UNIVERSITÀ DEGLI STUDI DI PARMA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

UN SISTEMA DI PERCEZIONE 3D PER LA NAVIGAZIONE DI ROBOT MOBILI E L'INDIVIDUAZIONE DI OGGETTI

A 3D PERCEPTION SYSTEM FOR MOBILE ROBOT NAVIGATION AND OBJECT DETECTION

Relatore:
Chiar.mo Prof. Ing. STEFANO CASELLI

Correlatore:
Dott. Ing. DARIO LODI RIZZINI

Tesi di laurea di:
DAVIDE VALERIANI

21 MARZO 2013

Indice

Alla mia famiglia

Ringraziamenti

La tesi di laurea magistrale non è solamente la conclusione di un corso di laurea, ma segna la fine di una parte importante della vita di una persona, quella relativa alla sua formazione. Ecco perché con queste poche righe non voglio ringraziare solamente chi ha contribuito alla stesura di questa tesi ma, in generale, tutti coloro che, per un motivo o per l'altro, hanno contribuito alla mia formazione e reso questa mia fase della vita indimenticabile!

Innanzitutto, un sincero ringraziamento desidero rivolgerlo al mio correlatore, il Dott. Ing. Dario Lodi Rizzini, per la disponibilità dimostrata e per avermi seguito in modo puntuale nel corso di tutto il lavoro di tesi. Grazie anche al mio relatore, il Prof. Ing. Stefano Caselli, per i preziosi suggerimenti e per avermi accettato (sopportato?) come suo tesista anche in questa laurea magistrale.

Un grande ringraziamento va a tutti gli amici del laboratorio di Robotica e della Palazzina 1 (il numero non è casuale!), quasi una seconda famiglia in questi duri anni di studio, che non posso ringraziare singolarmente perché sono in troppi e sforerei dallo spazio concesso, ma un grazie particolare va a Fabio Oleari, sempre disponibile ad aiutarmi a risolvere i tanti problemi riscontrati durante lo sviluppo della mia tesi e vero punto di riferimento del laboratorio di robotica, Alessandro Costalunga, instancabile compagno di studi e di tante avventure in questa laurea magistrale, e Federico Parisi, per avermi insegnato, con i suoi count, quel pizzico di pragmaticità che, anche nella robotica, non guasta.

Anche Marco Cigolini, Isabella Salsi, Andrea Signifredi e Marco Patander meritano una menzione speciale come appartenenti, assieme ad Alessandro e Federico, al RedBeard Button, il team dell'Università di Parma con il quale ho condiviso otto mesi di lavoro in preparazione al Sick Robot Day 2012 e una spettacolare vittoria internazionale, che mi ha motivato a proseguire la specializzazione nel campo della robotica mobile.

Ne approfitto per ringraziare della disponibilità tutti i professori, i ricercatori, i dottorandi, gli assegnisti e i tecnici della Palazzina 1 e non solo, con i quali, in questi anni, ho avuto modo di confrontarmi sia per i miei studi che per la mia attività di rappresentante degli studenti in Consiglio di Corso di Laurea e, negli ultimi mesi, in Consiglio di Dipartimento e in Senato Accademico.

E infine, come sempre, un grazie particolare e doveroso alla mia famiglia tutta, in particolare ai miei genitori, per avermi sostenuto su più fronti, non solo in questi cinque anni di università ma, in generale, da quasi 25 anni a questa parte. Questa tesi è dedicata a voi.

Davide Valeriani

Nella vita non bisogna mai rassegnarsi, arrendersi alla mediocrità, bensì uscire da quella "zona grigia" in cui tutto è abitudine e rassegnazione passiva, bisogna coltivare il coraggio di ribellarsi.

Rita Levi Montalcini

Introduzione

La robotica mobile è quella branca della robotica che si occupa dei robot mobili e dei metodi che consentano la loro navigazione e l'interazione con l'ambiente che li circonda. La robotica mobile pone problemi differenti rispetto alla robotica industriale, focalizzata sui manipolatori. Per esempio, la scena inquadrata da un sensore montato su una base mobile tende a cambiare repentinamente con gli spostamenti del robot. Invece, un robot industriale montato su una base fissa, sebbene possa muovere i propri giunti, ha uno *spazio di lavoro* limitato, definito come la regione descritta dall'origine della terna utensile quando ai giunti del manipolatore si fanno eseguire tutti i moti possibili. Nella robotica mobile, invece, lo spazio di lavoro è virtualmente infinito e, pertanto, viene introdotto lo *spazio delle configurazioni*, definito come il numero di parametri che servono per esprimere una posa del robot. Ne deriva che la percezione degli ostacoli è un forte elemento di diversità rispetto alla robotica industriale.

Così, mentre nella robotica fissa è possibile movimentare il manipolatore senza l'ausilio di sensori che gli garantiscano la possibilità di *percepire* eventuali ostacoli, in quanto lo spazio di lavoro è limitato e, spesso, statico, nella robotica mobile la percezione rappresenta una funzione fondamentale del robot. La percezione è funzionale principalmente alla navigazione, sia perché permette la movimentazione in sicurezza sia perché consente, all'occorrenza e quando opportunamente gestita ed integrata nel tempo in un'unica rappresentazione, di pianificare le azioni da eseguire. Inoltre, il dato sensoriale può essere impiegato per estrarre informazioni dall'ambiente, identificare gli oggetti, interagire con le persone e, più in generale, intepretare, a vari livelli, l'ambiente circostante.

I sensori tradizionalmente impiegati nella robotica mobile sono tipicamente quelli di prossimità, come sonar, laser scanner e bumper. In genere, tali sensori

(a) Robot industriale (b) Robot mobile

Figura 1: Esempi di robot.

sono attivi e misurano continuamente la distanza tra il sensore ed il primo ostacolo incontrato lungo il cammino di propagazione del segnale stesso, misurando il *tempo di volo*, ovvero il tempo che impiega il segnale per andare dal laser all'oggetto e tornare indietro, dopo essere stato riflesso. Nel caso del laser scanner, i dati di prossimità riguardano un unico piano di scansione (laser scanner planari) o alcuni piani di scansione, e il dato fornito è molto preciso e semplice da utilizzare.

Tuttavia, la presenza di eventuali ostacoli su piani diversi dal piano di scansione del laser scanner non verrebbe rilevata, il che potrebbe provocare collisioni del robot con un ostacolo più o meno delicato. D'altra parte, l'utilizzo di un laser scanner per ogni piano interessato dal percorso del robot o di un laser scanner 3D sarebbe una soluzione molto costosa e, quindi, non adatta ad ogni tipo di applicazione. Inoltre, il laser scanner fornisce le sole informazioni di distanza e posizione di un oggetto, limitando la possibilità di identificare gli oggetti, ad esempio, in base al colore.

Nasce quindi la necessità di utilizzare un altro tipo di sensore, più economico, per garantire al robot la sicurezza di muoversi nello spazio libero e fornirgli la possibilità di identificare oggetti in base alle loro caratteristiche intrinseche. Il sensore più economico con queste caratteristiche risulta essere una telecamera a colori, quale può essere anche una semplice webcam per computer acquista-

bile per poche decine di euro, che fornisce informazioni meno precise del laser scanner, ma che può colmare la mancanza di informazioni di cui si parlava in precedenza.

L'utilizzo esclusivo di una telecamera per la percezione dell'ambiente (visione *mono*) consente di acquisire dati sensoriali caratterizzati da una posizione (x,y) e da una terna di parametri (r,g,b) rappresentanti il colore del pixel acquisito. Questa ulteriore informazione consente di aumentare il numero di applicazioni della percezione, ad esempio utilizzando il colore per identificare gli oggetti. Tuttavia, una semplice telecamera è un sensore in due dimensioni e, sebbene fornisca l'informazione di colore, produrrebbe comunque piani ciechi pericolosi per la navigazione.

Negli ultimi anni, largo spazio è stato dato, in ambito di ricerca, alla percezione 3D, anche grazie allo sviluppo di sensori tridimensionali a basso costo. Si pensi, a titolo esemplificativo, al Kinect di Microsoft, una periferica inizialmente nata per la console Xbox 360 e dotata di una telecamera RGB, un array di microfoni per la calibrazione e un doppio sensore di profondità a raggi infrarossi, composto da un proiettore a infrarossi e da una telecamera sensibile alla stessa banda. Grazie al suo basso costo (un centinaio di euro), numerosi sono stati i gruppi di ricerca che lo hanno utilizzato per la percezione 3D da applicare alla navigazione di robot mobili.

Un altro sensore utile alla percezione tridimensionale è il laser scanner 3D, utilizzato, ad esempio, dalla Google Car, un'automobile a guida autonoma tuttora in fase di sviluppo. Questo sensore consente di ottenere una nuvola di punti tridimensionale contenente le coordinate (x,y,z) dei pixel della scena. Il principio di funzionamento è simile ai laser scanner planari illustrati in precedenza. Tuttavia, il costo di un sensore di questo tipo è piuttosto alto (alcune decine di migliaia di euro), rendendolo inadatto per molte applicazioni.

La soluzione più economica per la percezione 3D dello spazio è lo sviluppo di un sistema stereo di telecamere a colori, il cui utilizzo è in crescita in ambito accademico e industriale per molteplici applicazioni. La testa stereo è composta da due telecamere poste su uno stesso asse e a una distanza d, che acquisiscono la stessa scena da due angolazioni diverse. Con l'applicazione di opportune leggi geometriche, è possibile ottenere la cosiddetta *visione stereo*, che si basa sull'i-

dentificazione di punti omologhi nelle due immagini acquisite e sul successivo calcolo dell'informazione di profondità del punto. Questo consente, con poche decine di euro, di acquistare due webcam e ottenere un sensore tridimensionale che fornisca anche l'informazione di colore.

In questo lavoro di tesi si è sviluppato un sistema di percezione tridimensionale di un ambiente per un duplice scopo: la navigazione di un robot mobile in sicurezza e l'individuazione di oggetti. La costruzione di un sistema di percezione 3D pone anche le basi per ulteriori utilizzi futuri, che potrebbero fornire funzioni aggiuntive al robot.

La scelta del sensore su cui basare la percezione tridimensionale non è stata semplice. Tralasciando il laser scanner 3D per ovvi motivi di costo, la scelta poteva ricadere su tre tipi di sensore, già presentati in precedenza: il Kinect, la telecamera stereo oppure il tilted laser scanner.

Il Kinect poteva essere una buona scelta, in virtù del suo basso costo e dell'alto numero di sensori integrati, tuttavia le sue prestazioni peggiorano notevolmente in ambienti esterni, il che rappresenta una notevole limitazione nell'applicazione alla navigazione di robot mobili. Infatti, il sistema di navigazione realizzato potrebbe essere impiegato sia su robot da *indoor* che su robot da *outdoor* (si pensi, ad esempio, ai tagliaerba automatici).

Il sistema stereo di telecamere rappresenta anch'esso una soluzione economica per la percezione, oltre ad essere notevolmente sviluppato in ambito di ricerca. La sua principale limitazione riguarda la bassa densità della nuvola di punti generata dall'algoritmo di visione stereo che, basandosi sull'immagine di disparità, presenta numerose zone cieche.

