vada ad intaccare la capacità di individuare



(a) (b)

Figura 4.23: Test di navigazione in cui un manico di scopa è stato posizionato in modo da ostruire parzialmente il corridoio, al fine di costringere il robot a ricalcolare il plan inizialmente generato al fine di evitarlo.

ostacoli dalle dimensioni ridotte.

Il robot tuttavia ha superato con successo il test, modificando la propria traiettoria per tempo e raggiungendo entrambi i goal.

Ulteriori test sono stati messi in atto per assicurarsi che anche in presenza di ostacoli dinamici il robot sia in grado di evitare collisioni.

Alcuni esperimenti hanno tuttavia messo in luce delle debolezze, per le quali è necessario adottare specifiche misure precauzionali. È già stato citato il fatto che, a causa del ridotto FoV, talvolta il robot si de-localizzi. Ciò avviene soprattutto quando la posizione da cui si avvia la navigazione è situata in un ambiente che presenta numerosi ostacoli dinamici o che è cambiato sensibilmente da quando è stata acquisita la mappa.

Una buona pratica per evitare simili inconvenienti quindi può essere quella di posizionare il robot in un posto ben rappresentato dalla mappa fornitagli, rivolgendolo verso il muro od un preciso ostacolo, indipendentemente dalla direzione del goal che si vuole raggiungere. Così facendo, sarà più probabile che all'avvio il robot acquisisca match tra dati sensoriali e mappa sufficienti ad evitare la perdita della posizione corretta.

Capitolo 5

Rilevamento di persone a terra

In questo capitolo saranno presentati i differenti approcci sperimentati per l'individuazione delle persone a terra. Tutti necessitano di una prima fase di preparazione della point cloud e di un sistema di misurazione delle prestazioni ben definito.

In particolare, sono stati inizialmente forniti due algoritmi già implementati e funzionanti. Dopo la messa a punto di alcuni parametri e l'esecuzione di alcuni test approfonditi, sono emersi chiaramente i punti di forza e le debolezze di ognuno. Il primo sfrutta le dimensioni del corpo umano per classificare i cluster presenti nella scena. Il ricorso al clustering euclideo tuttavia lo rende poco efficace in ambienti con molti ostacoli, dove tende a sollevare falsi allarmi. Il secondo è maggiormente preciso nel riconoscimento delle caratteristiche proprie delle persone a terra; tuttavia, non ricorrendo al clustering, necessita di un metodo efficace per aggregare le informazioni ottenute nelle varie parti della cloud esaminata.

L'algoritmo che sarà qui proposto consiste in una fusione di questi due approcci, volta a sfruttare i vantaggi di ognuno. Si andrà quindi a proporre un programma basato su due fasi di classificazione, dove la prima corrisponde a quella del secondo approccio descritto. Il progetto della seconda fase di classificazione costituisce il contributo principale di questo lavoro di tesi, e terrà conto, tra le altre feature, anche delle dimensioni del corpo umano.

Nella sezione "Preparazione della point cloud" vengono quindi discusse le operazioni preliminari effettuate sulla nuvola di punti ottenuta dal sensore Kinect One.

In "Training set e test set per l'allenamento dei classificatori" sono presentati i metodi ed i dati sui quali viene effettuato l'allenamento dei classificatori utilizzati.

In "Validazione delle detection con controllo sulla mappa" si motiva e discute il funzionamento di una tecnica per ridurre il numero di falsi allarmi, implementata in tutti gli algoritmi provati.

In "Rilevamento mediante bounding box orientata" viene descritto il primo metodo sperimentato, basato sulle dimensioni del corpo umano.

In "Rilevamento mediante feature veloci" viene discusso il secondo metodo provato, basato sull'utilizzo di numerosi descrittori a ridotto impatto sulle prestazioni.

In "Rilevamento mediante doppio classificatore" vengono presentati i metodi ed i procedimenti che hanno portato all'algoritmo finale, basato su due classificatori che elaborano i dati in sequenza.

In "Sistema di misurazione delle prestazioni" sono spiegati l'apparato di test e gli indici utilizzati per misurare le performance degli algoritmi discussi.

Infine in "Risultati sperimentali" sono discusse e confrontate le prestazioni degli algoritmi visti in questo capitolo.

5.1 Preparazione della point cloud

Per gestire le operazioni di preparazione della point cloud è stata realizzata una libreria apposita, chiamata `CloudFilterAndScan`. Le operazioni che è necessario svolgere prima di procedere alla divisione in cluster sono, nell'ordine, le seguenti:

1. **Selezione di un'area di interesse nella nuvola di punti.** Analogamente a quanto fatto nel capitolo 4, la prima operazione consiste nel ridurre il numero di punti sui quali si andrà ad operare. In questo caso, la regione di interesse è configurata in modo differente rispetto a quella utilizzata per la creazione della mappa. Questa,

infatti, deve comprendere il suolo, dove sono situate le eventuali persone da rilevare. Al contrario, non è necessario includere zone dove non è possibile trovare persone cadute; da tale osservazione consegue la scelta di tagliare l'altezza massima della nuvola a 0.5 m al di sotto del sensore. Altri punti non di interesse sono quelli situati a meno di 1.6 m dal sensore, distanza alla quale il FoV della Kinect incontra il terreno. Per quanto riguarda il range massimo sfruttato per l'individuazione di persone a terra, inizialmente si sono scelti 3 m; successivamente si è deciso di estendere il range di rilevamento a 5 m, sfruttando a pieno il range della Kinect.

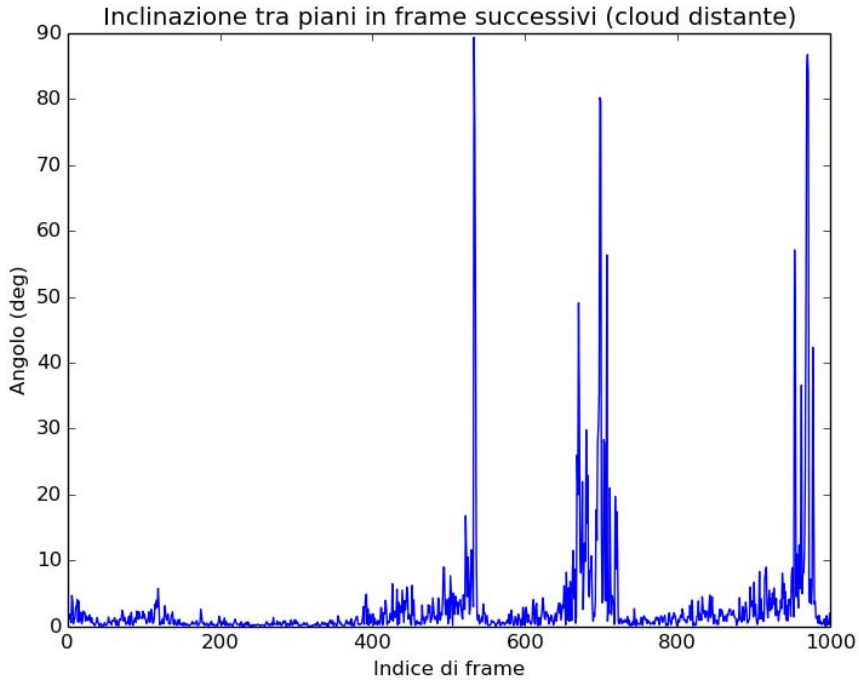
2. **Sottocampionamento della point cloud.** Anche in questo caso, il sottocampionamento viene effettuato con le stesse modalità viste nel capitolo precedente, ovvero utilizzando un filtro di tipo `VoxelFilter`. La `leaf_size` scelta in questo caso è 0.06 m.

3. **Rimozione del terreno.** Tale operazione è di importanza cruciale al fine della rilevazione di persone cadute. Pertanto, una tecnica di rimozione grossolana, basata sul filtraggio a soglia del terreno ad un'altezza corrispondente approssimativamente all'altezza della Kinect da terra, non è una scelta praticabile. Si è quindi optato per l'utilizzo di una tecnica di plane segmentation, basata sul metodo RANSAC [45]. Questa tecnica permette di ricavare i coefficienti che descrivono il piano, presente nella point cloud, che descrive nel modo migliore il maggior numero di punti (detti "inliers") selezionati in modo casuale nella point cloud. Le operazioni necessarie all'ottenimento di tali coefficienti sono svolte mediante un oggetto della classe `SACSegmentation`, appartenente alla repository ufficiale Pointclouds.org. Questo procedimento permette di ottenere una maggiore precisione al momento della rimozione del terreno, grazie anche al fatto che la rimozione è più robusta a beccheggio e rollio del robot in movimento. È importante infatti che si mantenga il più possibile inalterata l'informazione sulle persone e gli oggetti a contatto col terreno per ottenere migliori performance in fase di classificazione.

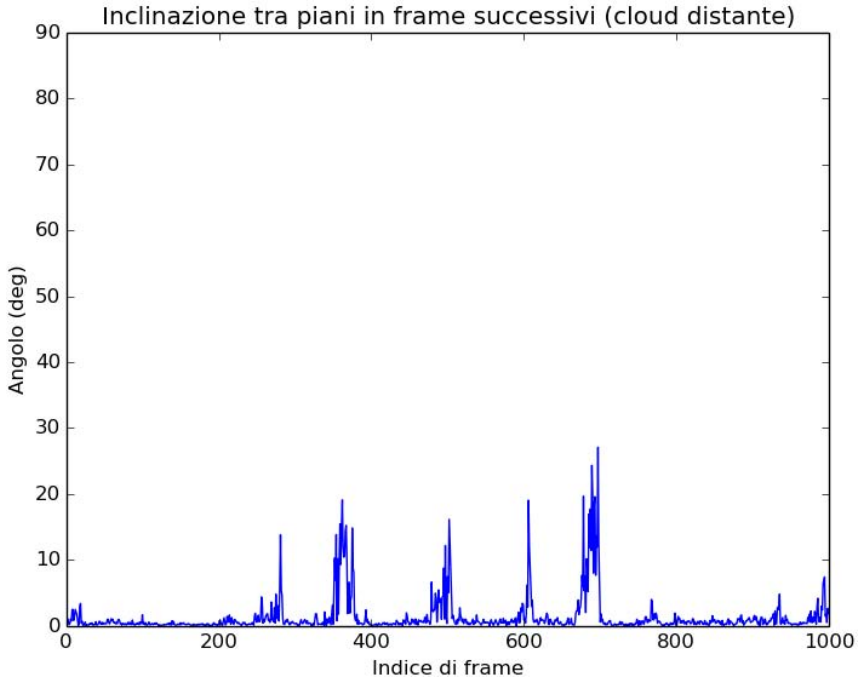
4. **Filtraggio di rumore ed outliers.** Questa fase serve per ridurre il più possibile i disturbi derivanti dal rumore luminoso, che possono influire sulla corretta identificazione dei cluster da parte del classificatore. Tuttavia la necessità di eseguire l'algoritmo in tempo reale limita le scelte in ambito di filtri, notoriamente influenti negativamente sul carico computazionale. Si è pertanto deciso di limitarsi all'utilizzo di un unico passaggio di filtraggio con `StatisticalOutlierRemoval` (con parametri `meanK` = 50 e `stdDevMulThresh` = 0.3).

Le operazioni di rimozione del terreno meritano una trattazione approfondita, in quanto la semplice applicazione della tecnica sopra descritta alla point cloud sottocampionata non è sufficiente a garantire la corretta individuazione del pavimento in ogni frame. L'algoritmo è infatti progettato per individuare il piano che include il maggior numero di punti della scena; è frequente tuttavia che porzioni di muro o di mobilio con superfici piane siano individuate al posto del pavimento.

Per evitare tale evenienza, la rimozione del terreno viene effettuata in due fasi. La prima, denominata "rough ground segmentation", consiste nella selezione di una regione di punti ancora più restrittiva, consistente in una "fascia" dove ci si aspetta che idealmente sia contenuto il pavimento. Questa è una regione di interesse rettangolare, di altezza 0.3 m e non limitata in profondità e larghezza, posta 1.16 m sotto al sensore. Così facendo viene ridotta drasticamente la possibilità che siano individuati piani verticali o piani appartenenti ad oggetti bassi come tavolini e scatoloni al posto del pavimento. La seconda fase, denominata "planar segmentation", applica la tecnica RANSAC per ottenere i coefficienti del piano che meglio descrive il terreno, quindi individua tutti i punti entro una



(a)



(b)

Figura 5.1: Effetto dell'utilizzo del parametro `SACMODEL_PERPENDICULAR_PLANE` sui coefficienti del pavimento rilevato durante l'esecuzione della bag cad_1. I grafici confrontano l'inclinazione tra piani rilevati in frame successivi nella porzione di cloud più distante dal robot. In (a) il parametro non viene utilizzato, in (b) sì.

data distanza da tale piano (detta "ground tolerance") e li rimuove. La ground tolerance scelta è generalmente un terzo dello spessore della regione di interesse selezionata in fase di rough ground segmentation.

La necessità di estendere il range di operatività del rilevamento di persone a terra dagli originali 3 m a 5 m ha sollevato alcuni importanti problemi riguardanti la corretta

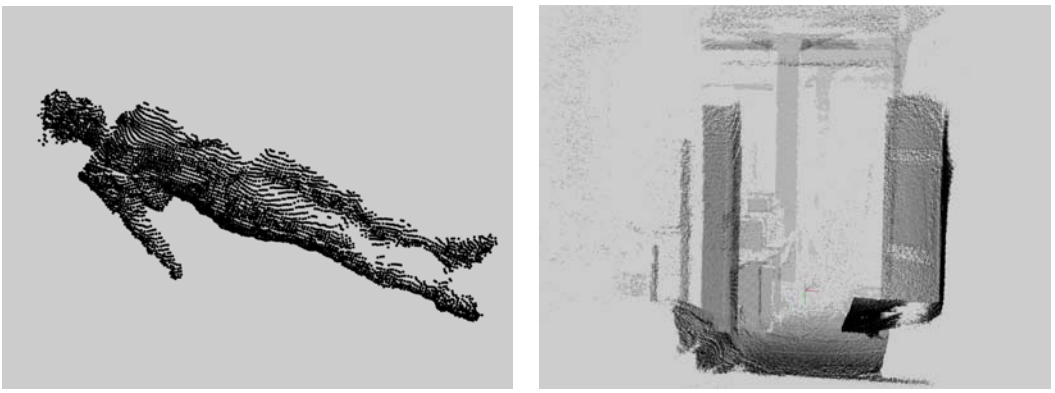
rimozione del piano. Spesso infatti le persone oltre i 3 m non sono individuate poiché anch'esse rimosse dalla fase di planar segmentation. La riduzione della ground tolerance tuttavia causa la permanenza di disturbi luminosi tanto accentuati da essere identificati come persone, nonostante il successivo passaggio di filtraggio.

Si è pertanto deciso di operare la rimozione del pavimento dividendo la point cloud in due zone, a seconda della distanza dal sensore. La tecnica descritta precedentemente è quindi applicata alle due porzioni della nuvola in modo separato, prima di riunificare i risultati. Tale procedimento permette di adottare parametri differenti quanto a ground tolerance e spessore della rough ground segmentation. Come soglia per la divisione della point cloud si è scelta la distanza di 3 m. I risultati di tale scelta hanno permesso un sensibile miglioramento delle prestazioni di rilevamento, raggiungendo quindi l'obiettivo di monitorare aree più ampie.

Successivi test hanno evidenziato come il piano individuato non sempre corrisponda al pavimento, in quanto i coefficienti restituiti descrivono piani fortemente inclinati o comunque inesatti. Figura 5.1a rappresenta l'inclinazione tra i piani trovati in frame successivi nella porzione di cloud più distante dal robot. Come si può notare dai numerosi picchi, in diverse occasioni il piano individuato arriva ad avere un'inclinazione prossima ai 90° rispetto al precedente. In alcune scene in cui la visuale del pavimento è praticamente occlusa (situazioni in cui il robot è eccessivamente vicino a dei tavoli o ad una parete, ad esempio) RANSAC restituisce comunque dei coefficienti che individuano piani la cui posa è quasi sempre incompatibile con quella del suolo. Per risolvere tale problema si è ricorsi ad un parametro dell'oggetto `SACSegmentation`, ovvero `SACMODEL_PERPENDICULAR_PLANE`; questo permette di specificare un vettore al quale il piano dovrà essere il più possibile perpendicolare, oltre che un angolo corrispondente alla massima tolleranza per l'inclinazione di tal vettore rispetto alla normale al piano restituito. Figura 5.1b dimostra gli effetti di tale espediente, dove è stata impostata una tolleranza alla massima inclinazione del piano pari a 15°.

L'efficacia delle soluzioni individuate ha fatto propendere per il loro utilizzo anche nell'algoritmo finale.

5.2 Training set e test set per l'allenamento dei classificatori



(a) (b)

Figura 5.2: Esempi di campione positivo (a sinistra) e campione negativo (a destra) del dataset utilizzato per l'allenamento dei classificatori.

Training e test set utilizzati per l'allenamento dei classificatori menzionati nel corso di questo capitolo sono ricavati da un insieme di point cloud predeterminato e disponibile allo IAS-Lab, salvato in forma di file pcd. Le point cloud catalogate come campioni positivi

contengono persone segmentate manualmente (come visibile in Figura 5.2a), mentre quelle appartenenti all'altra classe consistono in riprese dell'ambiente, senza persone a terra (di cui è fornito un esempio in Figura 5.2b).

Come training set sono utilizzate point cloud ottenute da riprese dell'aula LAB. Questo complessivamente comprende in totale 718 nuvole di punti distinte, delle quali 240 sono campioni positivi. Come test set è utilizzato un insieme di cloud con esempi positivi e negativi ottenuti nell'aula CAD. Il test set è composto da 465 elementi, di cui 121 classificati come positivi.

Ai fini dell'allenamento del classificatore, sia training che test set devono essere processati mediante un opportuno programma che calcoli gli attributi di interesse di ogni campione. Questo generalmente consiste in un ciclo che carica una per volta le point cloud e svolge su di esse le stesse operazioni che saranno poi svolte in tempo reale durante le procedure di rilevamento di persone a terra. Ottenuti i dati utili ai fini della classificazione si provvede a salvarli ordinatamente in una riga di un file CSV. Per convenzione, il primo elemento della riga corrispondente ad ogni point cloud rappresenta la classe: 1 se la cloud contiene una persona a terra, 0 se non la contiene. L'informazione sulla classe è ricavata dal nome stesso del file pcd caricato.

I due file CSV così ottenuti sono quindi caricati da un altro programma, che li passa come parametri all'oggetto rappresentante il classificatore di interesse per ottenere il modello e valutarne le prestazioni.

I classificatori utilizzati ai fini di questa tesi sono tutti reperibili nelle librerie specifiche per il machine learning di OpenCV. In particolare, per quanto riguarda l'allenamento di Support Vector Machine (SVM), implementate utilizzando oggetti di tipo CvSVM[1], si è sempre ricorsi all'ottimizzazione automatica dei parametri. Questa tecnica, richiamabile mediante il metodo `train_auto`, sceglie i valori dei parametri che minimizzano la stima dell'errore sul test set ottenuto mediante cross-validazione.

Nella parte finale del progetto, dovendo migliorare le prestazioni del classificatore basato su feature veloci, si è deciso di ampliare il training set introducendo delle sequenze del dataset pubblico NYUV2 per la segmentazione semantica[2] [46]. I benefici derivanti da tale mossa si sono manifestati con una maggiore robustezza dell'algoritmo al variare delle condizioni ambientali, testimoniata da migliori prestazioni nelle simulazioni sul test set.

5.3 Validazione delle detection con controllo sulla mappa

Tutti gli algoritmi presentati sono stati forniti di un metodo comune, denominato `checkObstaclesOnMap`. Questo si occupa di validare ogni rilevamento di persona a terra mediante il confronto con la mappa dell'ambiente.

Precisamente, ad ogni avvio del programma di rilevamento di persone a terra, è caricata in memoria la mappa dell'ambiente, resa disponibile dal nodo `map_server` in forma di occupancy grid. Come visto nel capitolo 4, per consentire la navigazione del robot il nodo è avviato automaticamente; nel caso delle simulazioni di test discusse in questo capitolo, invece, è stato avviato manualmente.

Ogniqualvolta l'algoritmo di rilevamento individua una persona a terra, invoca `checkObstaclesOnMap` passando come parametro le coordinate del centroide della presunta persona caduta, già riferite al frame */map*. Tale centroide viene utilizzato per risalire alla corrispondente cella della matrice rappresentante la occupancy grid. Il valore letto in tale cella è compreso tra 0 (cella libera) e 100 (cella occupata da un ostacolo), con i valori intermedi proporzionali alla probabilità che questa sia occupata. Il valore speciale

[1] Per una spiegazione dettagliata dei parametri e delle caratteristiche di tale classe si rimanda a `http://docs.opencv.org/2.4/modules/ml/doc/support_vector_machines.html`

[2] Link: `http://cs.nyu.edu/~silberman/datasets/nyu_depth_v2.html` .

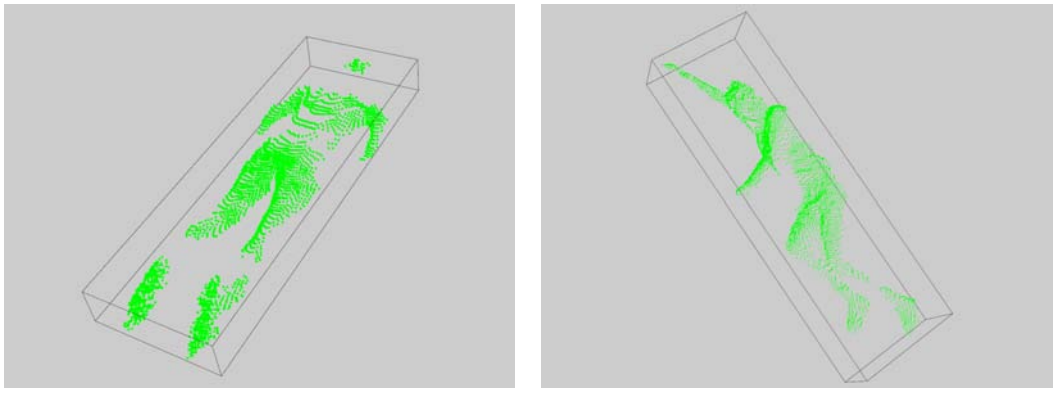
(a) (b)

Figura 5.3: A sinistra, il tronco situato in aula CAD. A destra è evidenziata la sua rappresentazione sulla mappa.

-1 viene invece utilizzato per indicare le celle sconosciute, ovvero quelle non raggiunte dai sensori del robot in fase di mappatura; tipicamente è questo il caso di aree oltre i confini della stanza o al centro di ostacoli aventi un'estesa sezione orizzontale, come gli armadi. Mediante un controllo a soglia sul valore di tale cella, il metodo restituisce un valore booleano che concede il via libera finale alla notifica del rilevamento o, in caso di esito negativo, sancisce la sua invalidità. Il controllo, nello specifico, fornisce esito negativo nei casi in cui l'intero contenuto nella cella è superiore a 30 oppure uguale a -1.

L'utilità di un simile espediente è esaltata dal fatto che spesso numerosi falsi allarmi sono generati da componenti dell'arredamento permanenti, quindi molto probabilmente presenti nella mappa creata dal robot. Un esempio molto rilevante nel caso in esame è quello costituito da un tronco d'albero situato in aula CAD, ritratto in Figura 5.3a. Forma e dimensioni di tale oggetto lo rendono una detection ricorrente in tutti gli algoritmi qui descritti (anche se con incidenza differente a seconda dell'approccio scelto). Il fatto che questo sia presente nella mappa dell'aula (è evidenziato in Figura 5.3b) ha permesso di evitare numerosi rilevamenti ad esso associati, nonostante alcuni casi isolati ancora persistano a causa di una non sempre precisa localizzazione del centroide del tronco.

5.4 Rilevamento mediante bounding box orientata



(a) (b)

Figura 5.4: Esempi di point cloud rappresentanti persone a terra racchiuse nei rispettivi bounding box orientati.

La prima e più semplice tecnica utilizzata per effettuare il rilevamento di persone a terra è basata sulle dimensioni del corpo di un essere umano adulto. Questa tecnica, già testata e funzionante al momento dell'inizio della tesi, è implementata mediante un nodo ROS chiamato `fall_detection_node`, facente parte del pacchetto `orobot_fall_detection`, reperibile all'interno della repository `orobot`.

La prima operazione effettuata sulla point cloud in uscita dalla fase di preparazione è quella della divisione in cluster. Per effettuare tale operazione ci si affida ad un oggetto della classe `EuclideanClusterExtraction` disponibile nella repository ufficiale Pointclouds.org. Tramite questo è possibile effettuare il clustering euclideo della nuvola di punti, ovvero raggruppare i punti visibili in gruppi utilizzando come metrica la distanza geometrica. Di ogni cluster è quindi calcolata la bounding box orientata. Questa è ottenuta effettuando l'analisi delle componenti principali del cluster ("Principal Component Analysis" o PCA) [47], una tecnica utile ad individuare la principale direzione lungo la quale si sviluppa il cluster. Questa richiede che siano individuati il centroide del cluster e la matrice di covarianza dei punti del cluster rispetto ad esso. Sono quindi calcolati gli autovalori associati a tale matrice. Tra questi, quello di modulo maggiore corrisponde alla direzione lungo la quale la minima bounding box che racchiude i punti del cluster ha dimensione maggiore. Analogamente, il secondo autovalore in ordine decrescente di modulo individua la seconda direzione della bounding box, mentre la terza è perpendicolare al piano individuato dalle prime due. Tali direzioni individuano un sistema di riferimento avente gli assi paralleli o perpendicolari ai lati della bounding box di interesse. Le dimensioni della bounding box orientata sono quindi salvate in ordine crescente in un vettore, e vanno a costituire i tre attributi secondo i quali sarà valutato il cluster per decidere se rappresenta o meno una persona a terra.

Prima di procedere alla classificazione, tuttavia, sono effettuate delle operazioni di filtraggio per eliminare i candidati palesemente negativi. Sono quindi scartati i cluster che non hanno abbastanza punti e quelli i cui punti superano una determinata soglia d'altezza; quest'ultima operazione è utile per evitare di classificare come persona cluster troppo alti rispetto al pavimento, spesso parti di oggetti più alti "tagliati" al momento della selezione della regione di interesse.

Per le operazioni di classificazione ci si affida ad una Random Forest, rappresentata da un'istanza della classe `CvRTrees` della repository ufficiale di OpenCV[3]. Nel file CSV usato per l'allenamento del classificatore vengono salvate le tre dimensioni della bounding box orientata, in ordine crescente.

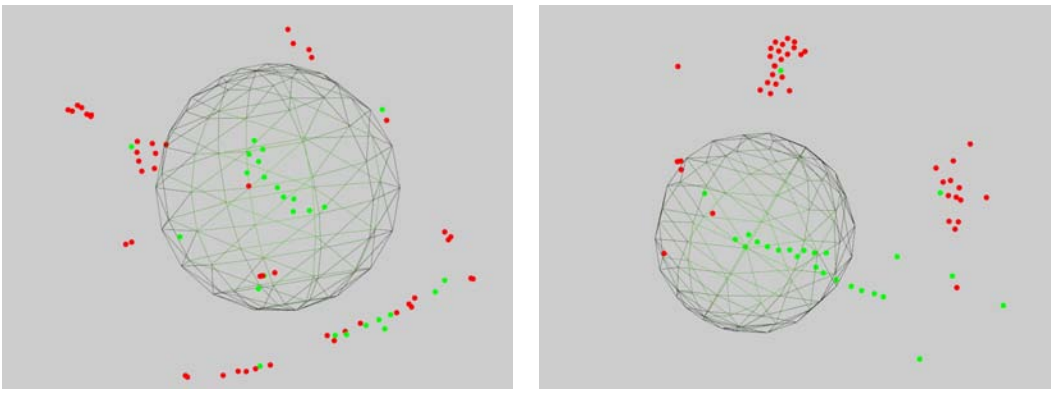
Il modello ottenuto in fase di training viene caricato all'avvio di `fall_detection_node` ed è utilizzato per esaminare i possibili cluster candidati rinvenuti in ogni frame. Qualora vi fosse esito positivo dalla classificazione di un candidato, il centroide del cluster individuato viene pubblicato nel topic `orobot_fall_detection/detections` come messaggio di tipo `geometry_msgs/PointStamped`. La detection è inoltre notificata sullo standard output del programma.

In sintesi, `fall_detection_node` restituisce esito positivo ogniqualvolta rileva, in un frame, un cluster le cui dimensioni sono compatibili con quelle di una persona distesa.

La principali critiche mosse all'algoritmo sono due. La prima riguarda la sua incapacità di distinguere la natura dei cluster che va ad analizzare. Da ciò consegue il fatto che qualsiasi oggetto o gruppo di oggetti disposti a terra aventi una bounding box orientata di dimensioni paragonabili a quelle di una persona distesa saranno classificate positivamente dall'algoritmo. In numerose simulazioni si è verificato che lo stesso rumore luminoso spesso è fonte di falsi positivi. La seconda critica invece è legata all'uso del clustering euclideo, il quale, basandosi su una soglia minima sulla distanza tra i cluster individuati, limita l'efficacia dell'algoritmo in scenari dove sono presenti numerosi ostacoli.

[3]Per una spiegazione dettagliata dei parametri e delle caratteristiche di tale classe si rimanda a `http://docs.opencv.org/2.4/modules/ml/doc/random_trees.html`

5.5 Rilevamento mediante feature veloci



(a) (b)

Figura 5.5: Esempi di rilevamenti ottenuti con il metodo delle feature veloci. I punti contenuti all'interno della sfera (centrata su un centroide verde) soddisfa le condizioni imposte sul numero minimo e massimo di centroidi rispettivamente verdi e rossi.

Per superare le debolezze della tecnica di rilevamento mediante bounding box orientata, si è deciso di sperimentare una tecnica alternativa, che permetta di tener maggiormente in considerazione le caratteristiche morfologiche di una persona distesa per terra. La tecnica, di seguito discussa, è stata fornita già implementata nel nodo `fallen_person_detection`, situato nel pacchetto `dofp_vision_tools` della repository `detection_of_fallen_people`.