Infine, il tilted laser scanner consiste nella movimentazione di un laser scanner planare lungo l'asse z al fine di acquisire dati sensoriali provenienti da diversi piani di scansione ed ottenere, quindi, una percezione 3D, seppur parziale. Tuttavia, questa soluzione presenta due problemi: la necessità di un attuatore che muova il laser scanner in modo automatico da montare sul robot, con relativo driver, e la mancanza di informazioni di colore della point cloud acquisita. Infatti, il laser scanner restituisce solamente un'informazione di distanza del robot dagli ostacoli che, sebbene molto precisa, non è sufficiente per l'identificazione completa degli oggetti. Inoltre, la necessità di un attuatore che movimenti il laser complica note-

volmente il sistema, dovendo realizzare un carrello motorizzato che mantenga il laser perfettamente orizzontale durante il moto, oltre a dover studiare la velocità di movimentazione ideale. Infatti, una velocità troppo alta rischierebbe di ridurre drasticamente il numero di piani di scansione analizzati, vanificando lo sforzo di un attuatore per il laser, mentre una velocità troppo bassa ridurrebbe la velocità di aggiornamento del dato sensoriale e, quindi, della mappa di navigazione costruita, aumentando la probabilità di non percepire ostacoli che invece si sono inseriti nella scena da poco tempo.

La soluzione adottata è stata quindi la combinazione di una testa stereo di telecamere a colori, che consenta di avere un'informazione sensoriale tridimensionale e dotata di colore, e un laser scanner planare, che completi la percezione in quelle zone cieche che costituivano il principale problema della visione stereo. La fusione dei dati sensoriali, infatti, risulta vitale per poter godere dei vantaggi di entrambi i sensori e ridurre al minimo l'influenza degli svantaggi. Tali dati sono stati anche utilizzati per l'individuazione di oggetti, funzione anch'essa legata alla navigazione in quanto consente al robot di diventare più consapevole e autonomo nell'impostazione del *goal*, ovvero la posizione verso la quale muoversi.

L'organizzazione della tesi cerca di dare conto del lavoro svolto e dei risultati ottenuti, confrontandoli anche con iniziative di ricerca simili condotte a livello internazionale.

Nel primo capitolo si analizzeranno i principali lavori presenti in letteratura, con particolare riferimento al caso preso in esame in questa tesi, ovvero la navigazione di un robot mobile in un ambiente dinamico e l'individuazione di oggetti.

Nel capitolo 2 si presenteranno gli strumenti hardware e software utilizzati. Una parte importante del lavoro, infatti, è stata l'integrazione e la conseguente configurazione di librerie software già sviluppate che, talvolta, è stato necessario adattare e completare. Dal punto di vista hardware, si è resa necessaria la progettazione e la realizzazione di appositi strumenti che consentissero di velocizzare il processo di configurazione del robot.

Nel capitolo 3 si illustrerà nel dettaglio l'architettura del sistema realizzato, presentando la pipeline di elaborazione e analizzando, con adeguato approfondimento, ogni operazione effettuata dai nodi componenti il sistema.

Nel capitolo 4 si presenteranno i risultati derivanti da alcune prove sperimentali condotte presso il laboratorio di Robotica dell'Università di Parma, al fine di collaudare la reale applicazione del sistema realizzato e presentare i punti di forza e i problemi riscontrati.

Infine, si trarranno le conclusioni e si proporranno alcuni possibili sviluppi futuri per il miglioramento di questo lavoro e l'integrazione di nuove funzionalità al sistema complessivo.

Capitolo 1

La percezione tridimensionale nella robotica

L'impiego della percezione tridimensionale nell'ambito della robotica si è notevolmente diffuso ed ampliato negli ultimi anni, grazie anche agli innovativi metodi e algoritmi per l'elaborazione delle immagini di profondità sviluppati dai ricercatori.

In questo capitolo si presenterà una panoramica dei principali lavori presenti in letteratura sulle tematiche affrontate in questo lavoro di tesi. Per ragioni di opportunità, nel presentare lo stato dell'arte è stato necessario focalizzare l'attenzione su temi e lavori specifici della percezione 3D su base mobile. In particolare, sono stati individuati quattro ambiti principali:

- le **strutture dati** maggiormente utilizzate per rappresentare ambienti tridimensionali;
- metodi e algoritmi per la **percezione 3D** da utilizzare per la navigazione;
- l'**identificazione di generici oggetti** all'interno di macro scene, senza particolari conoscenze pregresse;
- la **pianificazione dei punti di osservazione di un oggetto**, al fine di costruire una vista più ricca dell'oggetto da diverse angolazioni.

1.1 Strutture dati per il 3D

Il tipo di rappresentazione dei dati influenza notevolmente la gestione dei comportamenti di navigazione del robot. In tal senso, nel mondo tridimensionale, la letteratura fornisce vari tipi di strutture dati per la gestione delle informazioni di percezione che derivano da sensori quali laser scanner e telecamere.

Nel mondo planare, la struttura dati più utilizzata è la *gridmap 2D*, ovvero una matrice bidimensionale in cui ogni cella contiene un valore che rappresenta lo stato di libero, occupato o sconosciuto di quella particolare posizione. Risulta pertanto naturale estendere questo concetto al 3D, con *gridmap 3D*, composte da celle disposte nello spazio, che prendono il nome di point cloud (letteralmente, nuvole di punti).

Una prima struttura dati di base, derivante direttamente dai tipi di dato standard, può essere una point cloud intesa come vettore di punti, caratterizzati da tre valori che indichino la posizione (x, y, z) e tre valori che indichino, ove presente, il colore del pixel (r, g, b). Tuttavia, questa rappresentazione risulta piuttosto inefficiente e laboriosa da gestire, in quanto non ordinata e quindi difficoltosa da utilizzare nella ricerca di punti di vicinato, operazione molto comune nell'elaborazione di immagini tridimensionali.

Una delle prime strutture dati introdotte appositamente per rappresentare dati tridimensionali è stata la *voxel grid* [1], una griglia di celle cubiche di egual dimensione che consente di discretizzare l'immagine acquisita, riducendo la mole di dati memorizzata a seconda del lato delle celle (*risoluzione*). Tuttavia, questo ancora non è sufficiente, in quanto una bassa risoluzione (celle grandi) non consente di rappresentare fedelmente la scena, mentre un'alta risoluzione (celle piccole) aumenta esponenzialmente l'occupazione in memoria, specie nel tridimensionale. Infatti, la griglia andrebbe comunque inizializzata alla dimensione del bounding box che racchiude tutta l'area da acquisire (scena), anche se poi molti spazi rimarrebbero scoperti a causa delle zone cieche causate dalla stereovisione.

In [2] è stata introdotta una potente e completa libreria per l'elaborazione di point cloud. Dal punto di vista delle strutture dati, essa introduce il tipo di dato *pcl::PointCloud* che integra al suo interno informazioni sulla dimensione, l'origine dei dati, l'orientamento del sensore e il sistema di riferimento. Grazie a questa

più efficiente struttura dati, è possibile gestire point cloud composte da centinaia di migliaia di punti che si aggiornano più volte al secondo. In [3] è presente la documentazione integrale della libreria.

Nel caso di elaboratori non troppo potenti, la memorizzazione delle tre informazioni di dimensione di tutti i punti nello spazio può risultare insostenibile. In [4] viene proposta una griglia 2.5D (*elevation map*), ovvero una griglia bidimensionale che consente di memorizzare, in ogni cella, un unico valore di altezza, ottenuto proiettando una retta perpendicolare al piano della griglia e misurando la distanza dal primo ostacolo incontrato. Questo approccio consente di ridurre la mole di dati memorizzata, ma non consente di memorizzare informazioni relative a tutte le celle libere e agli ostacoli posti uno sopra all'altro.

Un altro approccio alla memorizzazione di informazioni sensoriali è presentato in [5] e si basa sugli *octree*, strutture dati gerarchiche in cui ogni nodo rappresenta lo spazio contenuto in una cella cubica (*voxel*) come in [1], ma con l'aggiunta della gerarchia ad albero che consente di tagliare rami ed efficientare la costruzione.

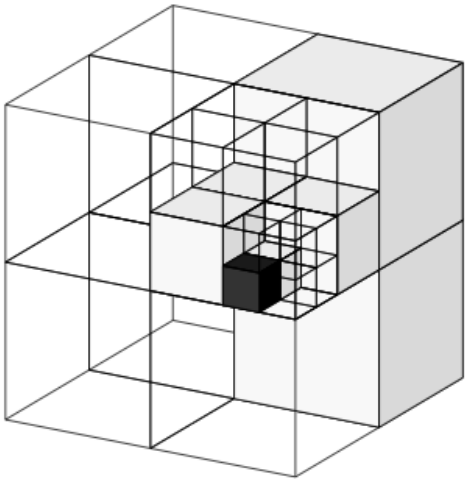


Figura 1.1: Esempio di *octree* in cui le celle bianche rappresentano quelle vuote, le celle nere quelle occupate e le celle grigie quelle parzialmente occupate [5]

.

In virtù della struttura ad albero, gli *octree* si adattano bene alla modellizzazione di proprietà booleane come l'occupazione di un *voxel*. Se un *voxel* viene

rilevato occupato, il nodo corrispondente dell'albero viene inizializzato con un valore preciso; se invece viene rilevato come libero, il nodo corrispondente viene inizializzato all'altro valore booleano. I nodi non inizializzati rappresentano lo spazio sconosciuto. Il vantaggio principale di una struttura ad albero riguarda la possibilità di tagliare i nodi i cui figli risultino tutti occupati o tutti liberi, per ridurre la quantità di dati da memorizzare.

1.2 Percezione 3D per la navigazione

Un robot che si muove nello spazio necessita di una percezione tridimensionale, per poter evitare ostacoli posizionati a qualunque coordinata (x, y, z). Le modalità di percezione tridimensionale ai fini della navigazione sono le più svariate.

Un primo tipo di approccio [6] consiste nell'utilizzo di sensori bidimensionali, quali laser scanner planari, per la progressiva costruzione di una mappa tridimensionale, ottenuta facendo spostare il laser scanner 2D su più piani (*tilting laser scanner*). Infatti, un laser scanner consente di ottenere informazioni sulla distanza degli ostacoli che insistono su un determinato piano, detto *piano di scansione*, offrendo quindi una percezione bidimensionale. Facendo spostare il laser scanner lungo la terza dimensione e accumulando i dati ottenuti, è possibile costruire una point cloud tridimensionale della scena. Tuttavia, l'accumulo di dati sensoriali acquisiti in momenti diversi è funzionale solamente per scene statiche e robot fermo; viceversa, i dati acquisiti non corrisponderebbero alla medesima scena e si potrebbero avere problemi di falsi positivi, che andrebbero poi filtrati.

Una volta acquisita la point cloud, Marder-Eppstein *et al.* [6] la utilizzano per costruire una *voxel grid*, ovvero una griglia di occupazione basata su celle cubiche di egual dimensione, ciascuna delle quali può essere in uno dei tre stati: occupato, libero, sconosciuto. Un esempio è mostrato in fig. 1.2.

Ogni colonna della *voxel grid* viene poi proiettata in una griglia bidimensionale, attribuendole un costo proporzionale alla distanza dell'ostacolo dall'impronta lasciata dal robot. Se tale costo supera una certa soglia, significa che il robot potrebbe urtare l'ostacolo e, pertanto, il pianificatore di traiettorie considererà quella posizione come occupata.