Per riuscire a distinguere le persone a terra da oggetti di diversa natura si è sfruttato un procedimento simile a quello descritto da Wolf, Prankl e Vincze in [48, 49] per la segmentazione semantica degli oggetti contenuti in una point cloud. In particolare, l'attenzione è stata rivolta al modo in cui gli autori propongono di segmentare la point cloud mediante una tecnica chiamata "Voxel Cloud Connectivity Segmentation" [50], implementata e disponibile nella repository online di pointclouds.org. Tale procedimento permette di ottenere delle patch, ovvero porzioni di superfici chiamati anche "supervoxel", che hanno la proprietà di adattare i propri confini ai bordi degli oggetti sulla scena. Su ogni supervoxel è poi calcolato un alto numero di descrittori (rispettivamente, 18 e 14 per quanto riguarda i lavori discussi rispettivamente nel primo e nel secondo articolo), tutti accomunati dal ridotto carico computazionale necessario per ottenerli (sono calcolati rispettivamente in 730 ms e 500 ms). I risultati dimostrano come questa tecnica restituisca buoni risultati, riconoscendo e assegnando il corretto label a diversi componenti dell'arredamento di una stanza sui quali un classificatore Random Forest era stato precedentemente allenato.

Il metodo implementato applica quindi tale procedimento alle cloud del training e del test set (dopo le consuete operazioni di preparazione). Sono calcolati tutti i supervoxel della nuvola di punti, quindi per ognuno sono calcolate le seguenti feature veloci:

- Massima altezza del supervoxel rispetto al piano del pavimento
- Minima altezza del supervoxel rispetto al piano del pavimento
- Altezza del centroide del supervoxel rispetto al piano del pavimento
- Media del seno degli angoli delle normali alla superficie della patch rispetto al suolo
- Deviazione standard dei seni degli angoli delle normali rispetto al suolo
- Compattezza

- Planarità
- Linearità
- Densità di punti, ovvero il numero di punti nel supervoxel diviso l'area approssimata della superficie di quest'ultimo

Data la matrice di covarianza associata ai punti del supervoxel, siano $\lambda_0, \lambda_1, \lambda_2$ gli autovalori ad essa associati ordinati per grado crescente. Si ha che la compattezza è espressa da λ_0, la planarità da $(\lambda_1 - \lambda_0)$ e la linearità da $(\lambda_2 - \lambda_1)$.

Come classificatore sono stati provati sia Random Forest che Support Vector Machine. Quest'ultimo è stato preferito in quanto più costante nelle prestazioni al variare degli scenari in cui è stato testato. Tale caratteristica della SVM è stata ulteriormente migliorata in seguito all'ampliamento del training set descritto alla fine della sezione "Training set e test set per l'allenamento dei classificatori".

Dopo aver effettuato la classificazione, in `fallen_person_detection` viene calcolato il centroide di ogni supervoxel, successivamente colorato di verde se il supervoxel è stato classificato come persona, rosso altrimenti. Tutti i centroidi dei supervoxel della scena sono salvati in un'apposita point cloud. Viene quindi avviata una ricerca nella nuvola di punti, dove per ogni centroide verde sono esaminati tutti i vicini entro il raggio di 1 m. Se il conteggio dei centroidi verdi entro tale distanza supera una soglia minima (impostata a 15 elementi) ed il numero di centroidi rossi non supera una soglia massima (impostata a 5 elementi) viene segnalato nello standard output il rilevamento di una persona. Conseguentemente sono pubblicate le coordinate del centroide (sotto forma di messaggio di tipo `geometry_msgs/PointStamped`) esaminato nel topic `oversegmenter/detections`.

In un secondo tempo, si è provato ad aggiungere ulteriori feature veloci all'elenco fatto sopra, sempre secondo quanto riportato negli articoli di riferimento sopra citati. Si è al contempo deciso di non utilizzare più la densità e di usare l'angolo al posto del seno dell'angolo nelle feature relative all'inclinazione delle normali. Di seguito sono riportate le nuove feature veloci:

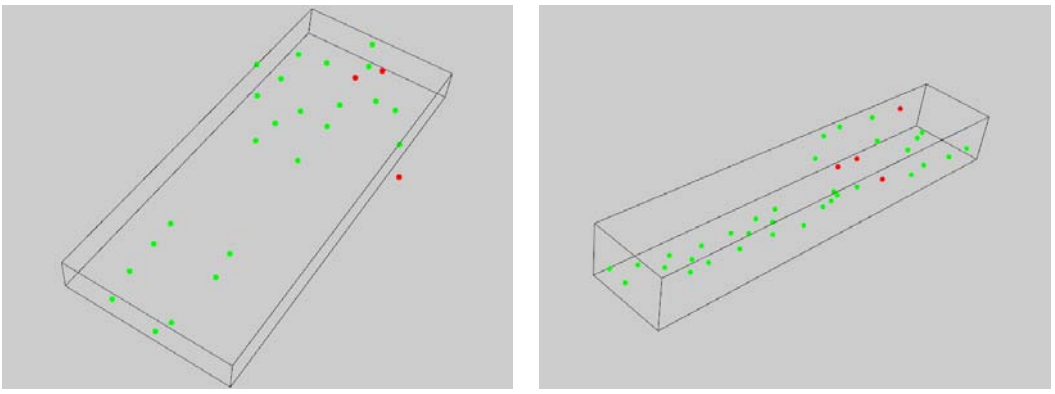
- Larghezza del bounding box che racchiude i punti del supervoxel ($width$)
- Altezza del bounding box che racchiude i punti del supervoxel ($height$)
- Profondità del bounding box che racchiude i punti del supervoxel ($depth$)
- Area della faccia frontale del bounding box (ottenuta come $width * height$)
- Area della faccia superiore del bounding box (ottenuta come $width * depth$)
- Elongazione verticale del bounding box (ottenuta come $height/width$)
- Elongazione orizzontale del bounding box (ottenuta come $depth/width$
- Indice di spessore del bounding box (ottenuto come $height/depth$)

Il secondo modello con le nuove feature si è dimostrato molto più performante del primo, andando a migliorare sensibilmente la capacità del programma di distinguere supervoxel appartenenti alle persone da porzioni dell'ambiente circostante.

Una considerazione riguardante il funzionamento dell'algoritmo è legata al fatto che spesso, nel raggio di 1 m attorno ad un centroide verde, ricadono altri oggetti del circondario, rappresentati da centroidi rossi. Per questo motivo persone troppo vicine a muri o mobili spesso non sono riconosciute, a causa del mancato rispetto della soglia sul massimo numero di esemplari negativi nelle vicinanze. D'altro canto, l'algoritmo si basa sull'analisi

di porzioni di superficie di area ridotta, pertanto spesso vi sono numerosi supervoxel classificati in modo errato disseminati per la scena. Questa evenienza non permette di eliminare la soglia massima di patch rosse nel circondario senza andare ad incrementare il numero di falsi allarmi generati dal programma.

5.6 Rilevamento mediante doppio classificatore



(a) (b)

Figura 5.6: Esempi di point cloud rappresentanti cluster di centroidi di supervoxel classificati mediante il sistema che sfrutta le fast features, racchiusi nei rispettivi bounding box orientati.

Il terzo algoritmo, presentato come il contributo principale di questa tesi, è motivato dalla necessità di migliorare le prestazioni del metodo basato sulle feature veloci. A grandi linee, il metodo qui discusso si propone come una sintesi del funzionamento dei due algoritmi precedentemente descritti. Inizialmente viene elaborata la scena nel modo descritto nel paragrafo "Rilevamento mediante feature veloci", fino ad ottenere due nuvole di centroidi di supervoxel, una contenente quelli classificati come appartenenti ad una persona (verdi), l'altra contenente i rimanenti (rossi). A questo punto, si applica la procedura vista in "Rilevamento mediante bounding box orientata" sulla nuvola di centroidi verdi: si ricavano i cluster, ognuno dei quali viene considerato un possibile "candidato". Di ogni candidato è calcolato il bounding box orientato, dal quale si ricavano le tre dimensioni in modo analogo a quanto visto nel primo metodo presentato. La situazione al termine di questa procedura è rappresentata graficamente in Figura 5.6. Inoltre, si sfrutta l'occasione di conoscere la possibile posa approssimativa della persona nello spazio per operare il conteggio dei centroidi verdi e rossi solamente all'interno del volume del bounding box. Dimensioni e conteggio dei punti saranno quindi fornite in input ad un secondo classificatore, che restituirà il responso finale sulla natura del candidato in esame. Come di consueto, in caso di detection di una persona, il fatto viene segnalato sullo standard output e viene pubblicato il centroide (sempre come messaggio `geometry_msgs/PointStamped`) del cluster giudicato positivo nel topic `oversegmenter/detections`.

I vantaggi di un simile approccio sono tangibili:

- Si va ad eliminare un'importante debolezza del secondo classificatore, che era data dalla scelta di cercare i candidati in uno spazio sferico, non avendo informazioni sulla possibile postura della persona ricercata.
- Utilizzando un classificatore, si usa un procedimento automatico per la gestione dei centroidi verdi e rossi nelle vicinanze di una possibile detection, eliminando la dipendenza da soglie impostate manualmente da un operatore esterno.

- Si reintroduce la classificazione basata sulla forma geometrica dei cluster di punti, permettendo di andare a ridurre ulteriormente i falsi positivi generati da agglomerati di forma arbitraria di centroidi verdi.

Quanto ai classificatori utilizzati, ci si è affidati nuovamente a Random Forest e Support Vector Machine. I primi hanno tuttavia dimostrato una tendenza eccessiva all'overfitting dei dati del training set, quindi ancora una volta è stato preferito l'uso di SVM.

Altri parametri di importanza centrale per il corretto funzionamento dell'algoritmo sono quelli utilizzati per operare il clustering della nuvola di centroidi. È necessario infatti che i cluster ottenuti non siano rappresentazioni parziali di una persona, bensì la racchiudano tutta al proprio interno. Al contempo, è fondamentale che le persone siano ben separate dagli oggetti vicini, come muri ed arredamento. Il fallimento in uno di questi due scopi può portare alla mancata rilevazione di una caduta o alla generazioni di falsi positivi. Per tale motivo si è deciso di introdurre un passaggio di filtraggio prima di operare il clustering della nuvola di punti verdi. Tale filtraggio consiste nella rimozione, con un filtro di tipo Radius Outlier Removal, dei punti che non hanno almeno 5 vicini entro il raggio di 0.5 m.

Successivamente, la volontà di migliorare ulteriormente le prestazioni ha portato a variare il numero e la natura delle feature utilizzate per la seconda classificazione. Di seguito sono brevemente riassunti gli esperimenti effettuati:

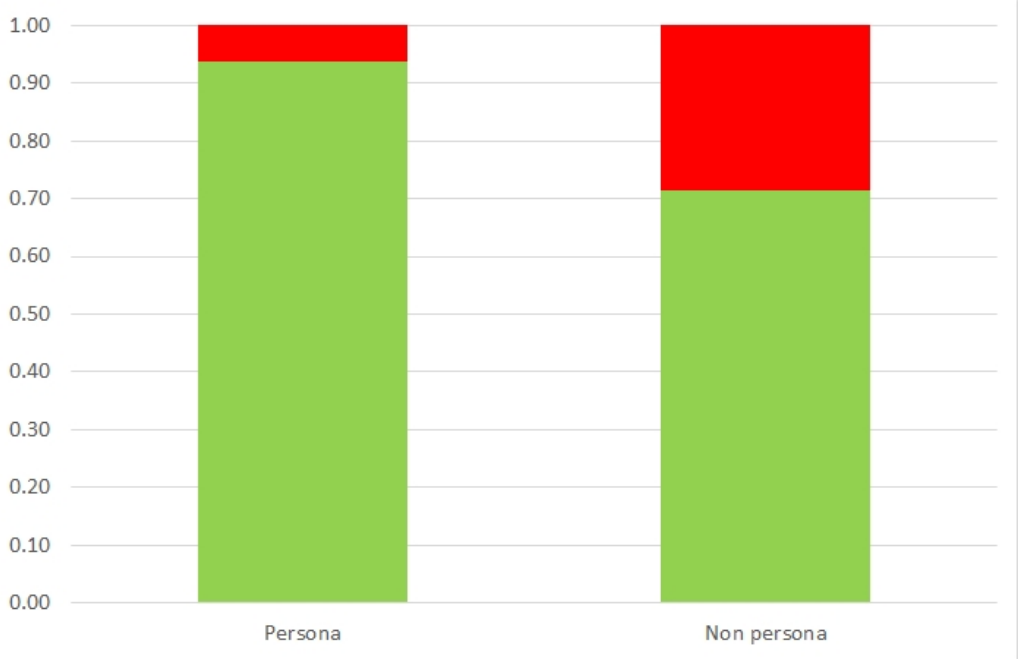


Figura 5.7: Grafico che riporta la proporzione tra centroidi verdi e rossi (in media) nei cluster classificati come persona e in quelli classificati come non persona. Dati tratti dal file CSV usato per allenare il classificatore.

- **Percentuale di centroidi verdi.** Nell'ottica di migliorare le prestazioni della classificazione mediante SVM, si è cercato di sostituire feature rappresentanti conteggi di elementi con più opportuni attributi con valori normalizzati. In questo caso si è quindi deciso di sostituire il conteggio dei centroidi rossi con il rapporto tra centroidi verdi e punti totali (rossi e verdi) nella bounding box. In seguito a tale contromisura gli indici di prestazioni hanno registrato un contenuto ma sensibile miglioramento. Da ciò consegue la scelta di adottare questa selezione di features come nuova base di partenza per sperimentare le soluzioni discusse nei successivi punti.

La proporzione media tra centroidi verdi e centroidi rossi all'interno dei bounding box orientati dei cluster appartenenti alle due possibili classi è rappresentata in Figura 5.7.

- **Selezione a soglia della detection.** Il risultato restituito da un modello di SVM implementato mediante le librerie di OpenCV permette di scegliere se avere come output della classificazione la classe di appartenenza del campione fornito oppure la distanza di quest'ultimo dall'iperpiano che separa le due classi. Inizialmente per il secondo classificatore è stata sfruttata la prima modalità, la quale decreta l'appartenenza ad una classe o ad un'altra in base al segno della distanza dall'iperpiano. In seguito si è preferito poter intervenire direttamente sull'esito della classificazione mediante una soglia che permette di scartare i campioni che, nonostante siano giudicati persone cadute, presentano una distanza ridotta dalla soglia di decisione. Così facendo si può influire sul numero di falsi positivi e falsi negativi rilevati, andando a bilanciare il comportamento dell'algoritmo secondo le proprie necessità. Nel caso in esame, dove è preferibile evitare l'insorgere di falsi allarmi anche al prezzo di alcune mancate detection in qualche frame, si è preferito filtrare tutti i campioni la cui distanza dall'iperpiano in modulo è inferiore a 0.5. Una soluzione più bilanciata e comunque poco prona a generare falsi allarmi si può ottenere abbassando tale soglia a 0.25.

- **Preselezione a soglia dei centroidi.** Tale tecnica riguarda il medesimo procedimento descritto al punto precedente, ma applicato alla SVM del primo classificatore. Andare a ridurre il numero di centroidi verdi salvati nella nuvola, mantenendo solo i più affidabili, può essere visto come un ulteriore filtraggio oltre al già citato Radius Outlier Removal. Tale scelta tuttavia non ha prodotto i risultati sperati, in quanto la riduzione del numero di supervoxel classificati come appartenenti ad una persona interessa tanto la superficie di persone reali quanto il resto dell'ambiente. Ciò nonostante tale esperimento ha aperto le porte ad una serie di altri più efficaci metodi basati sulla distanza restituita dal primo classificatore, i quali sono discussi nei punti seguenti.

- **Istogrammma di affidabilità.** L'informazione riguardante la distanza del campione dall'iperpiano è un'importante indicatore dell'affidabilità della classificazione. Per quanto riguarda la classificazione dei supervoxel, tuttavia, la scelta di una soglia fissa per scartare i centroidi non giudicati sufficientemente affidabili si è rivelata una mossa sconveniente. Per tal motivo si è escogitato un espediente per far sì che l'affidabilità dei singoli centroidi che compongono i cluster abbia un peso sull'esito della seconda classificazione. Sono stati scelti quattro intervalli in cui dividere gli esiti positivi della classificazione, in base al modulo della distanza restituita:

  1. Esiti compresi in $[0, 0.25)$
  2. Esiti compresi in $[0.25, 0.5)$
  3. Esiti compresi in $[0.5, 1)$
  4. Esiti compresi in $[1, \infty)$

  All'atto della creazione della cloud di centroidi, ad ogni centroide è associata l'informazione sull'intervallo di appartenenza. Dopo le operazioni di clustering, mediante un loop sui punti appartenenti ad ogni cluster viene generato l'istogramma relativo, contenente il conteggio dei punti appartenenti a ciascun intervallo.

  Sono quindi state aggiunte quattro feature, corrispondenti ai quattro intervalli, in coda alle cinque sulle quali è basato il modello per la seconda classificazione.

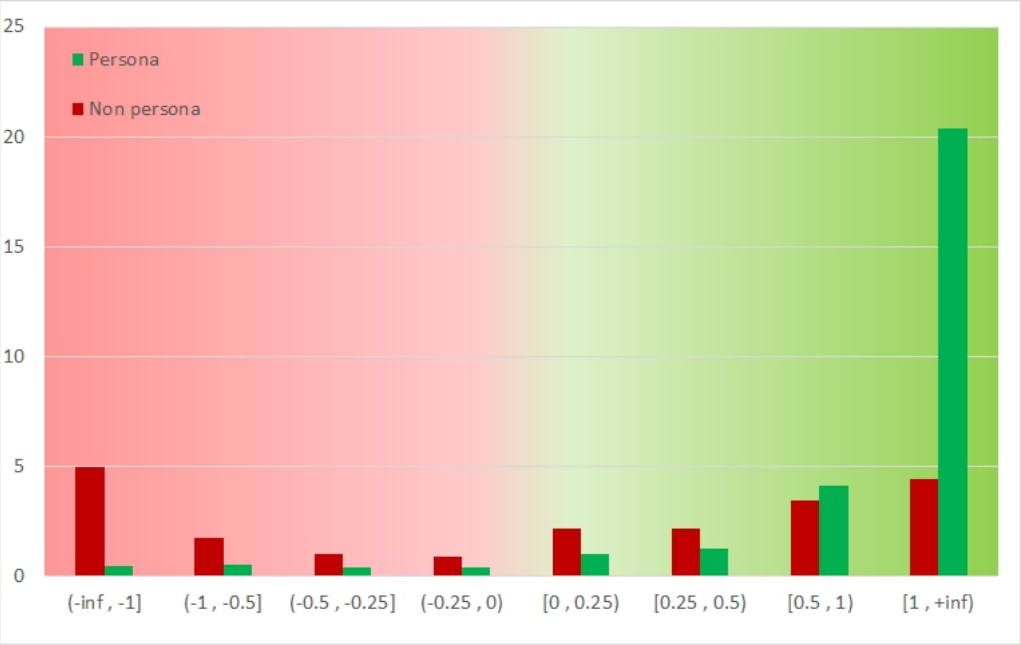


Figura 5.8: Istogramma completo, riportante i valori medi dei centroidi in ogni intervallo di distanza dall'iperpiano di separazione tra le due classi della SVM utilizzata per la prima fase di classificazione. I dati sono suddivisi tra i centroidi appartenenti a cluster rappresentanti persone a terra (verde) e cluster negativi (rosso).

Questo espediente ha apportato un ulteriore miglioramento alle prestazioni dell'algoritmo.

- **Istogramma di affidabilità normalizzato.** Per le medesime motivazioni descritte al secondo punto, si è deciso di cambiare la natura dei quattro attributi corrispondenti agli intervalli dell'istogramma di affidabilità. Al posto di riportare il conteggio di punti in ogni intervallo viene riportato tale conteggio diviso per il numero di punti totali nel cluster.

  Sorprendentemente tale metodo non si è dimostrato una mossa migliorativa, in quanto i risultati ottenuti in tutti i test effettuati sono peggiori di quelli fatti registrare dalla precedente versione.

- **Istogramma di affidabilità completo.** Un'altra variante dell'istogramma di affidabilità testata consiste nel considerare anche i centroidi rossi che ricadono all'interno del bounding box orientato del cluster. Per fare ciò l'istogramma è ampliato con quattro ulteriori intervalli, simmetrici rispetto a quelli noti rispetto allo zero. Figura 5.8 riporta i dati ottenuti dal file CSV utilizzato per il training del classificatore. Ogni barra rappresenta il numero medio di centroidi, divisi in base alla classe (persona o non persona) del cluster a cui appartengono, che ricade in ogni intervallo.

  Anche in questo caso, tuttavia, i seppur buoni risultati non si sono dimostrati all'altezza di quelli registrati con l'istogramma a quattro intervalli.

La scelta finale degli attributi utilizzati per operare la seconda classificazione è quindi ricaduta su: dimensioni ordinate della bounding box del cluster, numero di punti nel cluster (corrispondente al numero di centroidi verdi nella bounding box), percentuale di centroidi verdi sul totale dei punti compresi nella bounding box, conteggio dei punti nei quattro intervalli dell'istogramma, ordinati dal più vicino al più lontano dall'iperpiano.

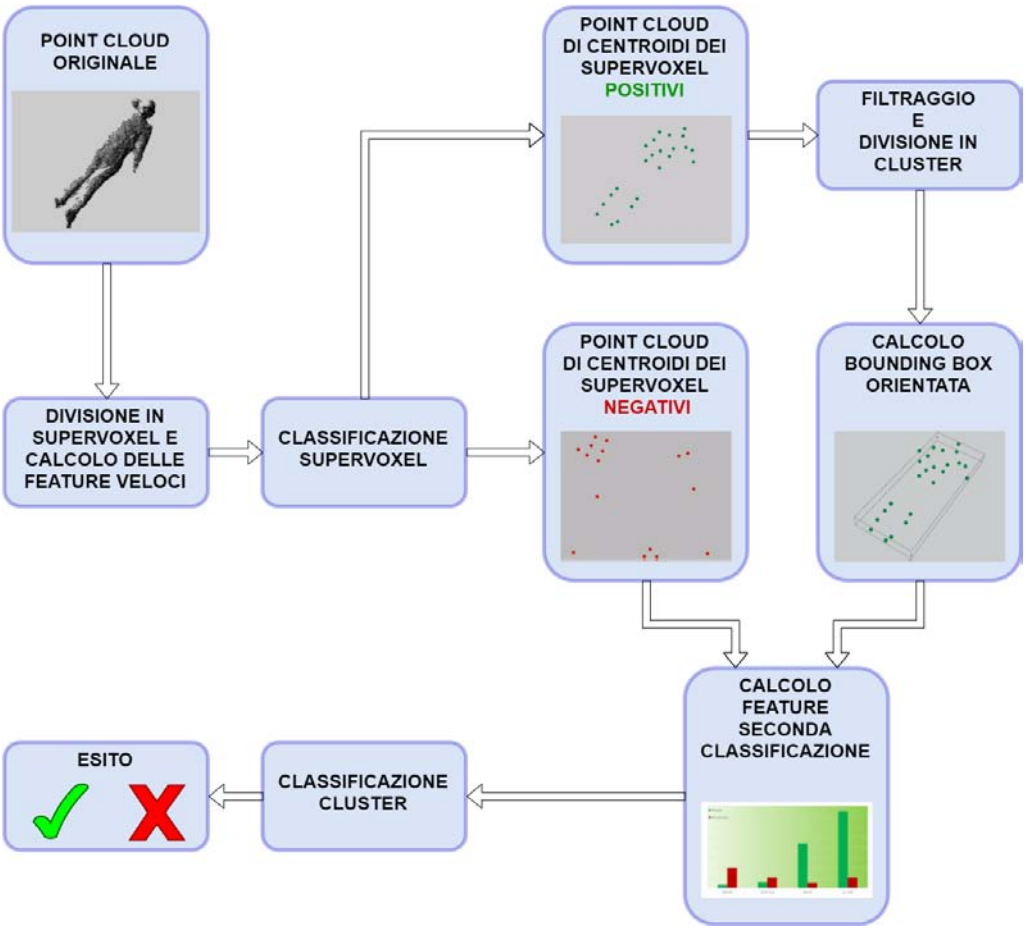


Figura 5.9: Schema riassuntivo del funzionamento del rilevamento mediante doppio classificatore.

Figura 5.9 riporta uno schema che riassume il funzionamento dell'implementazione finale dell'algoritmo descritto in questa sezione.

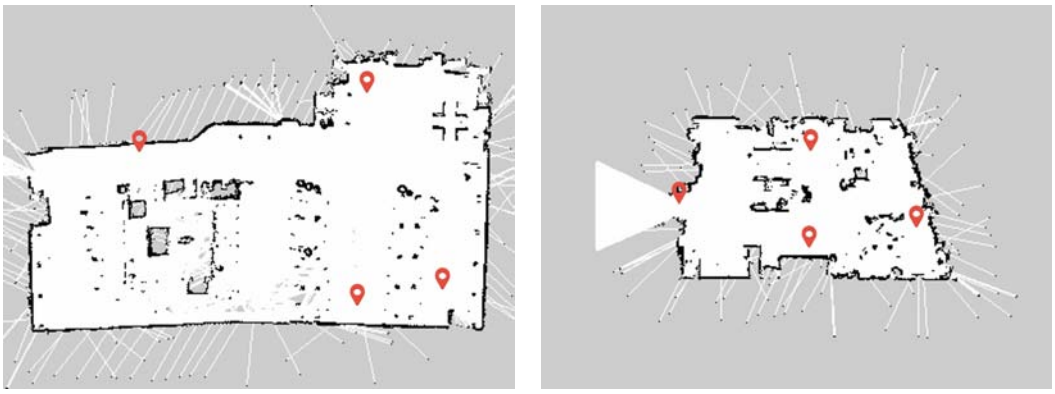
5.7 Sistema di misurazione delle prestazioni

Per valutare e confrontare i diversi algoritmi sperimentati è necessario predisporre un sistema di test e definire le metriche con le quali si andrà a misurare come questi performano. Nel caso in esame, si è deciso di ricorrere ad un sistema di annotazione delle performance basato sulla mappa dell'ambiente in cui viene condotto il test. Essendo note infatti le coordinate delle persone a terra all'interno della mappa e l'orientamento della telecamera del robot, si va a ricavare la "ground truth" necessaria a valutare l'algoritmo.

Una procedura alternativa per il monitoraggio delle prestazioni consiste nel ricorso ad un software apposito per l'etichettatura delle detection frame per frame, in un modo analogo a quello discusso in [51], dove ogni persona viene manualmente racchiusa in un rettangolo.

5.7.1 Sistema di test

Per quanto riguarda il sistema di test, si hanno a disposizione 8 bag, 4 registrate nell'aula CAD e 4 nell'aula LAB. Le bag relative alla stessa aula sono state ottenute facendo raggiungere posizioni predeterminate (sempre le stesse per ogni aula) al robot, in completa autonomia. In ogni bag, in prossimità del percorso seguito dal robot, sono presenti 4



(a) (b)

Figura 5.10: Posizione delle persone nell'ambiente di training (a sinistra) e di test (a destra).

persone distese a terra. Nonostante le coordinate della mappa in cui è possibile trovare le persone siano sempre approssimativamente le stesse (note a priori e riportate in Figura 5.10), le persone da una bag all'altra possono cambiare postura. Allo stesso modo, dato il fatto che il robot si muove autonomamente, tra una bag e l'altra della stessa aula possono registrarsi differenze per quanto riguarda le inquadrature, i tempi di percorrenza e l'interazione del robot con eventuali ostacoli dinamici presenti nel circondario.

Per evitare di generare soluzioni che troppo si adattano alle caratteristiche dei due ambienti ripresi dalle bag, si è deciso di procedere mantenendo separati i due ambienti. Le operazioni di taratura dei parametri degli algoritmi provati sono condotte utilizzando solamente le 4 bag relative all'aula CAD, che pertanto costituiscono il cosiddetto training set delle bag a disposizione. Di conseguenza, le 4 bag ottenute nella più piccola aula LAB sono sfruttate come test set, per verificare se le prestazioni ottenute in fase di training sono attendibili oppure affette da overfitting. Tale scelta è stata motivata anche dalla conformazione dell'aula CAD, ampia ed indicata per valutare con precisione la capacità dell'algoritmo di individuare persone prossime ai limiti del range di rilevamento; inoltre, come approfonditamente trattato nel precedente capitolo, tale aula presenta delle zone particolarmente soggette a disturbi di natura luminosa, utili per comprendere la robustezza al rumore visivo dell'algoritmo di detection. L'aula LAB, d'altro canto, presenta un ambiente più piccolo e popolato da numerosi ostacoli, dove alcune persone a terra sono inquadrate per un numero ridotto di frame e spesso vicine ad altri componenti dell'arredamento. Lo scenario è quindi più simile a quello che può essere ritrovato in un piccolo appartamento, con il robot costretto a passare spesso a distanza ridotta da tavoli e pareti.

5.7.2 Misurazione delle performance

Le metriche utilizzate per valutare la bontà delle scelte fatte devono garantire la possibilità di confrontare oggettivamente le soluzioni ideate, andando il più possibile a riassumere i pregi ed i difetti di ognuna.

Per valutare la bontà del codice creato si è deciso di impostare la misurazione delle performance a livello di frame. Si suppone inoltre che ogni frame contenga al massimo una persona da individuare. Per ogni point cloud elaborata dall'algoritmo quindi si va a verificare la sussistenza delle due seguenti condizioni:

1. *Una persona è inquadrata*

2. *Una persona è individuata*

Gli esiti di questi due test operati su ogni frame possono quindi essere riassunti in una matrice di confusione, la quale riporta:

- **True Positives** (**TP**), ovvero il numero di frame processati in cui una persona è inquadrata ed individuata correttamente, cioè dove la posizione del centroide che la colloca sulla mappa è distante meno di 1 m dall'effettiva posizione della persona.
- **True Negatives** (**TN**), ovvero il numero di frame processati in cui non è inquadrata né rilevata alcuna persona.
- **False Positives** (**FP**), ovvero il numero di frame processati in cui, indipendentemente dal fatto che la persona sia effettivamente inquadrata, il rilevamento è situato in una posizione distante più di 1 m dall'effettiva posizione della persona nella mappa.
- **False Negatives** (**FN**), ovvero il numero di frame processati in cui la persona è inquadrata ma non vi sono rilevamenti da parte del programma.