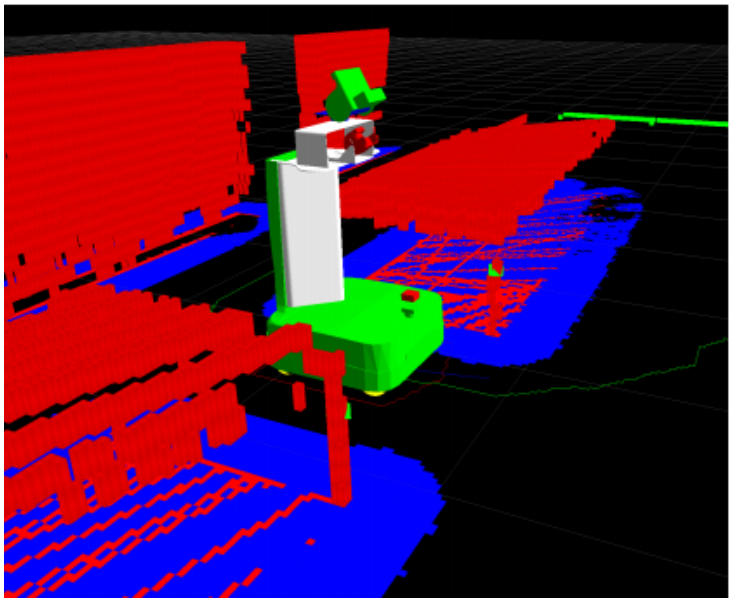


Figura 1.2: Esempio di *voxel grid* costruita dall'implementazione di [6].

Un altro approccio [7] utilizza veri e propri sensori tridimensionali, quale può essere un laser *Velodyne*, per l'acquisizione di una point cloud e la costruzione di una mappa locale da sovrapporre ad una mappa globale preimpostata. La mappa locale, dopo le relative elaborazioni di filtraggio, analogamente al lavoro precedente viene proiettata in una griglia di occupazione bidimensionale per essere poi utilizzata per la navigazione. In fig. 1.3 è mostrato il processo di costruzione della mappa bidimensionale.

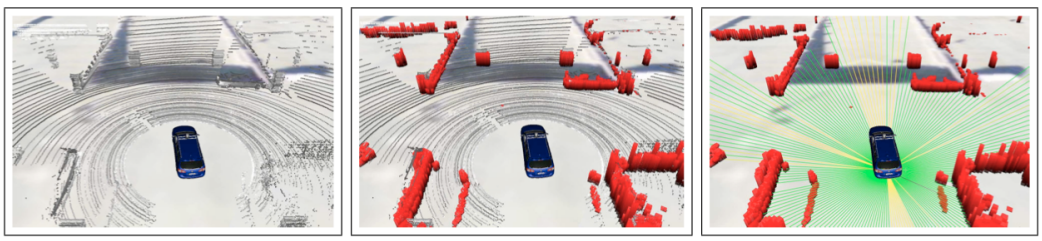


Figura 1.3: Processo di costruzione della mappa [7]. Dopo aver analizzato i dati sensoriali del *Velodyne* (immagine a sinistra), gli ostacoli vengono discretizzati in celle da 15 cm (in rosso nell'immagine centrale) che sono poi utilizzate per generare una scansione virtuale bidimensionale (immagine a destra) utile per aggiornare la mappa (i raggi gialli indicano le direzioni senza ostacoli).

Tuttavia, visto l'alto contenuto tecnologico, i laser scanner tridimensionali quali il Velodyne risultano piuttosto costosi e, quindi, inadatti alla maggior parte delle applicazioni.

Per la verifica efficiente delle collisioni nel tridimensionale, è possibile utilizzare strutture gerarchiche ad albero [8], che suddividono lo spazio 3D in tante sfere di raggio variabile per semplificare il riconoscimento di eventuali collisioni tra il robot e gli oggetti. Tuttavia, la ricerca delle collisioni nello spazio 3D risulta ancora piuttosto inefficiente dal punto di vista computazionale.

Un'ulteriore possibilità per la percezione tridimensionale è quella proposta da Hornung *et al.* [9], che cerca di unire i vantaggi di semplicità computazionale derivanti dalla proiezione di un ambiente 3D in una mappa di occupazione bidimensionale con i vantaggi di completezza di informazione della percezione tridimensionale. Il lavoro si basa su un sistema di percezione che utilizza una rappresentazione basata su *octree*, illustrati nella sezione 1.1, per la costruzione di una mappa di occupazione tridimensionale.

L'innovazione introdotta da questo lavoro riguarda il secondo passaggio, ovvero quello che negli altri approcci presentati in precedenza si preoccupava di proiettare nel piano bidimensionale la nuvola di punti acquisita dai sensori. In quel caso, non era possibile tenere in considerazione tutte le pose ammissibili del robot, ovvero quelle che non prevedono urti, in quanto la proiezione della sua impronta nel piano 2D risulta più ingombrante dell'effettiva occupazione del robot nel piano 3D. Questa limitazione è molto forte nel caso di robot mobili piuttosto grossi, quali il PR2, tenuti a muoversi in uno spazio altamente pieno di ostacoli, in cui ogni singola posa disponibile risulta preziosa.

La soluzione intrapresa in questo lavoro consiste nel proiettare l'impronta del robot su differenti livelli, corrispondenti alle diverse forme delle parti del robot, facendo risparmiare il tempo di elaborazione dovuto alla ricerca di collisioni nello spazio 3D durante la pianificazione quando nessuna collisione viene trovata nei vari livelli. I livelli di scansione sono tre: la base, il dorso con la testa e i bracci con gli eventuali oggetti attaccati. L'ambiente viene rappresentato, quindi, suddiviso in tre livelli, ognuno dei quali è associato ad una delle precedenti parti del robot (fig. 1.4). La mappa di navigazione viene poi aggiornata ad ogni iterazione, sulla base delle nuove acquisizioni sensoriali.

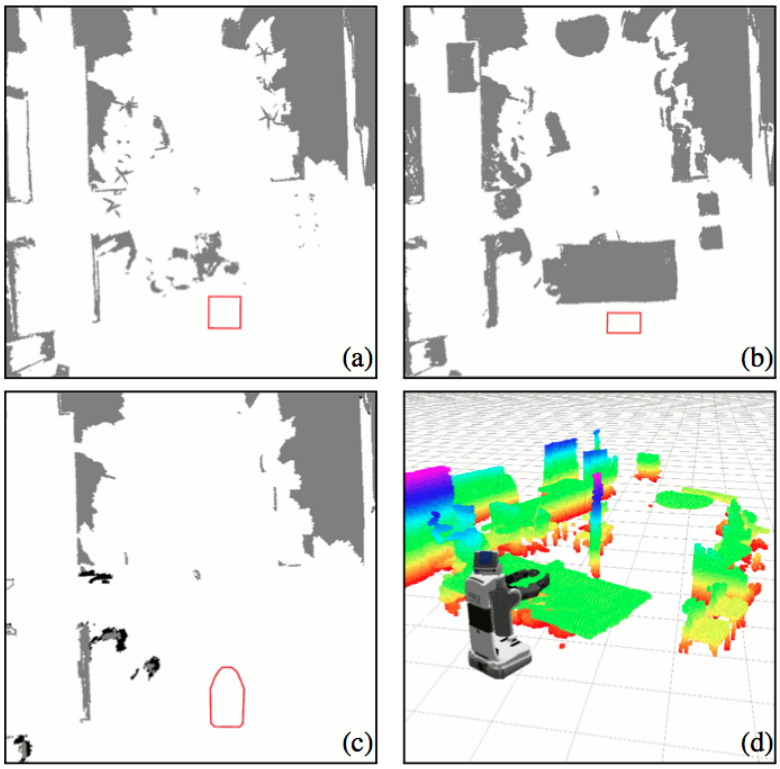


Figura 1.4: Proiezione planare dello spazio tridimensionale sui tre diversi livelli, base (a), dorso (b), bracci (c), e griglia di occupazione tridimensionale completa (d), derivante dalla rappresentazione basata su *octree*.

Una volta costruita la mappa, Hornung *et al.* la utilizzano per il comportamento di navigazione verso il goal. Dapprima, viene utilizzato un pianificatore globale per costruire un percorso per raggiungere il goal nello spazio delle posizioni e dell'orientamento (x, y, θ). Tale piano viene poi elaborato da un pianificatore locale, che utilizza i dati sensoriali provenienti da un laser per localizzare il robot utilizzando il metodo Monte Carlo. La localizzazione utilizza una mappa statica, contenente tutti gli oggetti statici dell'ambiente, combinata con il dato odometrico.

La metodologia utilizzata nell'ambito di questo lavoro di tesi, tuttavia, fonde i dati sensoriali provenienti da due diversi tipi di sensori per costruire una point cloud tridimensionale, che poi viene proiettata su uno spazio bidimensionale e utilizzata per la navigazione. Infatti, in virtù del basso ingombro del robot utilizzato e della sua forma convessa e simile a un parallelepipedo, i vantaggi dell'utilizzo

di un sistema multilivello si assottigliano, non rendendo più conveniente l'aumento di complessità di elaborazione. L'approccio adottato, inoltre, utilizza sensori a basso costo, superando gli svantaggi di [7].

1.3 Identificazione di oggetti dentro macro scene

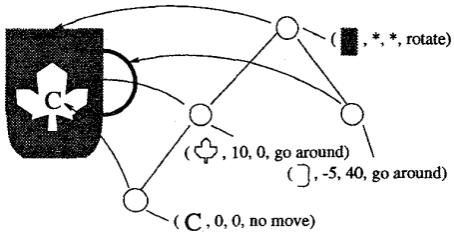
Lo scopo di questa funzione è l'identificazione di oggetti presenti in un'intera scena. Infatti, la maggior parte dei lavori presenti in letteratura si concentra sull'identificazione ed il riconoscimento di oggetti presenti in un particolare piano della scena (ad esempio, sopra ad un tavolo), di una certa dimensione e forma, spesso utilizzando manipolatori fissi.

Ciò che invece interessa maggiormente nell'ambito di questo lavoro è l'identificazione di oggetti da parte di un robot mobile. Occorre quindi tenere in considerazione la dinamicità della scena e considerarla al completo.

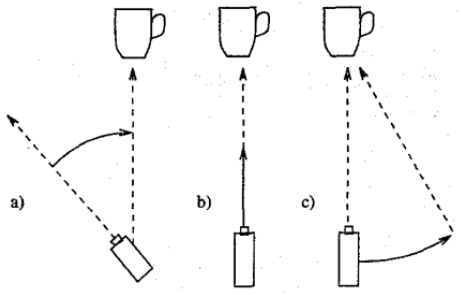
In letteratura, ci sono due grandi categorie di riconoscitori di oggetti: quelli che usano insiemi di *features* e quelli che usano la forma geometrica dell'oggetto. Di seguito verranno mostrati i principali lavori di entrambe le categorie.

Uno dei primi lavori di identificazione di oggetti da sensori posizionati su robot mobile è quello di Gvozdjak e Ze-Nian Li [10], che organizza i modelli degli oggetti da ricercare utilizzando una struttura ad albero in cui la radice rappresenta le caratteristiche grossolane e le foglie i particolari. Ogni nodo rappresenta una parte dell'oggetto ed è etichettato con una quadrupla contenente il suo *key aspect*, (la forma dell'oggetto), le sue coordinate (x,y) nel piano immagine ed il tipo di movimento da far compiere al robot per identificarlo, come mostrato in fig. 1.5(a).