Da tali contatori è possibile ricavare i seguenti indici di prestazioni:

$$precision = \frac{TP}{TP + FP} \tag{5.7.1}$$

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.7.2}$$

$$recall = \frac{TP}{TP + FN} \tag{5.7.3}$$

In accordo con quanto sostenuto da Volkhardt, Schneemann e Gross in [30], in fase di valutazione degli algoritmi testati è preferibile premiare un basso numero di FP rispetto ad un maggior numero di TP. Ciò significa che il robot, durante la sorveglianza dell'area, sarà meno incline a sollevare falsi allarmi, nonostante possa essere necessario passare più volte per il medesimo posto prima di riuscire ad individuare la persona a terra. In analogia con quanto fatto dall'autore dell'articolo, anche in questo lavoro si andrà ad utilizzare come indice riassuntivo delle prestazioni dell'algoritmo in esame il $F_{0.5} - score$, così definito:

$$F_{0.5} = \frac{(1 + 0.5^2) * precision * recall}{0.5^2 * precision + recall} \tag{5.7.4}$$

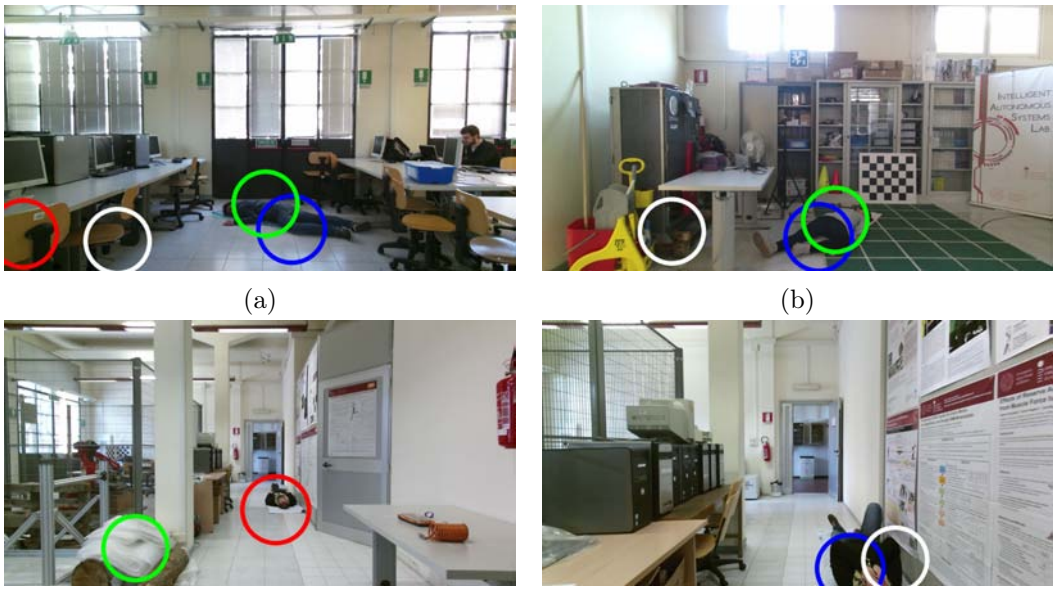
Questa metrica consiste nella media armonica pesata di *precision* e *recall*, dove viene posta maggiore enfasi sul contributo dato dalla *precision* rispetto a quello dato dalla *recall*.

Il reperimento delle informazioni necessarie a compilare i campi della matrice di confusione è svolto in modo automatico, mediante l'utilizzo di metodi creati ad-hoc ed inseriti direttamente nel codice del programma principale. Tale funzionalità è attivabile mediante l'impostazione di un opportuno parametro booleano direttamente dal file XML di configurazione.

All'avvio, il programma inizializza un contatore per ogni campo della matrice di confusione. Le coordinate reali delle persone a terra, riferite al frame */map* dell'aula, sono note a priori e caricate dal programma nella fase iniziale. Alla ricezione di una nuova point cloud, mediante un'opportuna trasformazione, tali coordinate sono riportate nel sistema di riferimento della Kinect, */kinect2_head_ir_optical_frame*. Si controlla quindi quali persone possono essere considerate visibili: il programma verifica se le loro posizioni ricadono all'interno del FoV della telecamera e se sono comprese nei limiti minimo e massimo di distanza entro i quali l'algoritmo è progettato per funzionare.

Determinato se una persona è visibile o meno, il programma procede come di consueto con le operazioni di ricerca di persone a terra nella point cloud. In ogni frame possono

esservi più detection da parte del programma: essendo la metrica di misura delle performance qui adottata basata sull'intero frame, è necessario stabilire un sistema di priorità tra i possibili eventi. È stato pertanto stabilito che le detection di tipo TP abbiano la priorità su tutte le altre: se in uno stesso frame, ad esempio, si registrano un TP ed un FP, solo il TP andrà ad incrementare il rispettivo contatore nella matrice di confusione. Ovviamente, inoltre, i FP hanno la precedenza sui FN. Se il programma segnala un rilevamento, viene avviato un controllo che misura la distanza tra le coordinate della persona individuata e quelle delle eventuali persone in vista. Se la distanza da una delle persone in vista è inferiore a 1 m, viene incrementato il contatore dei TP, e per il frame in corso l'annotazione delle performance è sospesa. Se invece la distanza è superiore a tale limite, il programma imposta un flag booleano, e procede fino a terminare l'elaborazione del frame corrente. Se non sono stati individuati TP nelle eventuali elaborazioni successive, viene incrementato il contatore dei FP. Se, al termine dell'elaborazione di un frame, il flag che segnala i falsi positivi è disabilitato, in funzione del test iniziale sulla visibilità delle persone sarà incrementato il contatore di TN o quello dei FN. Finita l'elaborazione della point cloud, un metodo apposito provvede a stampare nello standard output la matrice di confusione aggiornata, seguita dagli indici di performance calcolati a partire da essa.



(a) (b)

(c) (d)

Figura 5.11: Esempi di inquadrature ottenute con il visualizzatore creato.
Legenda: rosso: persona non in vista; blu: persona in vista; bianco: cluster non classificato come persona; verde: cluster classificato come persona.

Oltre a quelle appena descritte, nel codice sono state inserite anche funzioni di log che permettono di approfondire altri aspetti delle performance di identificazione, di utilità più pratica. Per ogni persona a terra nell'aula è mantenuto un contatore dei frame nei quali è inquadrata, mentre mediante un altro se ne annotano le detection. Questi dati aiutano a capire quale possa essere la natura dei mancati rilevamenti, spesso da ricercarsi in una particolare postura della persona o nella vicinanza ad altri oggetti della stanza.

È stato inoltre predisposto un sistema di visualizzazione in tempo reale delle detection. Grazie all'utilizzo del topic del sensore RGB della Kinect è infatti possibile vedere la scena dal punto di vista del robot in ogni istante (con minime differenze date di differenti FoV di sensore ad infrarossi e sensore a colori), pubblicando le sequenze di immagini ricevute mediante un visualizzatore di OpenCV. In tale visualizzatore sono anche indicati mediante indicatori circolari di diverso colore: la posizione reale delle persone cadute, le detection

del programma e gli eventuali cluster via via considerati. In Figura 5.11 è possibile vedere alcuni esempi del suo funzionamento. In particolare, Figura 5.11a esibisce i possibili colori che possono assumere gli indicatori: blu per le persone in vista, rosso per le persone non in vista, verde per i cluster classificati come persona, bianco per i cluster classificati come non persona. Figura 5.11c ritrae una situazione già discussa, ovvero il riconoscimento del tronco in aula CAD come una persona. Situazioni come quella ritratta in Figura 5.11d permettono invece di intuire con maggiore facilità le motivazioni di un mancato rilevamento, rendendo più rapida la ricerca di una soluzione.

Un'altra metrica di cui si tiene traccia, misurandone in tempo reale l'andamento, è il frame rate al quale lavora l'intero programma.

Test occlusione

Successivi test hanno fatto emergere un'ulteriore necessità, riguardante la fase di controllo della visibilità di una persona. Con il metodo sopra descritto, infatti, spesso accade che la persona sia considerata visibile quando invece è nascosta da oggetti presenti nell'ambiente. Tale situazione è particolarmente frequente nella piccola aula LAB, dove persone dall'altra parte della scrivania sono considerate visibili anche se totalmente nascoste al sensore. Per tal motivo si è deciso di implementare un metodo che vada a verificare che tra la telecamera e la persona a terra non siano presenti ostacoli. Tale sistema campiona una serie di posizioni, a distanza regolare, lungo la traiettoria tra il sensore (origine del frame di riferimento della Kinect) e la posizione della persona alzata di 0.5 m (espediente necessario per evitare che la stessa persona sia rilevata come un ostacolo). In ognuna di tali posizioni viene effettuata una ricerca dei vicini entro un dato raggio, utilizzando un oggetto `kdTree` della libreria `KdTreeFLANN` di Pointclouds.org, ed invocando il metodo `radiusSearch`. Qualora una ricerca restituisca un numero di punti superiore ad una data soglia, la visuale della persona è considerata ostruita da un ostacolo. Diverse prove sulle bag di training e di test hanno portato ad usare come parametri 0.2 come raggio e 20 come soglia massima di punti.

5.8 Risultati sperimentali

In questa sezione verranno presentate e discusse le performance ottenute dagli algoritmi descritti in precedenza.

5.8.1 Prestazioni del rilevamento mediante bounding box orientata

	accuracy	*precision*	*recall*	$F_{0.5}$
Training set	0.88	0.65	0.33	0.54
Test set	0.84	0.64	0.26	0.50
Totale	0.86	0.65	0.29	0.52

Tabella 5.1: Prestazioni medie del rilevamento mediante bounding box orientata

L'algoritmo si basa su un concetto molto semplice, ovvero quello delle principali dimensioni di un corpo umano. La semplicità di implementazione del sistema e ottenimento delle feature ad esso necessarie tuttavia paga il prezzo di peggiori performance di detection in casi d'uso reali. Tabella 5.1 riassume le performance di rilevamento ottenute nelle bag di training e di test.

L'incapacità di distinguere le persone da altri oggetti di simile forma presenti negli ambienti di simulazione influisce pesantemente sul numero di FP, con il risultato che la *precision* si attesta su valori generalmente troppo bassi per essere definiti accettabili. Al fine di mitigare l'alto numero di falsi positivi causati dal rumore luminoso, si è scelta una rimozione del terreno più aggressiva, la quale tuttavia innalza considerevolmente il numero di FN. Tali motivazioni giustificano i valori ottenuti per la *recall*, anche questi ben lontani dall'essere definibili accettabili in uno scenario di utilizzo reale del robot. Tale scelta tuttavia ha permesso di portare stabilmente lo score $F_{0.5}$ su valori superiori a 0.5.

Sicuramente uno dei pregi di questo approccio è la leggerezza dal punto di vista del carico computazionale, fattore importante dal momento che più alto è il frame rate, maggiori sono le probabilità di individuare persone inquadrate per poco tempo. Nei diversi test condotti si è registrato un frame rate medio pari a 14.2 Hz.

5.8.2 Prestazioni del rilevamento mediante feature veloci

	accuracy	*precision*	*recall*	$F_{0.5}$
Training set	0.85	0.84	0.29	0.61
Test set	0.78	0.84	0.05	0.17
Totale	0.82	0.84	0.17	0.39

Tabella 5.2: Prestazioni medie del rilevamento mediante feature veloci

L'algoritmo è stato implementato per superare le debolezze del primo approccio seguito. Come testimonia Tabella 5.2, tuttavia, l'algoritmo esibisce un comportamento completamente differente a seconda dell'ambiente in cui è utilizzato. È stato raggiunto l'obiettivo di migliorare la *precision* dei rilevamenti, con valori sempre superiori a 0.8 in tutte le simulazioni effettuate. Tale dato è confermato anche dai riscontri visivi nel visualizzatore, dove si nota chiaramente la minor tendenza a generare falsi positivi.

Mentre nelle bag di training lo score $F_{0.5}$ riporta un miglioramento complessivo rispetto all'approccio precedente, nelle bag di test la situazione è opposta, con il sistema a feature

veloci che si dimostra peggiore di 33 punti percentuali. Ciò può essere sintomo di overfitting nella scelta dei parametri di ground segmentation e riduzione del rumore, tuttavia sicuramente parte della responsabilità per le pessime prestazioni è da imputarsi alla logica con cui l'algoritmo rileva le persone a terra. Come già accennato, il fatto di contare il numero di detection entro una sfera di raggio prefissato attorno ad ogni supervoxel verde si rivela inefficace in alcuni casi ben noti. In tutte le bag ottenute nell'aula LAB, le persone, oltre ad essere spesso troppo vicine ad oggetti e componenti dell'arredo, sono inquadrate interamente per poco tempo.

Inoltre, nonostante l'algoritmo si basi su feature ottenibili a basso costo computazionale, la maggior complessità generale di questa soluzione si riflette anche sulle prestazioni. Nei diversi test effettuati il frame rate medio si è attestato su 3.9 Hz.

5.8.3 Prestazioni del rilevamento mediante doppio classificatore

	accuracy	*precision*	*recall*	$F_{0.5}$
Training set	0.92	0.87	0.77	0.84
Test set	0.90	0.89	0.73	0.85
Totale	0.91	0.88	0.75	0.85

Tabella 5.3: Prestazioni medie del rilevamento mediante doppio classificatore

Il buon funzionamento di questo algoritmo è testimoniato dal miglioramento uniforme degli indici di prestazione esposti in Tabella 5.3.

Con uno score $F_{0.5}$ medio pari a 0.85, il metodo a due fasi di classificazione si dimostra infatti nettamente più affidabile delle alternative proposte. Questo risultato è stato raggiungibile perfezionando la capacità dei due classificatori di individuare le persone nel maggior numero di frame possibili, andando di conseguenza a migliorare l'indice più critico nei casi precedenti, la *recall*.

La complessità introdotta con l'aggiunta di una seconda fase di classificazione è tuttavia considerevole, e tal fatto si riflette tanto sulle prestazioni quanto sulla difficoltà nella messa a punto. Per quanto riguarda il primo problema, è facilmente intuibile che l'algoritmo lavori ad un frame rate inferiore rispetto a quello raggiunto dalle soluzioni precedenti, pari a 3.2 Hz in media. Riguardo al secondo problema, l'introduzione del passaggio di filtraggio, del clustering e della possibilità di agire mediante una soglia sulla selettività della classificazione ha incrementato considerevolmente il numero di parametri da settare. Questi, peraltro, sono spesso fortemente correlati tra loro: la correzione di una soglia riguardante il clustering può richiedere l'adozione di una adeguata contromisura in fase di filtraggio, ad esempio. L'azione su determinati parametri richiede inoltre che siano generati nuovamente i file csv di training e test set, con conseguente allenamento e test dei classificatori, operazioni che richiedono considerevoli quantità di tempo e risorse computazionali.

5.8.4 Confronto tra le soluzioni individuate

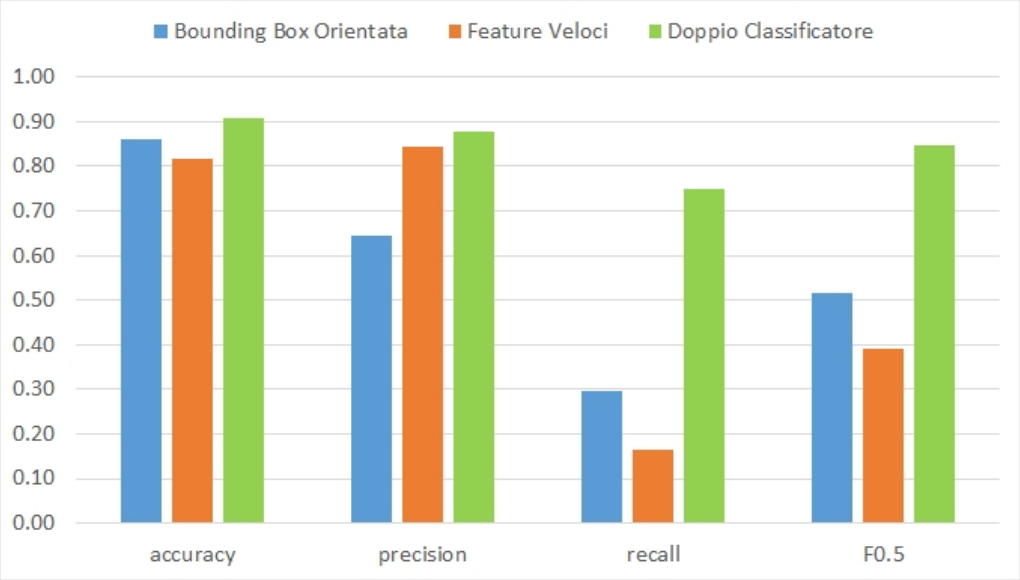


Figura 5.12: Confronto diretto tra i punteggi ottenuti da ciascun algoritmo nei quattro indici di prestazione.

Figura 5.12 riassume e paragona quanto visto nel dettaglio nelle tre precedenti sezioni. Considerando i primi due algoritmi implementati come il punto di partenza ed il metodo a doppio classificatore come il risultato finale, sono chiari gli importanti miglioramenti ottenuti. Ragionando in termini assoluti, tuttavia, una *precision* di poco inferiore a 0.9 significa che un rilevamento ogni 10 in realtà si è dimostrato un falso allarme. Ulteriori tecniche pensate per ridurre ulteriormente questo margine di errore saranno presentate nel capitolo 6.

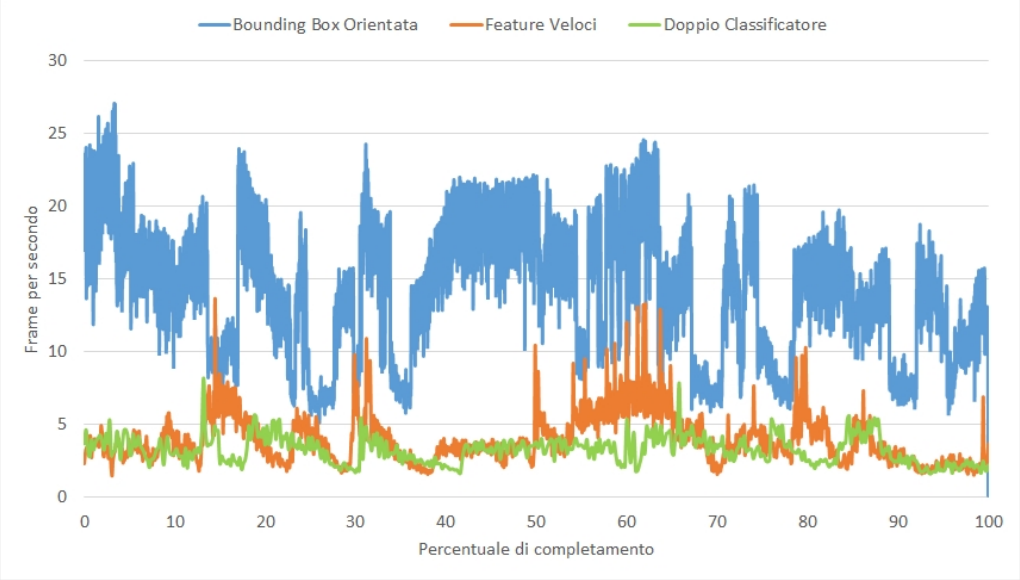


Figura 5.13: Andamento del frame rate dei tre algoritmi di rilevamento durante la simulazione con la bag cad_2.

In Figura 5.13 sono invece confrontati i tre algoritmi per quanto riguarda il numero di frame analizzati ogni secondo, durante l'esecuzione della bag cad_2. Come era pronosti-

cabile, il programma basato sulla bounding box orientata registra prestazioni nettamente migliori dei due concorrenti. Le zone dove si registrano le cadute di frame rate sono quelle che presentano maggiori cluster da analizzare, perché richiedono maggior tempo per elaborare l'intero frame. Le prestazioni dell'algoritmo basato solamente sulle feature veloci si dimostrano lievemente migliori di quelle dell'ultimo algoritmo analizzato, pur restando quasi sempre nell'intorno dei 5 fps. La versione finale del programma di rilevamento si attesta su prestazioni lievemente inferiori a quest'ultimo. Nonostante ciò, la versione proposta riesce ad individuare efficacemente tutte le persone in ogni ambiente di test, segnalandone la presenza nella maggior parte dei frame che le ritraggono completamente. Questo fattore sarà di cruciale importanza per risolvere i problemi affrontati nel prossimo capitolo.

Capitolo 6

Localizzazione delle persone a terra

In questo capitolo si andranno a discutere le operazioni di localizzazione e visualizzazione delle persone individuate sulla mappa dell'ambiente.

L'importanza delle operazioni di localizzazione non è limitata all'utilità di conoscere esattamente il luogo dove si trova la persona caduta, ma risponde soprattutto alla necessità di completare ed introdurre ulteriori controlli sulle informazioni provenienti dall'algoritmo di rilevamento. Quest'ultimo, come visto, basa il proprio funzionamento sull'elaborazione del singolo frame. Nel caso avesse prestazioni ideali (ossia *accuracy*, *precision* e *recall* pari a 1.0) il lavoro descritto in questo capitolo non sarebbe necessario. Il robot individuerebbe subito la persona caduta al primo frame, avviando le appropriate procedure di emergenza. Sfortunatamente, anche nella versione finale proposta, il sistema non è esente da falsi allarmi, né riesce sempre a garantire l'identificazione di un'emergenza al primo passaggio del robot. Per quanto ci si impegni a raffinare il tuning dei parametri e le prestazioni dei classificatori utilizzati, per garantire l'efficacia e l'affidabilità nell'individuazione di persone a terra è necessario svincolarsi dalla logica basata sui singoli frame e ragionare ad un più alto livello di astrazione, elaborando l'informazione ricavabile dalle detection. In sostanza, si andrà a sfruttare l'informazione temporale e spaziale per "convalidare" i rilevamenti oppure contrassegnarli definitivamente come falsi allarmi.

Di seguito, in "Algoritmo di localizzazione" verrà descritto il funzionamento dell'algoritmo, accompagnato da considerazioni sulle scelte fatte durante il percorso di implementazione.

In "Misura delle prestazioni" saranno introdotte le metriche utilizzate per valutare i risultati ottenuti dal sistema.

In "Risultati sperimentali", infine, saranno presentati e discussi i risultati dei test effettuati.

6.1 Algoritmo di localizzazione

Il compito della localizzazione è affidato al nodo `fallen_people_locator`, appartenente al pacchetto `dofp_vision_tools`. Ne è stata creata anche una versione (omonima) adattata all'uso con `fall_detection_node`, appartenente al pacchetto `orobot_fall_detection`. Eccezion fatta per il valore di alcuni parametri, il codice ed il modo in cui funzionano i due nodi sono identici, pertanto per semplicità in questa trattazione si farà riferimento sempre al primo.

`fallen_people_locator` all'avvio si iscrive al topic `/oversegmenter/detections`, dove sono pubblicati i messaggi di tipo `geometry_msgs/PointStamped` che riportano le coordinate dei rilevamenti. Questi ultimi sono già riferiti al sistema di riferimento */map*, pertanto sono immediatamente riconducibili ad una precisa posizione sulla mappa.

Le detection vengono gestite all'interno del programma in forma di markers. Questi sono particolari tipi di messaggio, istanze della classe `visualization_msgs::Marker`, pensati per permettere di visualizzare segnaposto di forma, posa, dimensione e colore personalizzabili in RViz. Sono inoltre utilizzate delle strutture dati ad-hoc, pensate per contenere tali marker, istanze della classe `visualization_msgs::MarkerArray`. Per la precisione se ne sfruttano due, `marker_array` e `people_array`. La prima contiene i rilevamenti singoli non ancora associati ad una persona; la seconda ospita invece i marker rappresentanti le persone individuate e confermate dall'algoritmo. Per convenzione, i marker appartenenti

alla prima sono generati di forma cubica, mentre per la seconda la forma prescelta è quella cilindrica.

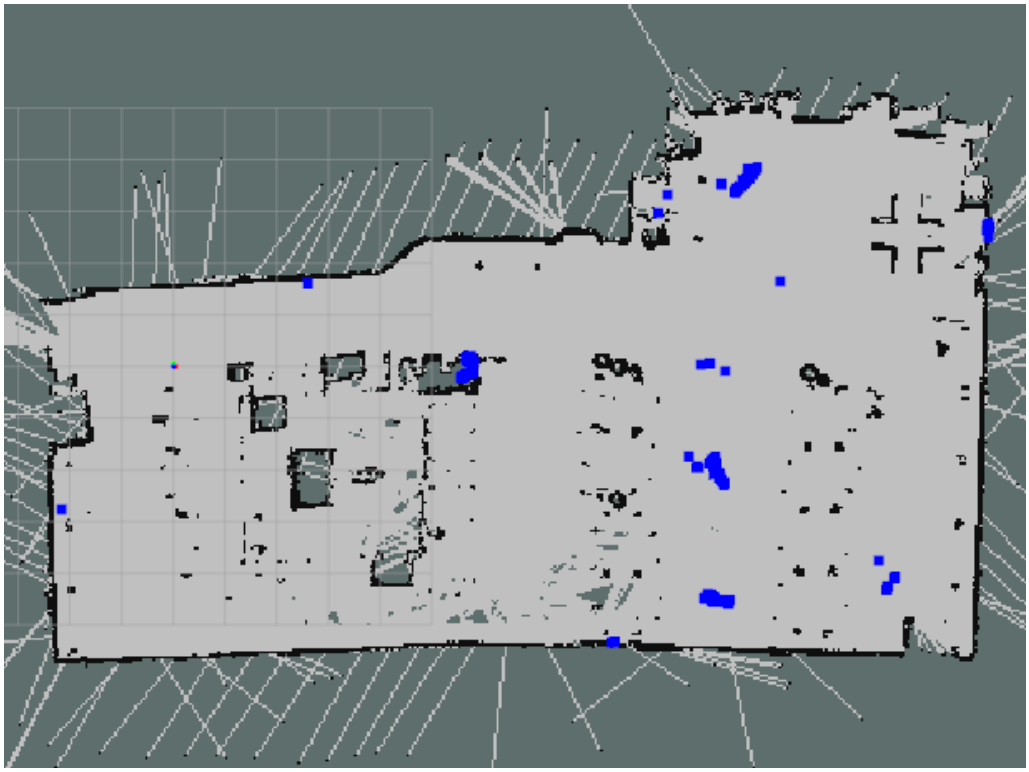


Figura 6.1: Marker non ancora elaborati che segnano la posizione delle detection sulla mappa. Queste sono state ottenute sfruttando i rilevamenti mediante bounding box orientata, con il controllo ostacoli sulla mappa disattivato.

Alla pubblicazione di un nuovo messaggio in `oversegmenter/detections`, viene attivata una callback. Questa controlla se la detection è associabile ad una delle persone in `people_array`, controllando se è a meno di `distance_threshold` metri dalla posizione di uno dei marker; `distance_threshold` è un parametro impostato da launch file, attualmente avente un valore pari a 1 m. Se il rilevamento è riconducibile ad una persona già individuata, per motivi di efficienza computazionale viene ignorato. Altrimenti è generato ed inserito in `marker_array` un marker di forma cubica, posizionato esattamente nelle coordinate specificate dal messaggio in ingresso. Tutti i marker generati in tal modo solo inizializzati con il colore blu, che contraddistingue quelli non ancora elaborati dagli altri. Il risultato alla fine di tale procedura è proposto in Figura 6.1.

L'elaborazione dei rilevamenti è affidata ad una callback periodica, attivata automaticamente ogni 10 secondi. Questa, all'avvio, ricerca, per ogni marker, i vicini entro un determinato raggio, pari a `distance_threshold`. L'indirizzo in memoria di ogni vicino è salvato in un'apposita lista, contenuta nella cella di un vettore avente il medesimo indice di quello del marker considerato in `marker_array`. Quindi, sfruttando l'informazione sui vicini appena ottenuta, un apposito loop crea i cluster di marker, collegando tutti i segnaposto separati da una distanza inferiore a `distance_threshold`; tale collegamento è effettuato sia impostando il namespace degli oggetti `visualization_msgs::Marker` allo stesso valore, sia facendo sì che tutti gli appartenenti ad un cluster abbiano ugual colore (scelto casualmente tra tutti quelli possibili, blu escluso). A questo punto la callback periodica ha individuato e raggruppato tutti i rilevamenti che, a causa della loro vicinanza, sono riconducibili alla stessa persona. Figura 6.2 riporta il risultato alla fine di questa fase.

L'ultima operazione della callback consiste nell'invocazione di un metodo il cui scopo

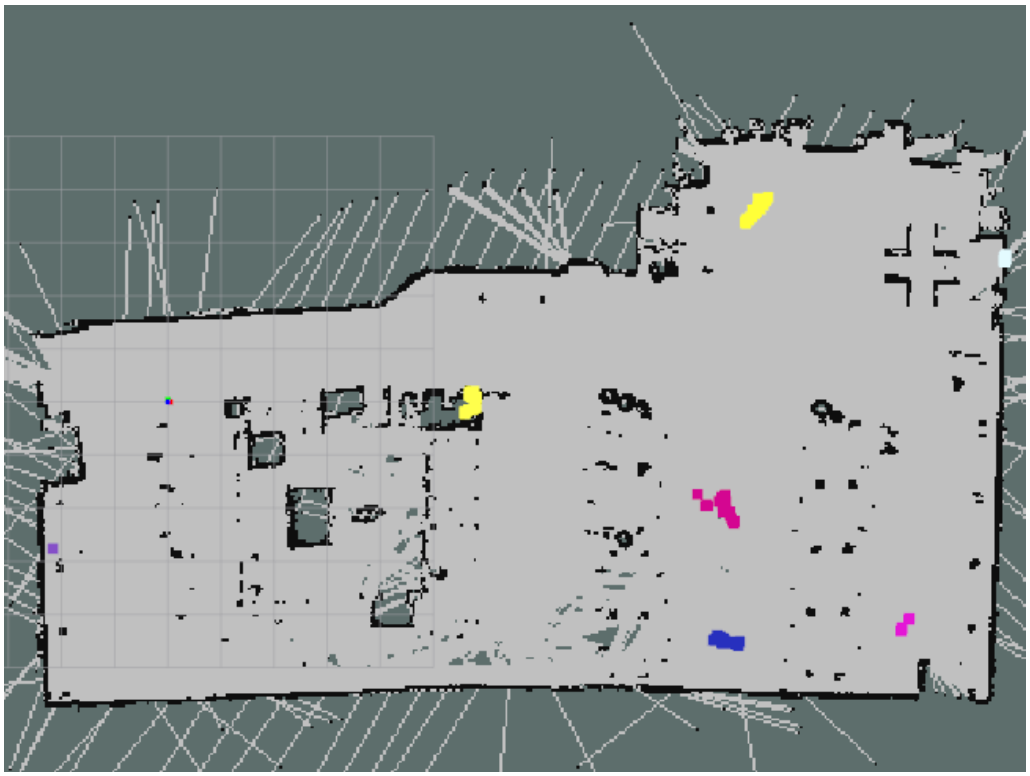


Figura 6.2: Marker raggruppati in cluster.