Il primo passo dell'elaborazione consiste nell'identificare grossolanamente gli oggetti della scena che soddisfano determinati parametri di ricerca, utilizzando la trasformata di Hough generalizzata [11]. Successivamente, la telecamera e/o il robot vengono mossi per raggiungere una posizione che consenta di acquisire i dettagli dell'oggetto. I moti possibili, mostrati in fig. 1.5(b), sono la rotazione della telecamera rispetto ai propri assi, il moto rettilineo del robot per avvicinarsi o allontanarsi dall'oggetto identificato e il moto del robot attorno all'oggetto. Durante questi moti, il sistema cerca di mantenere l'oggetto al centro della scena, combinando il moto del robot con quello della telecamera.



(a) Struttura dell'albero di una tazza con una foglia e una lettera C disegnata sul corpo



(b) Tipi di moto: rotazione della telecamera (a), moto verso l'oggetto (b), moto attorno all'oggetto (c)

Figura 1.5: Passi di elaborazione di [10].

Ad ogni passo è effettuato un confronto tra le caratteristiche grossolane e i particolari dell'oggetto trovato con le quelle dell'oggetto da ricercare. Se entrambi i tipi di confronto vanno a buon fine, l'oggetto viene identificato; in caso contrario, si ricomincia da capo la ricerca.

Ekvall *et al.* [12] utilizzano una telecamera *pan-tilt-zoom*, ovvero in grado di muoversi e zoomare su particolari, per identificare gli oggetti presenti in una scena e inserirli all'interno di una mappa di navigazione. Il riconoscimento degli oggetti viene effettuato sulla base di un database di modelli, costruito in un'opportuna fase di *training* che consiste nel far acquisire al robot un'immagine della scena senza l'oggetto e una con l'oggetto, procedendo poi alla sottrazione delle due immagini e ad alcune operazioni morfologiche (erosione, espansione, erosione) per isolare l'oggetto specifico dallo sfondo.

Una volta che l'oggetto è stato segmentato correttamente, occorre inserirlo nel database con una rappresentazione adatta alla veloce indicizzazione dello stesso. Per questo, dopo aver estratto alcune caratteristiche visuali dell'oggetto, si utilizza un Receptive Field Cooccurrence Histogram (RFCH) [13], che consente di catturare informazioni aggiuntive oltre alle semplici proprietà geometriche di un istogramma classico.

Terminata la fase di training, in fase di esecuzione il robot osserva l'ambiente e scansiona l'immagine acquisita con una finestra di ricerca per trovare possibili candidati ad essere l'oggetto voluto. Ad ogni step, viene costruito un RFCH e

confrontato con l'RFCH dell'oggetto da ricercare, assegnando al candidato un voto proporzionale alla somiglianza. Se questo voto supera una certa soglia, il candidato viene inserito in un elenco di ipotesi, che vengono poi valutate con SIFT [14] per l'effettivo riconoscimento, utilizzando un'immagine ripresa ad una distanza più vicina all'oggetto mediante lo zoom della telecamera.

Anche [15] tratta l'identificazione di oggetti, utilizzando la *spectral residual saliency* calcolata come somma delle mappe ottenute rispetto all'intensità, alla differenza tra rosso e verde e a quella tra giallo e blu. La mappa risultante viene poi elaborata dal detector *Maximally Stable Extremal Region* (MSER) per identificare le regioni candidate ad essere l'oggetto ricercato.

In [16] l'identificazione di oggetti viene effettuata mediante confronto con gli oggetti presenti in un database, creato in fase di addestramento, tenendo conto di dimensione, colore e forma degli oggetti e calcolando χ^2 e distanza euclidea per il confronto di similarità. Il sistema mette anche a disposizione un'interfaccia grafica (GUI) per agevolare l'inserimento delle caratteristiche dell'oggetto da ricercare all'utente.

Per ogni oggetto del database sono memorizzate varie viste, per ciascuna delle quali viene calcolato un istogramma Hue vs. Saturation (H-S) a sua volta suddiviso in 100 blocchi di 10 cm di lato. Per ottenere una rappresentazione compatta di ogni oggetto, il vettore di istogrammi viene clusterizzato utilizzando *k-means clustering*. La rappresentazione matriciale dell'istogramma risultante da questa prima fase viene memorizzata in un file XML, che viene di volta in volta ricaricato nei passaggi successivi.

Dopo la fase di training, quindi, il sistema acquisisce un'immagine della scena, ne estrae i candidati ad essere l'oggetto ricercato e per ciascuno calcola l'istogramma bidimensionale H-S. A questo punto, gli istogrammi delle varie viste (vicine e lontane) di una stessa immagine candidata ad essere l'oggetto andrebbero confrontati con tutti i 100 istogrammi rappresentanti l'oggetto ricercato, presenti nel training set. Tuttavia, per ridurre l'onerosità di questa operazione, viene calcolato un unico istogramma integrale, dato dalla semplice formula seguente:

$$H_{x_2,y_2} - H_{x_2,y_1} - H_{x_1,y_2} + H_{x_1,y_1} \tag{1.1}$$

dove $H_{x,y}$ sono gli istogrammi dei punti angolari dell'immagine candidata estratta.

Tale istogramma integrale viene confrontato con quelli dell'oggetto da cercare, mediante l'operazione di *histogram intersection (HI)*. Se il punteggio risultante da tale operazione (espresso in percentuale) è più grande del 75%, viene incrementata un'apposita mappa di confidenza, che verrà poi analizzata per ottenere due metriche: la confidenza media (AC), calcolata come il valor medio dei pixel della regione interessata, e il rapporto di matching (MR), calcolato come rapporto tra gli istogrammi che hanno avuto un punteggio HI superiore al 75% e i 100 istogrammi del training set. Entrambe le metriche consentono di fornire una funzione di verosimiglianza tra l'oggetto cercato e quello identificato.

Nalpanditis *et al.* [17] lavorano, invece, con un sistema senza conoscenza a priori, privilegiando l'aspetto di segmentazione a quello di classificazione di oggetti. In fig. 1.6 è possibile vedere la pipeline di elaborazione dell'algoritmo sviluppato.

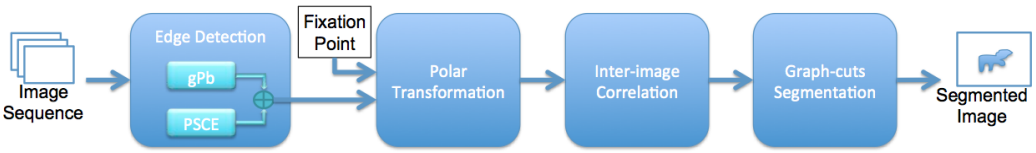


Figura 1.6: Pipeline di elaborazione dell'algoritmo proposto [17].

Il primo passo riguarda l'estrazione dei contorni dall'immagine acquisita. Per un'estrazione più robusta vengono combinati due differenti algoritmi: il *Globalized Probability of Boundary* (gPb) e il *Perceptually Salient Contours Enhancement* (PSCE). Successivamente, l'immagine viene trasformata da coordinate cartesiane (x,y) a coordinate polari (ρ,θ), dopo aver fissato la posizione del polo.

Vero cuore dell'algoritmo è il calcolo della correlazione tra le informazioni di contorno delle immagini acquisite consecutivamente, che viene effettuata utilizzando un'operazione di dilatazione della scala di grigi per ogni angolo θ dell'immagine in scala di grigi S_θ, dove S_θ rappresenta una fetta verticale del volume 3D contenente i contorni.

Infine, viene applicato l'algoritmo *graph cut* per produrre l'immagine segmentata di uscita.

1.4 Pianificazione dei punti di osservazione

Per il corretto riconoscimento di un oggetto, dopo la sua identificazione, occorre assegnarlo ad una particolare categoria. Tuttavia, la maggior parte delle immagini di addestramento dei classificatori, contengono solamente poche viste di un oggetto. Si rende quindi necessario accumulare più viste dell'oggetto identificato, pianificando il moto del robot affinché acquisisca immagini diverse dello stesso oggetto e le memorizzi in un'apposita struttura dati: un vettore di immagini riferite tutte allo stesso oggetto oppure un'unica point cloud 3D dell'oggetto.

In [15] il primo passo per svolgere questa funzione è la memorizzazione della posizione dell'oggetto nell'immagine come annotazione della mappa, relativamente alla posa del robot in un determinato istante. Oltre alla posizione viene memorizzata la vista dell'oggetto in un istogramma angolare suddividendola in 36 blocchi, ciascuno dei quali memorizza le informazioni relative a 10°.

Successivamente, occorre impostare il *goal* del robot per ottenere una vista diversa dell'oggetto. Ogni oggetto vota quindi una cella della mappa di occupazione tale che:

1. il robot sia in grado di acquisire un'immagine dell'oggetto;
2. la cella sia raggiungibile dal robot;
3. la vista dell'oggetto non sia già stata acquisita.

L'elenco delle viste può poi essere utilizzato in vari modi, tra cui il confronto dell'oggetto identificato dal robot con tutte le viste ottenute nella fase di training di tutti gli oggetti.

Foissotte *et al.* [18] utilizzano una particolare versione dell'algoritmo *Next-Best-View* per pianificare il moto di un robot umanoide attorno ad un oggetto. Il cuore di questo lavoro sta nell'utilizzo di un'innovativa formula matematica che consente di esprimere numericamente l'ampiezza delle aree di cui non si hanno ancora a disposizione dati sensoriali (*spazio sconosciuto*), tenendo in considerazione le occlusioni tra voxel contenenti dati e voxel con dati mancanti.

Anche [16] pianifica il moto del robot affinché possa vedere l'oggetto da più angolazioni. In particolare, il primo passo è acquisire tante immagini di un ambiente affinché il campo di vista possa spaziare su 230°. Una volta entrato in

un ambiente, dopo aver creato un sistema di coordinate posizionato in $(0,0)$ con orientamento pari a $0°$, il robot ruota di $90°$ in senso antiorario e cattura un'immagine. Successivamente, il robot ruota iterativamente di $45°$ in senso orario e cattura un'immagine, finché non arriva all'orientazione di $+90°$ rispetto alla posizione originale. Questo consente di acquisire un totale di 5 immagini della scena, come mostrato in fig. 1.7, e, grazie al campo di vista della telecamera pari a $50°$, di coprire un angolo di $230°$.

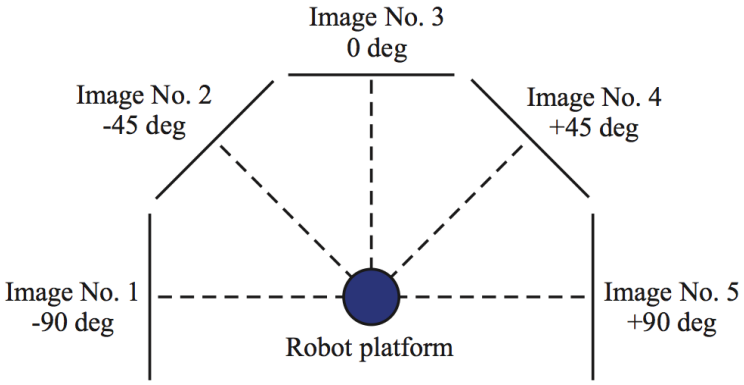


Figura 1.7: Immagini di scansione acquisite dal robot, per un totale di un campo di vista di $230°$ [16].