è quello di valutare ogni singolo cluster. Questo innanzitutto avvia un controllo sul timestamp dei rilevamenti nel cluster. Per evitare infatti che l'elaborazione di un numero eccessivo di detection vada a rallentare il programma, marker presenti sulla mappa da più di 60 secondi (parametro comunque regolabile da launch file) sono eliminati.

Ogni iterazione del metodo quindi è dedicata all'esame di un singolo cluster. Affinché quest'ultimo sia ritenuto rappresentativo di una persona a terra, devono essere soddisfatti alcuni criteri ben precisi. Inizialmente è stato previsto come unico criterio il superamento di un numero minimo di elementi appartenenti al cluster. Tale soglia, impostata per via empirica a 20, permette l'individuazione della maggior parte delle persone nella stanza, andando al contempo a sollevare falsi allarmi solo in un numero ridotto di occasioni. Successivamente si è constatato come il ricorso ad una soglia sul numero di detection sia una soluzione poco flessibile e troppo legata alle condizioni ambientali del luogo in cui è stato allenato l'algoritmo. Simulazioni nell'ambiente di test, infatti, hanno sollevato problemi a causa del minor numero generale di detection rilevate per ogni persona. Di conseguenza si è deciso di passare ad una soglia basata sulla frequenza delle detection. Tipicamente, infatti, i rilevamenti ottenuti quando è inquadrata una persona, anche se in numero contenuto, sono separati l'uno dall'altro da intervalli di tempo molto ridotti. Al contrario, i falsi positivi, causati da fenomeni rumorosi od ostacoli di forma "critica", anche se numerosi, sono spesso generati in lenta successione, con intervalli tra una segnalazione e l'altra solitamente superiori al secondo. Dividendo il numero di elementi nel cluster per la differenza tra l'ultima e la più datata detection al suo interno si ottiene il rate di pubblicazione relativo al cluster. Confrontando tale rate con un rate di riferimento impostato da launch file è quindi possibile decidere più accuratamente se le detection sono da ritenersi una persona o meno. Empiricamente, ed in funzione del frame rate raggiunto dal programma di detection nel computer in uso, la soglia di riferimento per il rate è stata impostata ad 1 Hz. Ad ogni modo, una soglia molto bassa sul minimo numero di rilevamenti in un cluster è stata mantenuta, per scongiurare l'eventualità che rapide successioni di 2 - 3 detection

rumorose siano considerate una persona. Pertanto non saranno segnalabili come persone i cluster composti da un numero di rilevamenti inferiore a 5. Cluster che non rispettano tali soglie sono mantenuti invariati nell'array di markers, in modo da dare la possibilità ad eventuali successivi rilevamenti di confermarne la veridicità. Alternativamente, il cluster sarà progressivamente disgregato, nelle successive chiamate della callback periodica, dal controllo sull'età dei marker. Qualora il cluster ottenga esito positivo nei test descritti, viene generato un nuovo marker di forma cilindrica, inserito in `people_array`. La callback periodica si conclude con la pubblicazione di `marker_array` e `people_array` rispettivamente nei topic `/fallen_detections` e `/fallen_people`. Un esempio di ciò che si vede in RViz al termine dell'algoritmo è riportato in Figura 6.3.

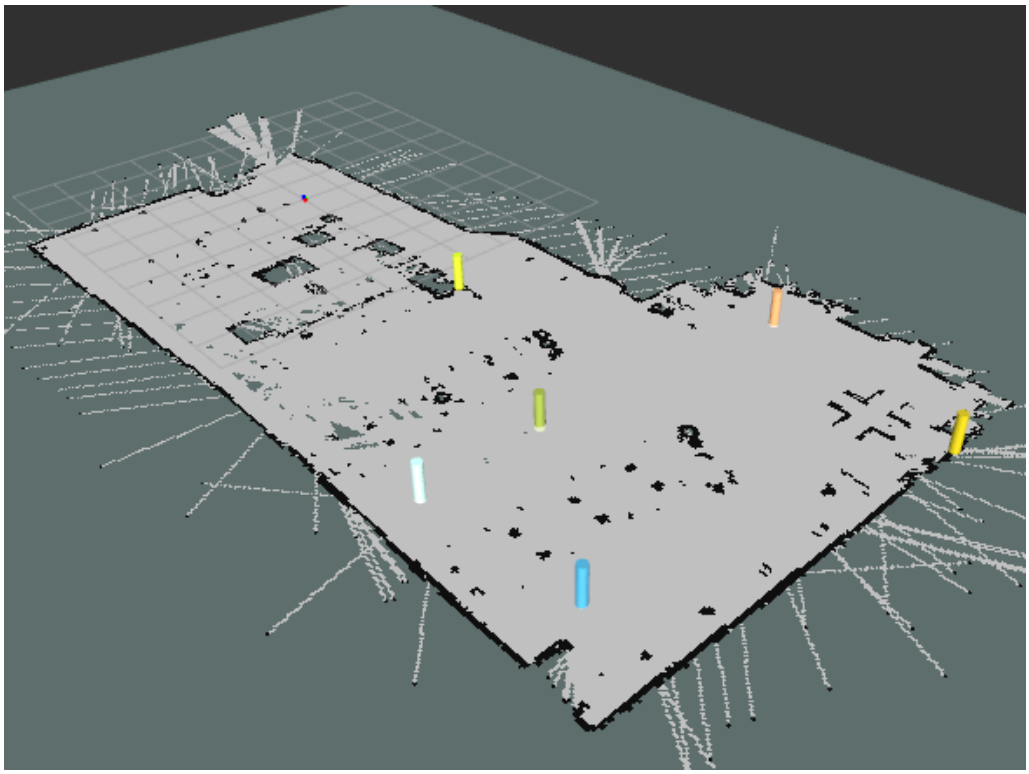


Figura 6.3: Marker rappresentanti le persone.

6.2 Misurazione delle prestazioni

Per quanto riguarda la scelta degli ambienti di training e test dell'algoritmo, è stata effettuata una scelta analoga a quella descritta nel capitolo "Rilevamento di persone a terra". Le 4 bag ottenute nell'aula CAD sono quindi utilizzate come ambiente di taratura dei parametri dell'algoritmo, mentre le 4 bag relative all'aula LAB sono riservate al test delle scelte effettuate.

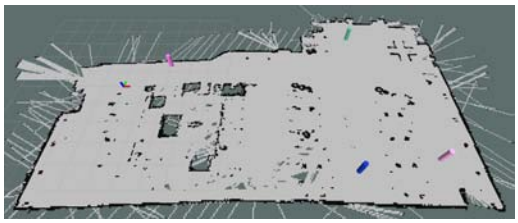
Per quanto riguarda le metriche, ancora una volta si ricorrerà alla matrice di confusione e agli indici di prestazioni da essa ottenibili, utilizzando le stesse formule presentate nel precedente capitolo. Cambia invece il modo in cui sono valutate le prestazioni. Non è più possibile basare la misura delle performance sui singoli frame, dato che l'algoritmo sfrutta informazioni temporali ricavate da più frame successivi per decretare la presenza di una persona in un determinato luogo. Di seguito è quindi riformulato il significato dei quattro campi della matrice di confusione:

- **True Positives** (**TP**), ovvero il numero di persone in `people_array` situate effettivamente in prossimità di una persona reale nella mappa.
- **True Negatives** (**TN**), ovvero il numero di cluster di detection in `marker_array` non trasformati in persona; il rilevamento è effettuato al momento dell'eliminazione dell'ultimo esemplare del cluster in seguito al controllo del timestamp.
- **False Positives** (**FP**), ovvero il numero di persone in `people_array` non situate in prossimità di una persona reale sulla mappa.
- **False Negatives** (**FN**), ovvero il numero di persone reali nella mappa non individuate da un segnaposto in `people_array`.

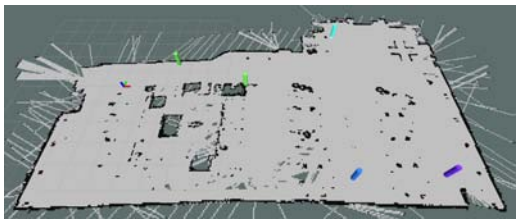
Per quanto riguarda il peso computazionale, le operazioni maggiormente critiche sono svolte nella callback periodica, e sono effettuate una volta al minuto. Gli espedienti introdotti per alleggerire il più possibile le operazioni in tempo reale permettono di non trascurare alcun rilevamento, anche nel caso di numerosi frame consecutivi contenenti una persona a terra.

6.3 Risultati sperimentali

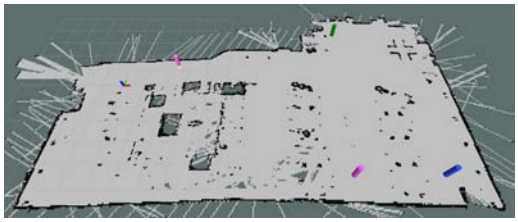
In Figura 6.4 e Figura 6.5 sono riportati gli esiti delle simulazioni nelle bag di training set e test set. L'obiettivo principale, ovvero quello di individuare tutte le persone a terra in tutti i test, è stato raggiunto. Tale risultato permette di affermare che il basso frame rate al quale lavora l'algoritmo di rilevamento frame per frame ideato non va comunque a compromettere l'efficacia del sistema di rilevamento e localizzazione nel suo complesso. Per quanto riguarda il conteggio dei FP, il programma ha sollevato complessivamente 2 falsi allarmi in tutte le simulazioni provate, visibili in Figura 6.4b e Figura 6.4d. Indagando la natura di tali falsi allarmi, si nota che questi non sono dovuti a rumore luminoso, bensì alle caratteristiche stesse dell'aula. Il falso positivo registrato in cad_2 è localizzato in prossimità del tronco d'albero, già discusso per la sua capacità di essere confuso con una persona a terra dai classificatori utilizzati. In genere la possibilità di un falso allarme ad esso dovuto è sventata dall'intervento del controllo ostacoli sulla mappa, test che qui ha fallito a causa del numero di rilevamenti non esattamente sovrapposti alle celle occupate. Una motivazione simile giustifica anche l'esistenza del falso positivo in cad_4, localizzato in corrispondenza di uno scaffale.



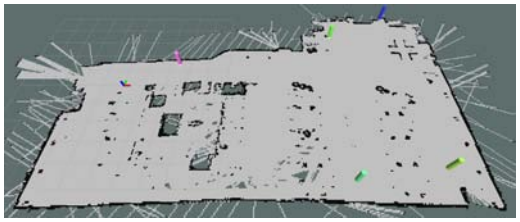
(a) cad_1, 4 persone individuate, 0 falsi positivi, 0 persone non trovate



(b) cad_2, 4 persone individuate, 1 falso positivo, 0 persone non trovate

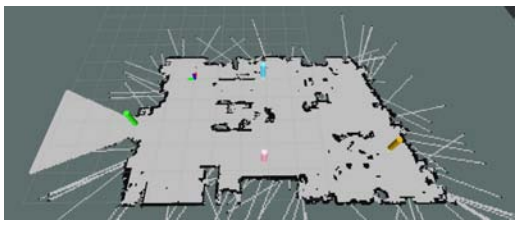


(c) cad_3, 4 persone individuate, 0 falsi positivi, 0 persone non trovate

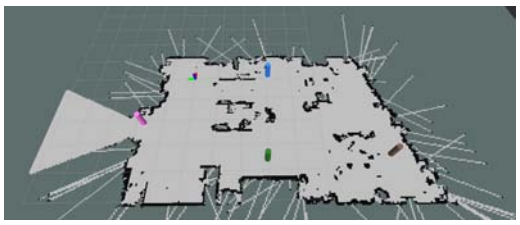


(d) cad_4, 4 persone individuate, 1 falso positivo, 0 persone non trovate

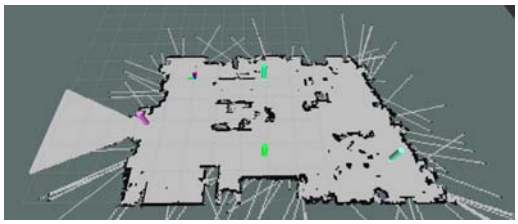
Figura 6.4: Risultati finali della localizzazione di persone a terra nella mappa nelle bag di training.



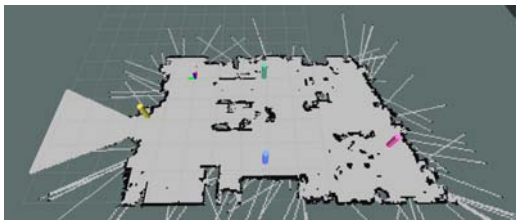
(a) lab_1, 4 persone individuate, 0 falsi positivi, 0 persone non trovate



(b) lab_2, 4 persone individuate, 0 falsi positivi, 0 persone non trovate



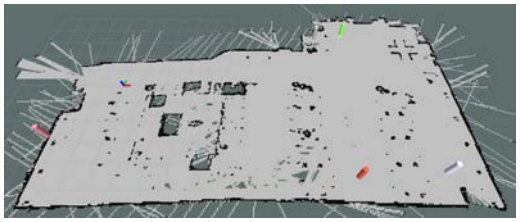
(c) lab_3, 4 persone individuate, 0 falsi positivi, 0 persone non trovate



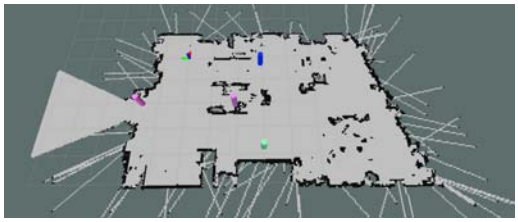
(d) lab_4, 4 persone individuate, 0 falsi positivi, 0 persone non trovate

Figura 6.5: Risultati finali della localizzazione di persone a terra nella mappa nelle bag di test.

Ai fini del confronto, sono state effettuate alcune simulazioni, in entrambi gli ambienti, anche utilizzando gli altri due algoritmi descritti nel capitolo "Rilevamento di persone a terra". Lo scopo di questo test comparativo è quello di comprendere quanto possa il programma di localizzazione realizzato migliorare il funzionamento di un algoritmo di detection frame per frame poco performante. Figura 6.6 e Figura 6.7 mostrano i risultati relativi alle bag cad_2 e lab_2. Si nota che le performance in tutti i casi sono peggiori di quelle ottenute con il programma di rilevamento finale. Il sistema che sfrutta la sola bounding box orientata è più sensibile al rumore, che in diverse occasioni è talmente accentuato da superare anche la soglia sul frame rate minimo. Il programma basato su feature veloci invece è molto più robusto contro il rumore ma fatica ad individuare persone poste troppo vicine a pareti ed ostacoli, come dimostrano le pessime prestazioni di Figura 6.7b.

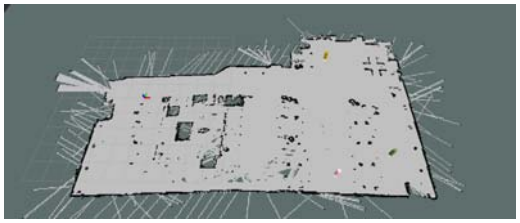


(a) cad_2, 3 persone individuate, 1 falso positivo, 1 persona non trovata



(b) lab_2, 3 persone individuate, 1 falso positivo, 1 persona non trovata

Figura 6.6: Localizzazione applicata a detection effettuate con bounding box orientata.