Per ciascuna delle immagini acquisite, si genera una mappa di confidenza, ordinando poi le regioni di interesse secondo la confidenza media (AC). Per ogni regione, si calcola la distanza dall'origine del sistema di riferimento d, come media delle distanze dei pixel appartenenti a quella regione, e l'orientamento θ, utilizzando la lunghezza focale della telecamera. A questo punto, si calcola la posizione della regione in coordinate cartesiane secondo la formula:

$$(x,y) = (d \cdot sin(\theta), d \cdot cos(\theta)) \tag{1.2}$$

Una volta ottenute le coordinate di ciascuna regione, si procede all'ispezione della stessa in modo piuttosto semplice, impostando al robot una traiettoria che lo porterà ad acquisire 3 immagini differenti della stessa regione, con un angolo di $60°$ ciascuna rispetto alla posizione di partenza. Risulta infatti inutile e difficoltoso acquisire una visione a $360°$ dell'oggetto, mentre la quantità di immagini scel-

ta sembra essere un buon compromesso. Per ciascuna delle immagini catturate, si procede poi alla fase di identificazione degli oggetti già descritta in precedenza, calcolando i punteggi AC e MR che consentiranno di stabilire se c'è o meno corrispondenza. In fig. 1.8 è possibile vedere un esempio di elaborazione.

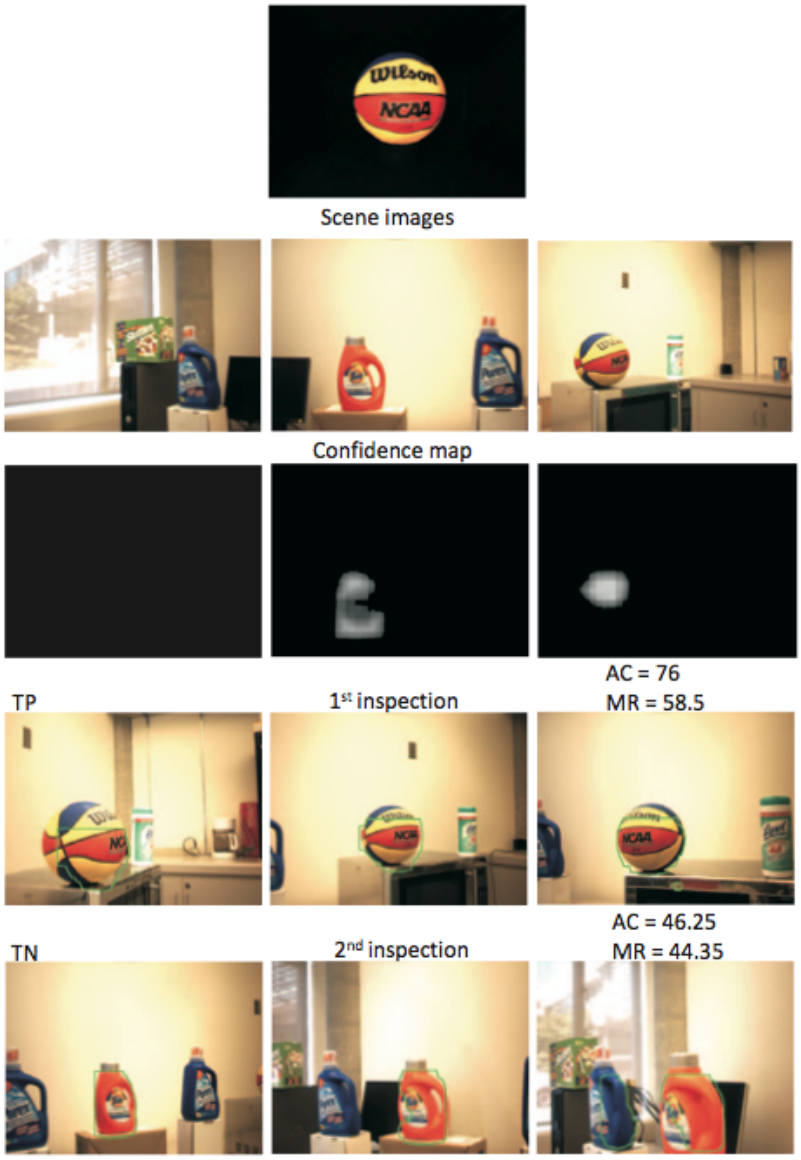


Figura 1.8: Esempio di elaborazione [16]: sulla prima riga, l'oggetto da cercare. Sulla seconda, 3 delle 5 immagini acquisite per la scansione dell'ambiente. Sulla terza riga, le mappe di confidenza di ciascuna scena. Sulla quarta e la quinta riga, il risultato dell'elaborazione.

Capitolo 2

Progettazione del sistema e strumenti utilizzati

Il sistema realizzato nell'ambito di questo lavoro di tesi consente ad un robot mobile di navigare e trovare oggetti nell'ambiente circostante utilizzando un sistema di percezione tridimensionale.

In fase di progettazione, il primo passo è stata la scelta del tipo di sensore, considerati i vantaggi e gli svantaggi di ciascuno già presentati nella capitolo 1. La scelta è ricaduta sulla combinazione di un sistema stereo di telecamere a colori con un laser scanner planare, che consenta di colmare, almeno in parte, quelle zone cieche causate dall'utilizzo dell'immagine di disparità nell'algoritmo di visione stereo. Come sistema stereo di telecamere è stata utilizzata la testa stereo realizzata nell'ambito di un precedente lavoro di tesi [19], composta da due webcam ad alta definizione Logitech C270, poste in un alloggiamento in alluminio ad una distanza l'una dall'altra di 12 *cm*. La scelta del laser scanner planare è invece ricaduta sul Sick LMS100, che può vantare specifiche tecniche di alto livello, tra le quali l'alta precisione.

L'utilizzo di una testa stereo di telecamere pone in essere alcuni problemi che è stato necessario risolvere. Dapprima la necessità della calibrazione dei parametri intrinseci ed estrinseci, al fine di consentire un buon funzionamento della stereovisione, che deve avvenire in modo rapido e semplice. Inoltre, un problema noto della visione stereo riguarda la bassa densità della point cloud generata,

risolto grazie all'integrazione dei dati sensoriali del laser scanner e all'accumulo delle ultime *N* point cloud acquisite.

La fusione sensoriale di dati provenienti da sensori diversi richiede necessariamente l'espressione degli stessi dati rispetto ad uno stesso sistema di riferimento e la loro memorizzazione in una stessa struttura dati, oltre alla decisione di un'opportuna politica di priorità che consenta, in casi di duplicazione dell'informazione su uno stesso punto, di decidere a quale sensore assegnare la precedenza.

La base mobile utilizzata è il robot Pioneer 3DX, già a disposizione del laboratorio di robotica e utilizzato ampiamente per scopi didattici e di ricerca, sul quale sono stati installati i due sensori presentati in precedenza.

A livello di strumenti software, i vari comportamenti (*behaviour*) realizzati sono stati integrati tramite il framework ROS (Robot Operating System), che consente di semplificare il processo di integrazione. La scelta della struttura dati da utilizzare per la memorizzazione della point cloud è ricaduta sulla classe PointCloud della Point Cloud Library [2], fornita di numerosi metodi predefiniti per l'elaborazione della point cloud e l'ottimizzazione della sua struttura, che si integra molto bene con il framework ROS utilizzato per la gestione dei behaviour. Infatti, sebbene anche ROS preveda una struttura dati di tipo point cloud, la classe mette a disposizione un semplice metodo per la conversione dal tipo di dato ROS al tipo di dato offerto dalla libreria PCL.

Anche l'identificazione degli oggetti è stata realizzata utilizzando strumenti e metodi offerti dalla classe PCL, come l'Euclidean Cluster Extraction, o sviluppati ad hoc con il suo ausilio. Questa costanza nell'utilizzo degli strumenti per le varie funzionalità offerte dal sistema ha consentito di semplificare il lavoro di integrazione.

Nelle sezioni seguenti sono presentate nel dettaglio le specifiche dell'hardware e le principali componenti del software utilizzato per lo sviluppo del sistema, concentrandosi particolarmente sulle caratteristiche in base alle quali si è optato per la scelta di essi.

2.1 Robot Pioneer 3DX

Il MobileRobots Pioneer 3DX (fig. 2.2(a)) è un robot mobile piccolo e leggero, dotato di due ruote comandate da altrettanti motori e da una ruota posteriore non attuata. L'equipaggiamento di base prevede, oltre a encoder per le ruote e pacco batterie, un insieme di sonar frontali e posteriori, che però sono stati rimossi nel robot a disposizione del laboratorio di robotica e sostituiti con un laser scanner frontale. In virtù della sua struttura e della configurazione, il Pioneer 3DX è facilmente approssimabile con il modello uniciclo della dinamica, mostrato in fig. 2.1.

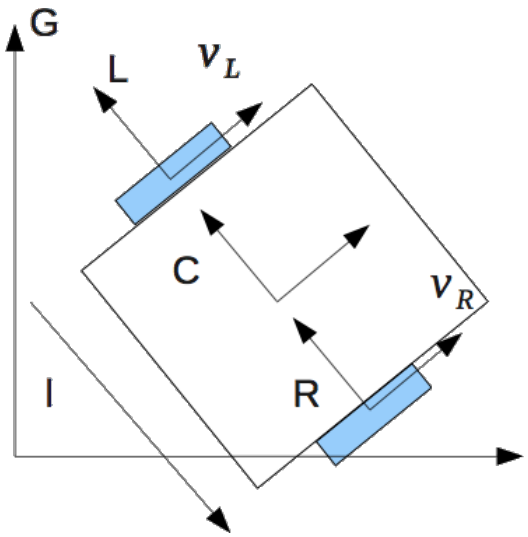
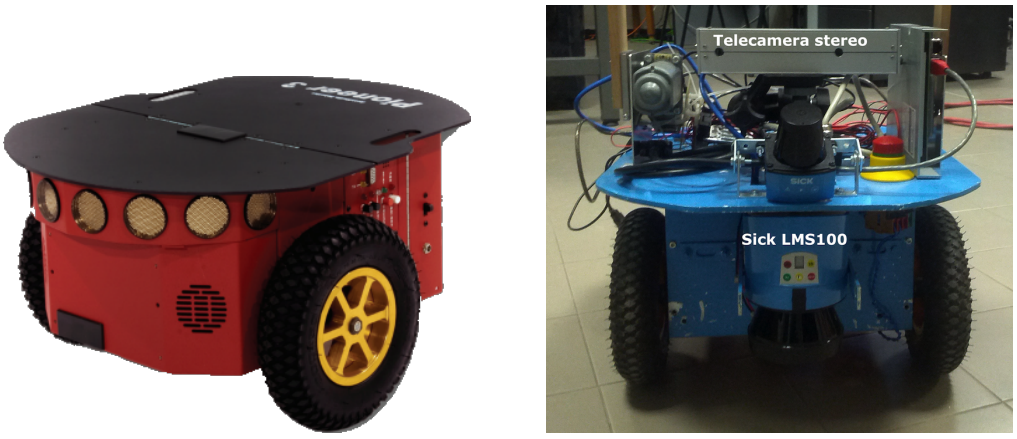


Figura 2.1: Modello uniciclo della dinamica di un robot mobile.

Per movimentare il robot è sufficiente inviare i comandi di velocità (lineare e angolare) tramite la porta seriale integrata. La velocità lineare massima sostenibile dal robot è di circa 0.7 m/s.

Il robot utilizzato in laboratorio, mostrato in fig. 2.2(b), è stato equipaggiato di un laser scanner Sick LMS100 (installato frontalmente), un laser scanner Sick TiM300 (installato frontalmente), non utilizzato in questo progetto, un sistema stereo di telecamere e una pinza attuata, non utilizzata in questo progetto.



(a) Versione originale (b) Versione utilizzata

Figura 2.2: Il robot mobile Pioneer 3DX.

2.2 Laser scanner Sick LMS100

Il Sick LMS100 (fig. 2.3) è un laser scanner 2D che viene utilizzato per misurare, in modo piuttosto preciso, la distanza degli oggetti presenti sul piano di scansione del laser. Il principio di funzionamento è piuttosto semplice: per ogni angolo di scansione viene inviato un raggio laser che colpisce il bersaglio (oggetto più vicino) e viene riflesso. Misurando il *tempo di volo*, ovvero il tempo che impiega il raggio riflesso a tornare indietro, è possibile calcolare la distanza dell'oggetto. L'angolo di scansione è compreso nell'intervallo $[-\frac{3\pi}{4}, \frac{3\pi}{4}]$ con una precisione di $0.5°$.

Figura 2.3: Il laser scanner planare Sick LMS100.

Il laser scanner è stato montato frontalmente al robot. Dato che la presenza delle ruote del Pioneer 3DX occlude parte del piano di scansione, il range effettivamente utilizzato è stato ridotto all'intervallo $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

Il driver del laser restituisce le distanze sotto forma di array di numeri in virgola mobile, in cui la posizione dell'elemento può essere facilmente convertita nell'angolo di scansione corrispondente.

2.3 Sistema stereo di telecamere

Il sistema stereo di telecamere, chiamato anche *testa stereo* in virtù della sua funzione visiva per il robot, è stato realizzato in un precedente lavoro di tesi [19] utilizzando due webcam Logitech C270 HD (fig. 2.4(c)), caratterizzate da una risoluzione massima di 1280 x 720 pixel, posizionate in uno chassis di alluminio (fig. 2.4(a)), dopo aver rimosso l'alloggiamento originale.

(a) Involucro esterno (b) Assemblaggio (c) Componentistica

Figura 2.4: La testa stereo utilizzata.

La testa stereo viene utilizzata dal robot per acquisire due immagini (*left* e *right*) dalle quali, tramite un apposito modulo software, è ricavata l'immagine di disparità, un'immagine in cui il colore di ogni singolo pixel contiene l'informazione di distanza dell'oggetto a cui quel pixel appartiene. In virtù della complessità del calcolo dell'immagine di disparità e al fine di ridurre il carico computazionale per le successive elaborazioni, la risoluzione delle immagini acquisite è stata ridotta a 640 x 480 pixel.

2.4 Staffa per l'autocalibrazione

Il calcolo dell'immagine di disparità presuppone che le due telecamere siano calibrate in modo corretto, ovvero che i parametri intrinseci ed estrinseci siano

impostati ai valori giusti.

I parametri intrinseci sono i parametri necessari a collegare le coordinate di un pixel dell'immagine con le coordinate corrispondenti nel sistema di riferimento della telecamera. Questi parametri dipendono esclusivamente dalle caratteristiche hardware della telecamera e, pertanto, non variano o variano poco nel tempo. Possono essere calcolati inquadrando un pattern noto, come una scacchiera, di dimensioni note e avviando il nodo ROS *camera_calibration*, descritto in seguito (capitolo 3.2).

I parametri estrinseci sono i parametri che definiscono posizione ed orientazione del sistema di riferimento della telecamera rispetto al riferimento mondo. Come si può intuire, ogni benché minima modifica, anche accidentale, della posizione o dell'orientazione della telecamera richiede una nuova calibrazione. La calibrazione dei parametri estrinseci avviene inquadrando un marker dal pattern noto, in una posizione e con un orientamento noti, e utilizzando il software ARToolkit [20] per ricavare il valore dei parametri.

Per semplificare il processo di calibrazione è stata costruita un'apposita staffa in alluminio, mostrata in fig. 2.5, sulla quale è stato fissato il marker, incollato ad un quadrato di plastica rigida, ad una distanza di 80.3 *cm* dal punto di fissaggio.

Ogniqualvolta si deve ricalibrare la telecamera, sarà sufficiente montare la staffa sul robot ed avviare il nodo di calibrazione (capitolo 3.2) senza dover modificare nessun parametro di configurazione, in quanto la posizione del marker rispetto al robot sarà sempre la stessa.

2.5 Framework ROS

ROS (Robot Operating System) è un framework open source sviluppato appositamente per facilitare la scrittura di applicazioni robotiche, mettendo a disposizione del programmatore svariati strumenti e librerie già pronti e testati. ROS fornisce astrazione hardware, driver per le periferiche più comuni (tra cui il laser LMS100), visualizzatori (per vedere il risultato delle elaborazioni) e molto altro. In particolare, il componente fondamentale di ROS è un middleware che gestisce la comunicazione tra i nodi, che prevede un sistema di *discovery* dei nodi, di indi-

Figura 2.5: Staffa per l'autocalibrazione.

rizzamento e redistribuzione dei messaggi fra i nodi, di memorizzazione di errori e avvisi (log) e alcune primitive di sincronizzazione approssimata.

La struttura di ROS si basa sui *nodi*, ovvero processi che eseguono l'elaborazione, i quali possono comunicare tra loro utilizzando, ad esempio, lo scambio di messaggi su specifici *topic*. Ogni nodo ha un compito ben preciso nel governo del robot, pertanto il sistema di controllo di un robot sarà composto da tanti nodi, in un'ottica di *modularità* della programmazione.

Uno degli strumenti integrati maggiormente utilizzato è il visualizzatore 3D *Rviz*, che consente di visualizzare tutti i dati scambiati tra i vari nodi di ROS, permettendo all'utente di scegliere quali dati visualizzare. Le immagini dei risultati del sistema sviluppato mostrate nei capitoli successivi sono state catturate grazie a questo potente strumento, configurato correttamente e avviato automaticamente dal nodo di acquisizione dei dati sensoriali.

In virtù della natura open source di ROS, attorno a questo framework si è creata una vera e propria comunità, composta da università, centri di ricerca e

singoli programmatori, che spesso mette a disposizione il codice delle proprie applicazioni, che possono poi essere integrate in nuovi lavori evitando di doverle sviluppare nuovamente. Questo aspetto è risultato molto importante nel lavoro di questa tesi in quanto ha consentito di concentrare l'impegno sulla *percezione* e tralasciare l'aspetto di pianificazione del moto verso il goal da parte del robot, per il quale è stato utilizzato un nodo già sviluppato dalla comunità.

2.6 Libreria PCL

La Point Cloud Library (PCL) è una libreria open source per l'elaborazione di immagini 2D/3D e di point cloud ([2] [3]). Una point cloud è una nuvola di punti in tre dimensioni, che possono contenere o meno l'informazione del colore in RGB, molto utile per visualizzare e memorizzare efficientemente i dati provenienti da un sensore come una telecamera stereo. Lo sviluppo di questa libreria, arrivata alla versione 1.6.0, è curato dal laboratorio di ricerca Willow Garage, specializzato nella robotica e nell'elaborazione delle immagini, e sponsorizzato da importanti università e grandi aziende internazionali, tra le quali Intel e Toyota.

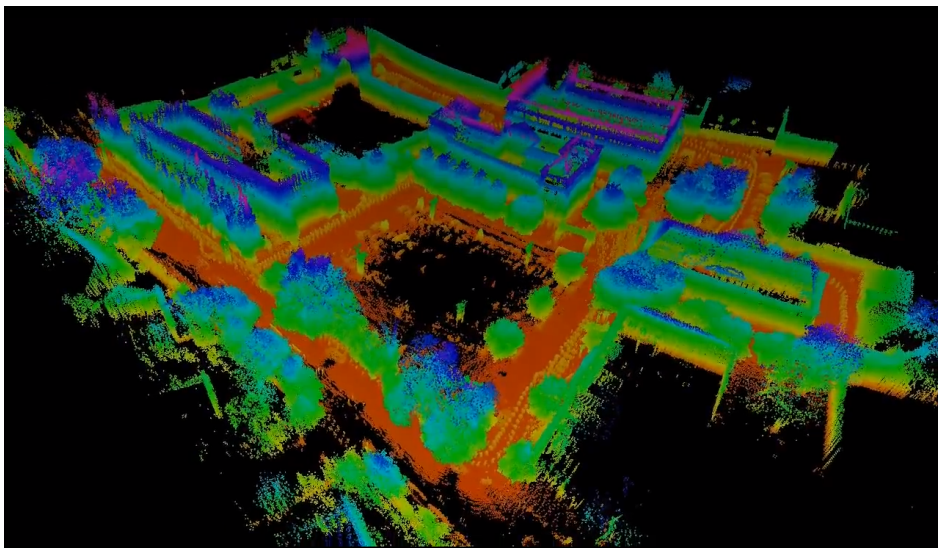


Figura 2.6: Rappresentazione di un ambiente mediante point cloud.

Questa enorme libreria comprende, oltre ai metodi standard di elaborazione di un'immagine 2D, specifiche implementazioni di algoritmi anche evoluti di com-

puter vision, molti dei quali sono stati utilizzati in questo lavoro di tesi. Infatti, sia i dati provenienti dal laser che quelli provenienti dalla testa stereo vengono inseriti in un'unica point cloud che servirà poi per la navigazione del robot mobile in sicurezza.

La sua integrazione con il framework ROS risulta facilitata da alcune primitive e API già implementate, motivo per cui è stato scelto questo strumento software nell'ambito dello sviluppo di questa tesi.

Capitolo 3

Realizzazione del sistema

In questo capitolo viene presentato il sistema di navigazione, individuazione e osservazione di oggetti che, utilizzando i dati della percezione 3D, permette al robot di osservare un oggetto da più angolazioni. Dopo aver visionato il sistema ad alto livello, si analizzeranno più nel dettaglio le implementazioni dei singoli nodi, ciascuno con una specifica funzione.

3.1 Architettura del sistema

Ad alto livello, il sistema realizzato è strutturato come mostrato in fig. 3.1. Ciascun passo presente nel diagramma corrisponde ad una funzione svolta, generalmente, da un'applicazione a sé stante (nodo), che viene poi integrata nel sistema complessivo grazie al framework ROS.

Dato che alcune operazioni costituiscono una fase di *preprocessing*, ovvero di preparazione dei dati sensoriali, o di *postprocessing*, ovvero di applicazione dei comandi di velocità al robot calcolati sulla base di elaborazioni sviluppate in questo progetto di tesi, si è preferito raggruppare alcune funzioni in macrofunzioni, indicate a lato del diagramma funzionale, che verranno dettagliate nelle sezioni successive.

Il programma principale che viene eseguito dal robot per la sua movimentazione esegue tutte queste funzioni ad ogni ciclo. Pertanto, misurati i tempi di elaborazione del sistema complessivo, si è impostata un'unica frequenza di la-

voro per ogni singola funzione al fine di ottimizzare il carico computazionale, evitando così che nodi meno complessi elaborino e producano dati che poi non vengono utilizzati in quanto i nodi più complessi stanno ancora elaborando i dati precedenti.