(a) cad_2, 3 persone individuate, 0 falsi positivi, 1 persona non trovata

(b) lab_2, 1 persona individuata, 0 falsi positivi, 3 persone non trovate

Figura 6.7: Localizzazione applicata a detection effettuate con feature veloci.

Capitolo 7

Conclusioni

In questa tesi sono stati presentati metodi efficaci per abilitare un robot di telepresenza a svolgere un ruolo di primaria importanza nell'assistenza alle persone di età avanzata.

La tecnica della fusione tra dati provenienti da sensore LRF e Kinect One si è dimostrata in grado di integrare efficacemente nella mappa bidimensionale tutte le informazioni necessarie al robot per evitare collisioni. Si è appurato come un efficace filtraggio dei dati provenienti dalla Kinect sia un elemento imprescindibile per ottenere un risultato soddisfacente. Cercare di soddisfare tale necessità ha tuttavia evidenziato un altro aspetto critico, ovvero la proporzionalità diretta tra frame rate dei dati fusi e bontà delle operazioni di mappatura e navigazione. La soluzione individuata, consistente nel dividere la point cloud in due fasce a seconda della distanza, filtrate in maniera differente, si è dimostrata in grado di garantire la pulizia delle tecniche di filtraggio più aggressive senza andare ad intaccare eccessivamente le prestazioni. I test condotti hanno permesso di individuare anche le principali limitazioni dell'approccio seguito. L'utilizzo del sensore Kinect comporta un impatto considerevole sulle prestazioni, soprattutto in fase di navigazione. Un'altra limitazione importante derivante dall'uso del sensore Kinect riguarda l'obbligo di sfruttare solo una porzione ridotta del FoV del LRF, con una conseguente maggiore difficoltà nel creare la mappa dell'ambiente.

L'algoritmo di rilevamento proposto si dimostra in grado di rilevare efficacemente tutte le persone presenti negli ambienti di test. Grazie al ricorso a due classificatori in serie è possibile individuare con buona precisione le persone a terra, dimostrando un'ottima robustezza contro i falsi positivi dovuti al rumore presente nella point cloud. La tecnica a due fasce utilizzata per segmentare il pavimento permette di sfruttare appieno il range visivo del sensore Kinect, aumentando le possibilità di rilevare persone distanti fino a 5 metri. Il sistema di monitoraggio delle performance realizzato per l'occasione inoltre si è rivelato uno strumento accurato e oggettivo nel confrontare soluzioni di natura differente. La maggiore debolezza dell'algoritmo di rilevamento rimane legata al basso frame rate al quale opera, che potrebbe costituire un fattore critico in ambienti piccoli e ricchi di ostacoli alla visuale.

L'algoritmo di localizzazione delle persone svolge efficacemente il suo lavoro, permettendo di filtrare la maggior parte dei falsi positivi generati in fase di rilevamento e di convalidare le detection attendibili. I test hanno confermato la capacità di riconoscere tutte le persone a terra disposte nell'ambiente. Hanno altresì presentato l'insorgere di due falsi allarmi, dovuti ad oggetti (o porzioni di essi) la cui forma e posizione è simile a quella di un essere umano disteso.

Date queste ultime considerazioni, si possono individuare facilmente delle linee guida per possibili sviluppi futuri.

Per quanto riguarda il robot, sicuramente il primo miglioramento potrebbe riguardare l'utilizzo di un'unità di elaborazione più prestante. L'adozione di una scheda integrata come NVidia Jetson TX1[1], ad esempio, potrebbe garantire le prestazioni necessarie per supportare al meglio tutte le operazioni connesse all'utilizzo della Kinect [52, 53]. Come conseguenza si avrebbe l'immediato miglioramento delle prestazioni di mappatura, navigazione e rilevamento delle persone a terra. Altri vantaggi derivanti da una simile scelta riguardano la riduzione di ingombri e consumi.

[1] link: `http://www.nvidia.com/object/jetson-tx1-module.html`

Un'altra possibile miglioria, mirata a colmare una lacuna dell'apparato sensoriale del robot, riguarda l'introduzione di ulteriori sensori posti ad un'altezza intermedia tra la telecamera ed il sensore LRF. Dei rilevatori di tipo ultrasonico, ad esempio, sarebbero sufficienti per andare ad eliminare l'angolo cieco di fronte al robot. Una seconda Kinect, posta nella stessa posizione, amplierebbe le possibilità di rilevare ostacoli e persone a terra. Sensori termici, in alternativa, potrebbero assicurare una svolta definitiva nella capacità di individuazione delle cadute.

Anche sul frangente software vi sono numerose possibilità di miglioramento. Per scongiurare la possibilità che il robot si de-localizzi si possono introdurre tag grafici con pattern prestabiliti in punti ben noti dell'ambiente. Il robot, una volta individuati i tag, calcola la propria posizione in relazione ad essi e di conseguenza può ri-localizzarsi sulla mappa.

La ricerca di altre combinazioni di feature veloci potrebbe permettere di aumentare ulteriormente la precisione dei rilevamenti, scongiurando l'eventualità di falsi allarmi. Un altro possibile sviluppo in tal senso riguarda l'introduzione dell'informazione sul colore, ricavata dal sensore RGB della Kinect One. Tale operazione aprirebbe la strada alla sperimentazione ed integrazione nel sistema esistente di numerose altre tecniche presenti in letteratura, tra le quali vi è il già citato lavoro di Wang, Zabir e Leibe [29]. Altri dati importanti ai fini della classificazione sempre ottenibili dalla telecamera montata sul robot riguardano lo spettro infrarosso.

Importante è infine condurre ulteriori test e più accurate sessioni di tuning dei parametri. L'ampliamento di training e test set permetterebbe ad esempio l'allenamento di classificatori più performanti, abilitando il robot a gestire con maggiore efficacia la varietà dei possibili ambienti interni che si può trovare ad affrontare. Ciò porrebbe le basi per raggiungere l'efficacia e la robustezza necessari a permettere il suo utilizzo in situazioni reali, consentendo a persone disabili o in età avanzata una vita il più possibile autonoma e sicura.

Bibliografia

[1] W. He, D. Goodkind, and P. Kowal, "An aging world: 2015," tech. rep., United States Census Bureau, 2016.

[2] P. C. of Advisors on Science and Technology, "Report to the president: Independence, technology, and connection in older age," tech. rep., 2016.

[3] S. R. Lord, C. Sherrington, H. B. Menz, and J. C. Close, *Falls in older people: risk factors and strategies for prevention*. Cambridge University Press, 2007.

[4] J.-A. Meyer and D. Filliat, "Map-based navigation in mobile robots:: Ii. a review of map-learning and path-planning strategies," *Cognitive Systems Research*, vol. 4, no. 4, pp. 283–317, 2003.

[5] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon," 2010.

[6] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pp. 146–151, IEEE, 1997.

[7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[8] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[9] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments," in *In the 12th International Symposium on Experimental Robotics (ISER*, Citeseer, 2010.

[10] S. Park and S. Hashimoto, "Autonomous mobile robot navigation using passive rfid in indoor environment," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 7, pp. 2366–2373, 2009.

[11] J.-B. Hayet, F. Lerasle, and M. Devy, "A visual landmark framework for mobile robot navigation," *Image and Vision Computing*, vol. 25, no. 8, pp. 1341–1351, 2007.

[12] Q. Li, J. A. Stankovic, M. A. Hanson, A. T. Barth, J. Lach, and G. Zhou, "Accurate, fast fall detection using gyroscopes and accelerometer-derived posture information," in *2009 Sixth International Workshop on Wearable and Implantable Body Sensor Networks*, pp. 138–143, IEEE, 2009.

[13] A. Williams, D. Ganesan, and A. Hanson, "Aging in place: fall detection and localization in a distributed smart camera network," in *Proceedings of the 15th ACM international conference on Multimedia*, pp. 892–901, ACM, 2007.

[14] P. Rashidi and A. Mihailidis, "A survey on ambient-assisted living tools for older adults," *IEEE journal of biomedical and health informatics*, vol. 17, no. 3, pp. 579–590, 2013.

[15] J. T. Perry, S. Kellog, S. M. Vaidya, J.-H. Youn, H. Ali, and H. Sharif, "Survey and evaluation of real-time fall detection approaches," in *2009 6th International Symposium on High Capacity Optical Networks and Enabling Technologies (HONET)*, pp. 158–164, IEEE, 2009.

[16] J. Boyle and M. Karunanithi, "Simulated fall detection via accelerometers," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 1274–1277, IEEE, 2008.

[17] U. Lindemann, A. Hock, M. Stuber, W. Keck, and C. Becker, "Evaluation of a fall detector based on accelerometers: A pilot study," *Medical and Biological Engineering and Computing*, vol. 43, no. 5, pp. 548–551, 2005.

[18] M. Popescu, Y. Li, M. Skubic, and M. Rantz, "An acoustic fall detector system that uses sound height information to reduce the false alarm rate," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 4628–4631, IEEE, 2008.

[19] R. Cucchiara, A. Prati, and R. Vezzani, "A multi-camera vision system for fall detection and alarm generation," *Expert Systems*, vol. 24, no. 5, pp. 334–345, 2007.

[20] A. Yazar, F. Erden, and A. E. Cetin, "Multi-sensor ambient assisted living system for fall detection," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'14)*, pp. 1–3, Citeseer, 2014.

[21] A. Tapus, M. J. Mataric, and B. Scassellati, "Socially assistive robotics," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, p. 35, 2007.

[22] J. Broekens, M. Heerink, and H. Rosendal, "Assistive social robots in elderly care: a review," *Gerontechnology*, vol. 8, no. 2, pp. 94–103, 2009.

[23] T. Tamura, S. Yonemitsu, A. Itoh, D. Oikawa, A. Kawakami, Y. Higashi, T. Fujimooto, and K. Nakajima, "Is an entertainment robot useful in the care of elderly people with severe dementia?," *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences*, vol. 59, no. 1, pp. M83–M85, 2004.

[24] S. Šabanović, C. C. Bennett, W.-L. Chang, and L. Huber, "Paro robot affects diverse interaction modalities in group sensory therapy for older adults with dementia," in *Rehabilitation Robotics (ICORR), 2013 IEEE International Conference on*, pp. 1–6, IEEE, 2013.

[25] M. E. Pollack, L. Brown, D. Colbry, C. Orosz, B. Peintner, S. Ramakrishnan, S. Engberg, J. T. Matthews, J. Dunbar-Jacob, C. E. McCarthy, *et al.*, "Pearl: A mobile robotic assistant for the elderly," in *AAAI workshop on automation as eldercare*, vol. 2002, pp. 85–91, 2002.

[26] F. Cavallo, M. Aquilano, M. Bonaccorsi, R. Limosani, A. Manzi, M. C. Carrozza, and P. Dario, "On the design, development and experimentation of the astro assistive robot integrated in smart environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 4310–4315, IEEE, 2013.

[27] D. Fischinger, P. Einramhof, K. Papoutsakis, W. Wohlkinger, P. Mayer, P. Panek, S. Hofmann, T. Koertner, A. Weiss, A. Argyros, *et al.*, "Hobbit, a care robot supporting independent living at home: First prototype and lessons learned," *Robotics and Autonomous Systems*, vol. 75, pp. 60–78, 2016.

[28] F. Palumbo, J. Ullberg, A. Štimec, F. Furfari, L. Karlsson, and S. Coradeschi, "Sensor network infrastructure for a home care monitoring system," *Sensors*, vol. 14, no. 3, pp. 3833–3860, 2014.

[29] S. Wang, S. Zabir, and B. Leibe, "Lying pose recognition for elderly fall detection," *Robotics: Science and Systems VII*, no. 1, pp. 345–353, 2012.

[30] M. Volkhardt, F. Schneemann, and H.-M. Gross, "Fallen person detection for mobile robots using 3d depth data," in *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3573–3578, IEEE, 2013.

[31] K. Nishi and J. Miura, "A head position estimation method for a variety of recumbent positions for a care robot," vol. 15, no. 210, pp. 157–158, 2015.

[32] J. Borenstein, H. R. Everett, L. Feng, and D. Wehe, "Mobile robot positioning-sensors and techniques," tech. rep., DTIC Document, 1997.

[33] M. Carraro, M. Antonello, L. Tonin, and E. Menegatti, "An open source robotic platform for ambient assisted living," *Artificial Intelligence and Robotics (AIRO)*, 2015.

[34] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[35] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1–4, IEEE, 2011.

[36] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[37] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, pp. 1–6, IEEE, 2013.

[38] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[39] S. Thrun, "Particle filters in robotics," in *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pp. 511–518, Morgan Kaufmann Publishers Inc., 2002.

[40] A. Doucet, N. De Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pp. 176–183, Morgan Kaufmann Publishers Inc., 2000.

[41] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," 2006.

[42] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[43] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, pp. 343–349, 1999.

[44] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[45] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[46] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[47] G. H. Dunteman, *Principal components analysis*. No. 69, Sage, 1989.

[48] D. Wolf, J. Prankl, and M. Vincze, "Enhancing semantic segmentation for robotics: The power of 3-d entangled forests," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 49–56, 2016.

[49] D. Wolf, J. Prankl, and M. Vincze, "Fast semantic segmentation of 3d point clouds using a dense crf with learned parameters," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4867–4873, IEEE, 2015.

[50] J. Papon, A. Abramov, M. Schoeler, and F. Wörgötter, "Voxel cloud connectivity segmentation - supervoxels for point clouds," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, (Portland, Oregon), June 22-27 2013.

[51] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: An evaluation of the state of the art," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 4, pp. 743–761, 2012.

[52] M. Carraro, M. Munaro, and E. Menegatti, "A powerful and cost-efficient human perception system for camera networks and mobile robotics," in *Intelligent Autonomous Systems 14*, Springer International Publishing, 2016.

[53] M. Carraro, M. Munaro, and E. Menegatti, "Cost-efficient rgb-d smart camera for people detection and tracking," *Journal of Electronic Imaging*, vol. 25, no. 4, pp. 041007–041007, 2016.

Ringraziamenti

Questa tesi rappresenta un importante traguardo della mia vita. Qui si conclude la mia carriera di studente, e ci tengo a cogliere l'occasione per ringraziare tutte le persone che mi hanno accompagnato in questo percorso di crescita.

Il primo e più sentito grazie lo dedico alla mia famiglia. Ai miei genitori Mario e Italietta, che mi hanno sempre sostenuto, incoraggiato e supportato economicamente, con tanto amore e pazienza. A Luca, fratello e compagno di avventure, con il quale ho condiviso i momenti più divertenti e felici della mia infanzia. A Lidia, sempre disponibile quando serve aiuto, sempre pronta a deliziarci con gustosi manicaretti. Al nonno Bruno, a cui voglio dedicare questa tesi.

Un grande grazie anche agli amici, con i quali ho condiviso gioie e dolori della vita studentesca, sempre presenti nel momento del bisogno e nei ricordi più belli che ho.

Voglio ringraziare infine tutti gli insegnanti che nel corso degli anni hanno saputo stimolare e mantenere viva in me la curiosità e la voglia di apprendere. In particolare, un sentito ringraziamento al relatore prof. Emanuele Menegatti, ai correlatori dott. Marco Carraro e dott. Morris Antonello ed in generale a tutto lo staff dell'IAS-LAB per avermi dato l'opportunità di avvicinarmi ed appassionarmi al mondo della robotica.