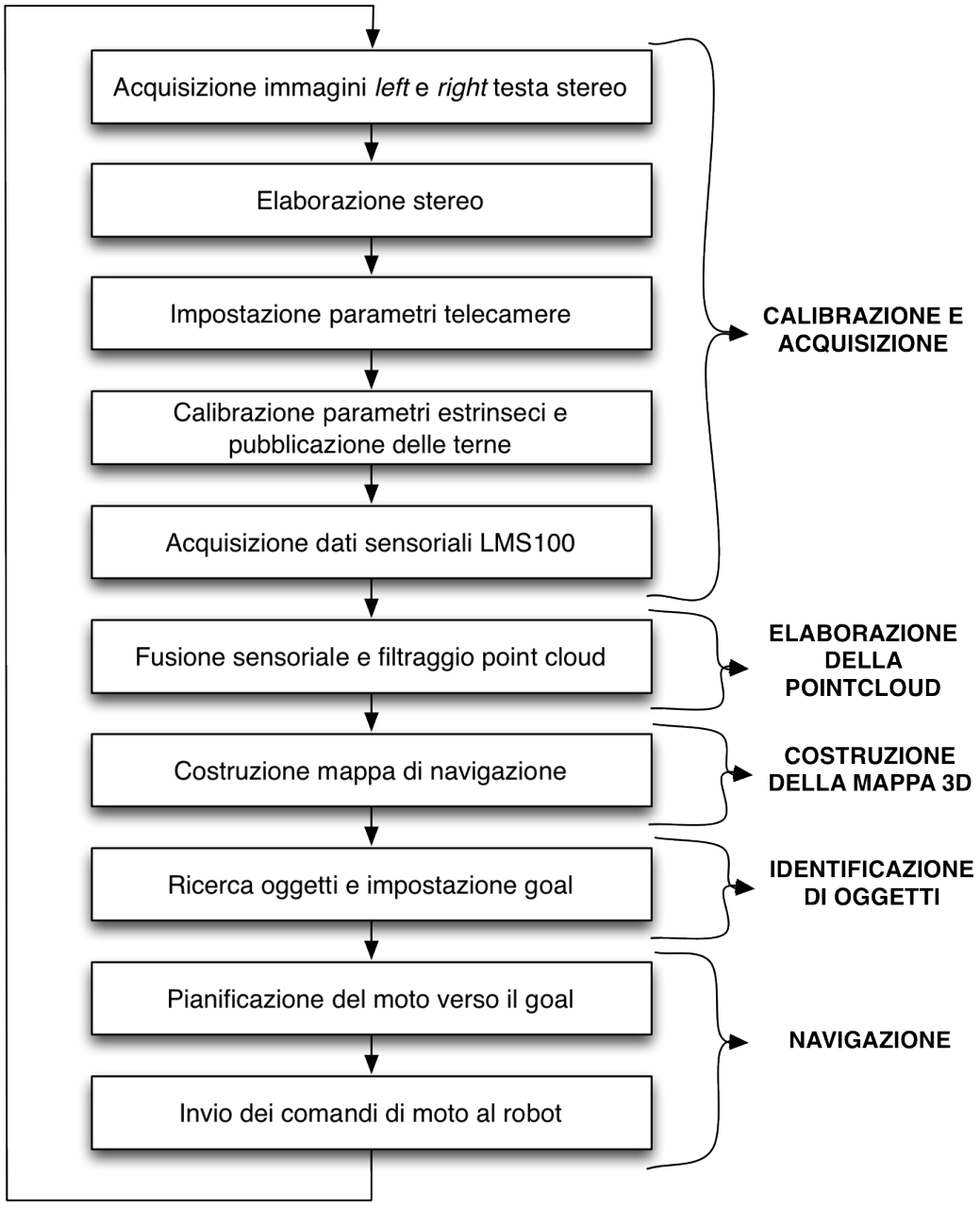


Figura 3.1: Pipeline di elaborazione del sistema realizzato.

Il diagramma di fig. 3.2 mostra graficamente i principali messaggi scambiati tra i vari nodi di ROS durante l'elaborazione. Gli ovali indicano i nomi dei nodi di ROS, ovvero dei processi attivi durante l'esecuzione del sistema complessivo; i nomi accanto alle frecce tra i nodi indicano il topic pubblicato da un nodo (output) che viene letto dal nodo successivo (input).

Il nodo */baseline12/bsl12_camera_settings* risulta isolato in quanto agisce direttamente sui parametri di configurazione della telecamera, senza utilizzare messaggi ROS di altri nodi e senza pubblicare nessun risultato. Infatti, il nodo di acquisizione */baseline12/uvc_camera_bsl12* acquisirà l'immagine con i parametri della telecamera già impostati internamente al dispositivo.

Come si può intuire dal diagramma, il sistema genera un discreto traffico di informazioni tra i vari nodi e, pertanto, l'utilizzo di un framework come ROS consente di garantire l'ottimizzazione dello stesso. In particolare, i dati vengono effettivamente pubblicati da parte di un *publisher* (scrittore) solamente se c'è almeno un *subscriber* (lettore) in ascolto su quel topic. Questo consente di migliorare l'efficienza e di ridurre il traffico generale, specie per quanto riguarda i nodi già implementati e integrati in questo sistema che, spesso, offrono più funzioni di quelle effettivamente utilizzate.

3.2 Calibrazione ed acquisizione

Lo scopo dei componenti di calibrazione ed acquisizione è determinare i parametri di identificazione dei sensori e preparare i dati sensoriali provenienti dalle telecamere e dal laser scanner per la successiva fusione ed elaborazione.

Il primo passo è quello di acquisire le due immagini a colori dalle telecamere posizionate nella testa stereo. Per farlo è stato utilizzato il nodo ROS *uvc_camera* specifico per le telecamere stereo, che mette a disposizione un pacchetto di driver standard per le telecamere USB più diffuse.

Tale nodo necessita di alcuni parametri di input, riassunti qui di seguito:

- *left/device*: percorso del dispositivo di acquisizione collocato a sinistra nella testa stereo;

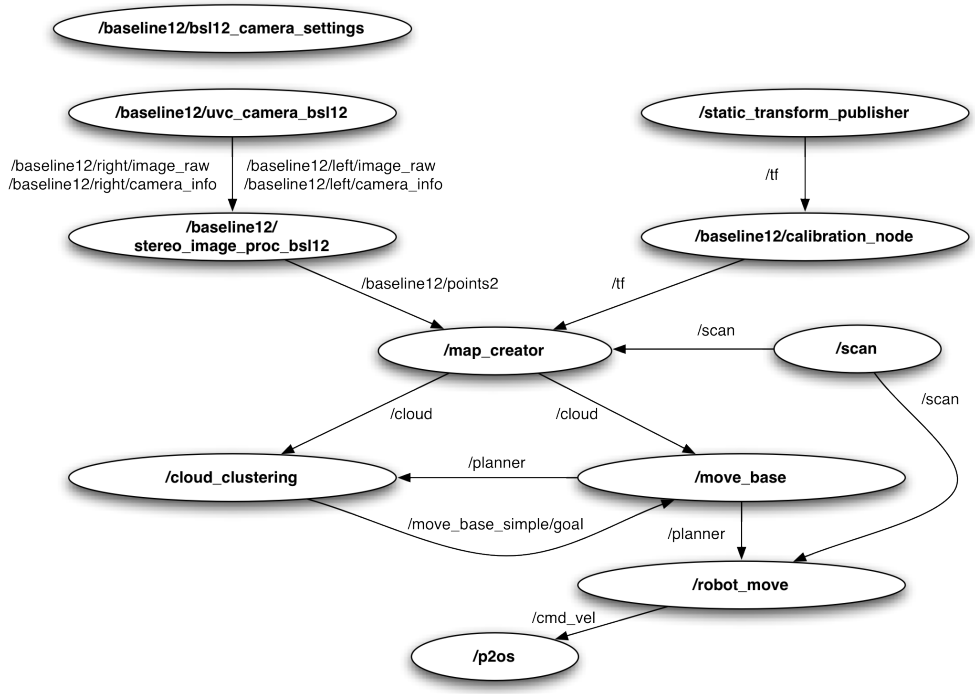


Figura 3.2: Messaggi scambiati tra i vari nodi di ROS.

- *right/device*: percorso del dispositivo di acquisizione collocato a destra nella testa stereo;
- *left/camera_info_url*: percorso del file di calibrazione dei parametri intrinseci della telecamera sinistra;
- *right/camera_info_url*: percorso del file di calibrazione dei parametri intrinseci della telecamera destra;
- *fps*: velocità di acquisizione richiesta alle telecamere, misurata in frame al secondo;
- *skip_frames*: numero di frame da saltare dopo ogni frame acquisito;
- *width*: larghezza massima dell'immagine acquisita dalle due telecamere;
- *height*: altezza massima dell'immagine acquisita dalle due telecamere;

- *frame_id*: nome della terna che rappresenta la posizione e l'orientazione delle telecamere.

Questo nodo pubblica attraverso ROS due messaggi su altrettanti *topic* per ogni telecamera, il primo, *image_raw*, contenente l'immagine acquisita in scala di grigi, il secondo, *camera_info*, contenente le specifiche del dispositivo di acquisizione. Tuttavia, queste immagini ottenute da telecamere non correttamente calibrate risultano essere inutilizzabili per la visione stereo.

Come detto in precedenza, i parametri di calibrazione da impostare in una telecamera sono di due tipi: intrinseci ed estrinseci.

Dato che i parametri intrinseci dipendono dal dispositivo telecamera e non dalla sua posizione o orientazione, è sufficiente eseguirne la calibrazione una volta sola. Per questa calibrazione è stato utilizzato il nodo ROS predefinito *camera_calibration*, che consente di calibrare due telecamere utilizzando una scacchiera di dimensioni assegnate. Tale nodo necessita delle opzioni di lancio riassunte di seguito, specificabili precedendole da $--$:

- *size*: numero di croci in orizzontale e in verticale della scacchiera, separate da una *x* (ad esempio *8x6*);

- *square*: misura, in metri, del lato di ogni quadrato della scacchiera.

La scacchiera utilizzata per la calibrazione è mostrata in fig. 3.3.

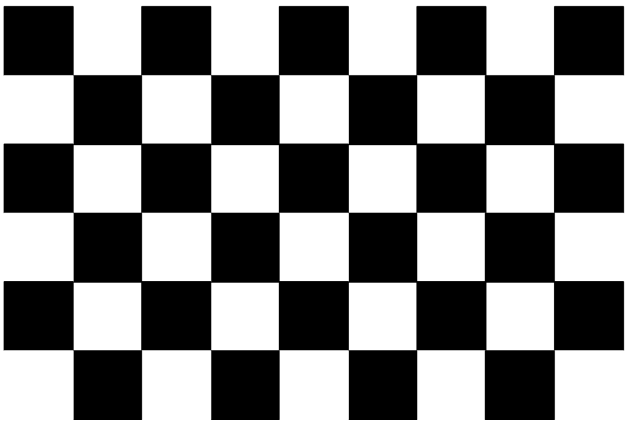


Figura 3.3: Scacchiera 8x6 utilizzata per la calibrazione delle telecamere.

Sono inoltre necessari alcuni parametri per specificare il dispositivo da calibrare:

- *left*: immagine della telecamera di sinistra pubblicata dal nodo *uvc_camera*;
- *right*: immagine della telecamera di destra pubblicata dal nodo *uvc_camera*;
- *left_camera*: nome da assegnare alla telecamera di sinistra;
- *right_camera*: nome da assegnare alla telecamera di destra.

Una volta avviato il nodo, è sufficiente posizionare la scacchiera frontalmente al robot, in modo che sia completamente visibile dalle telecamere, e muoverla in tutte le direzioni al fine di impostare correttamente i parametri di calibrazione adatti ad ogni posizione degli oggetti. L'interfaccia utente del nodo è mostrata in fig. 3.4. Dopo qualche minuto, il nodo produce l'output che consiste in due file yaml (dati serializzati) contenenti i parametri intrinseci ottenuti dal processo di calibrazione, che verranno caricati dal nodo di acquisizione ad ogni esecuzione.

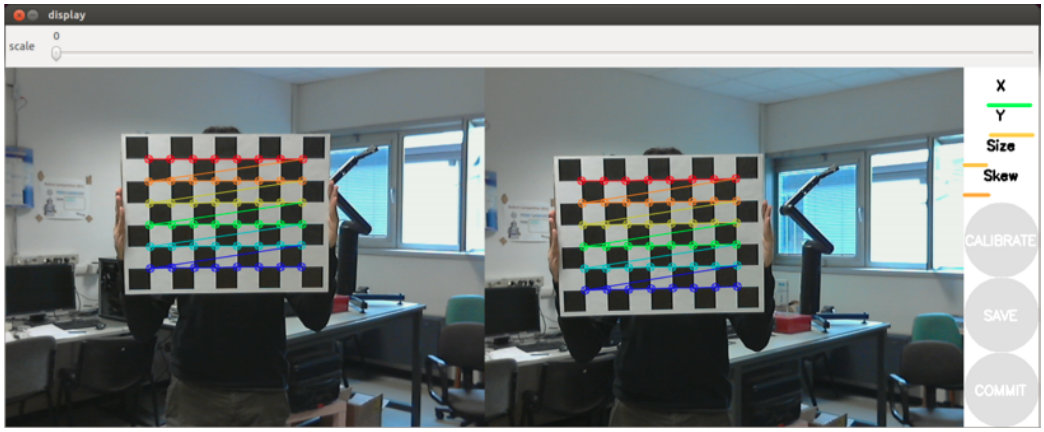


Figura 3.4: Interfaccia utente del nodo di calibrazione.

Una volta avviato il nodo di acquisizione, occorre impostare alcuni parametri correttivi relativi alle singole immagini utilizzando il nodo *camera_settings*, sviluppato in una tesi precedente [19] all'interno del laboratorio di robotica, che necessità dei seguenti parametri di configurazione:

- *brightness*: luminosità dell'immagine;

- *contrast*: contrasto dell'immagine;
- *saturation*: saturazione dei colori dell'immagine;
- *frequency*: frequenza di lavoro dell'alimentazione, che può essere o 50Hz o 60Hz;
- *exposure*: correttivo dell'esposizione.

Occorre inoltre specificare, sempre tra i parametri, il dispositivo della telecamera destra *right/device* e quello della telecamera sinistra *left/device* su cui andranno applicate le impostazioni. Una volta avviato, il nodo ROS controlla continuamente se i parametri letti dalla telecamera sono diversi da quelli impostati da utente e, nel caso, invia il comando di impostazione apposito tramite una libreria di comunicazione USB. Questo nodo rimane in esecuzione in background per tutto il periodo di funzionamento del sistema, al fine di garantire una costante applicazione dei parametri di configurazione e, quindi, una corretta percezione. Tuttavia, per ridurre il carico computazionale, potrebbe essere terminato subito dopo aver applicato le impostazioni, dato che la maggior parte delle volte gli ultimi parametri di configurazione impostati rimangono nella memoria della telecamera.

A questo punto, le due telecamere risultano configurate correttamente e sono pronte per essere utilizzate per produrre l'immagine di disparità, utile per la visione stereoscopica. Il calcolo della disparità e il successivo utilizzo per la visione stereoscopica viene realizzato dal nodo ROS *stereo_image_proc* che, dopo aver preso in ingresso le immagini RAW e le informazioni di configurazione della telecamera pubblicate dal nodo *uvc_camera*, esegue varie elaborazioni. In particolare:

- elimina le distorsioni delle due telecamere tramite rettificazione, un processo che consente di portare tutti i punti omologhi delle immagini sulla stessa linea orizzontale (fig. 3.5), pubblicando, per ogni telecamera, l'immagine mono *image_rect*;
- colora l'immagine acquisita utilizzando un semplice filtro di Bayer [21], pubblicando l'immagine *image_rect_color*;

- calcola l'immagine di disparità, utilizzando BMA (Block Matching Algorithm), pubblicando l'immagine *disparity*;
- crea una point cloud, pubblicata nel messaggio *points2* e risultante dal processo di stereoscopia, che sarà poi possibile elaborare con la libreria PCL [2].

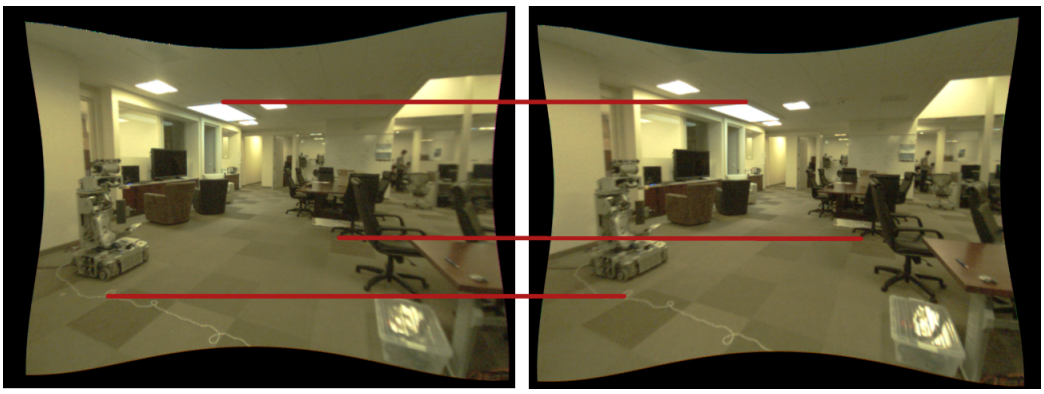


Figura 3.5: Immagini rettificate delle due telecamere.

La nuvola di punti generata viene espressa rispetto al sistema di riferimento della telecamera sinistra, impostato come mostrato in fig. 3.6. L'immagine effettivamente acquisita risulta essere quella della telecamera di sinistra, mentre la telecamera destra ha come compito principale l'aggiunta dell'informazione di profondità.

L'ultima fase del processo di calibrazione riguarda l'impostazione dei parametri estrinseci della telecamera, ovvero quelli relativi all'orientamento e alla posizione della telecamera rispetto al mondo. La taratura di questi parametri viene effettuata tramite il nodo ROS *ar_pose*, un wrapper della libreria software ARToolkit [20] che utilizza un apposito marker quadrato, come quello mostrato in fig. 3.7 chiamato *kanji*, per calcolare la matrice di trasformazione tra il sistema di riferimento della telecamera e quello centrato nel punto di mezzo del marker.

Questo nodo necessita di alcuni parametri di configurazione per funzionare, tra i quali:

- *marker_pattern*: percorso del file contenente la struttura del marker utilizzato;

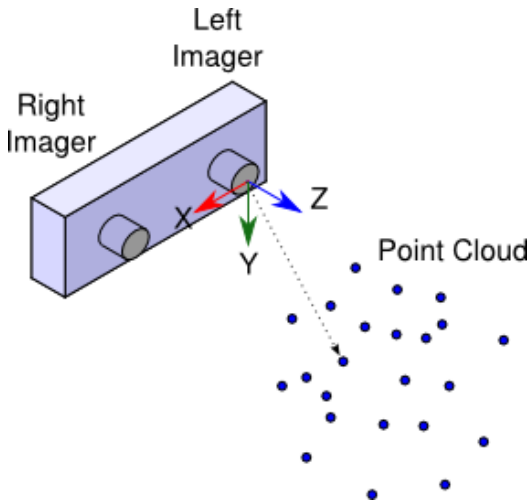


Figura 3.6: Sistema di riferimento della point cloud.

Figura 3.7: Marker kanji utilizzato per la calibrazione.

- *marker_width*: dimensione, in millimetri, del lato del marker;
- *marker_center_x*: ascissa del centro del marker;
- *marker_center_y*: ordinata del centro del marker;
- *marker_frame*: nome del sistema di riferimento del marker creato dal nodo.

Il nodo si aspetta di ricevere messaggi dai topic */usb_cam/image_raw* e */usb_cam/camera_info*, pertanto, se l'immagine utilizzata è pubblicata su un altro topic, occorre eseguire un'operazione di *remap*. Il risultato della calibrazione è una trasformazione pubblicata attraverso il pacchetto *tf*, che consente al robot di tenere traccia di tutti i sistemi di riferimento utilizzati nel tempo e delle trasformazioni per passare da un sistema ad un altro.

Sebbene i parametri di calibrazione estrinseci possano variare con più frequenza rispetto a quelli intrinseci, una nuova calibrazione ad ogni esecuzione del sistema di navigazione risulterebbe eccessiva e, inoltre, richiederebbe la presenza del marker sulla scena, riducendo notevolmente la libertà di movimento del robot, limitandola a zone in cui il marker sia visibile. Pertanto, è stato sviluppato un apposito nodo di calibrazione, denominato *calibration_node*, che utilizza le funzionalità offerte dal nodo *ar_pose*, basato sulla libreria ARToolkit [20], per determinare la posizione e l'orientamento nello spazio di particolari marker. Quando la staffa di calibrazione è smontata e, quindi, il marker non è presente nella scena, il *calibration_node* pubblica i dati di calibrazione ottenuti in precedenza e opportunamente salvati in un file di configurazione. Questo consente al robot di muoversi in ambienti non dotati di marker con le telecamere opportunamente calibrate.

Il nodo di calibrazione si occupa di pubblicare, attraverso *tf*, tutti i sistemi di riferimento (terne), con le relative trasformazioni, utilizzati dal sistema complessivo. In particolare:

- *baseline12* è la terna della telecamera stereo, centrata sulla telecamera di sinistra e creata dal nodo *stereo_image_proc*;
- *marker* è la terna del marker, con l'origine nel punto centrale del marker, creata dal nodo *ar_pose*;
- *base_link* è la terna del robot, con l'origine nel punto centrale dell'asse delle ruote, utilizzata in tutta la navigazione come sistema di riferimento principale;
- *laser* è la terna del laser, centrata sul laser LMS100.

Le terne *base_link* e *laser* sono create tramite una rotazione $(roll, pitch, yaw)$ e una traslazione (x, y, z) applicate ad hoc alla terna *marker* e specificate tramite file

di lancio del nodo. I parametri di traslazione sono stati misurati come traslazione tra il centro del marker e il centro dell'asse delle ruote del robot, nel caso della terna *base_link*, o il centro del laser LMS100, nel caso della terna *laser*. Il sistema di terne complessivo utilizzato per la percezione è mostrato in fig. 3.8.

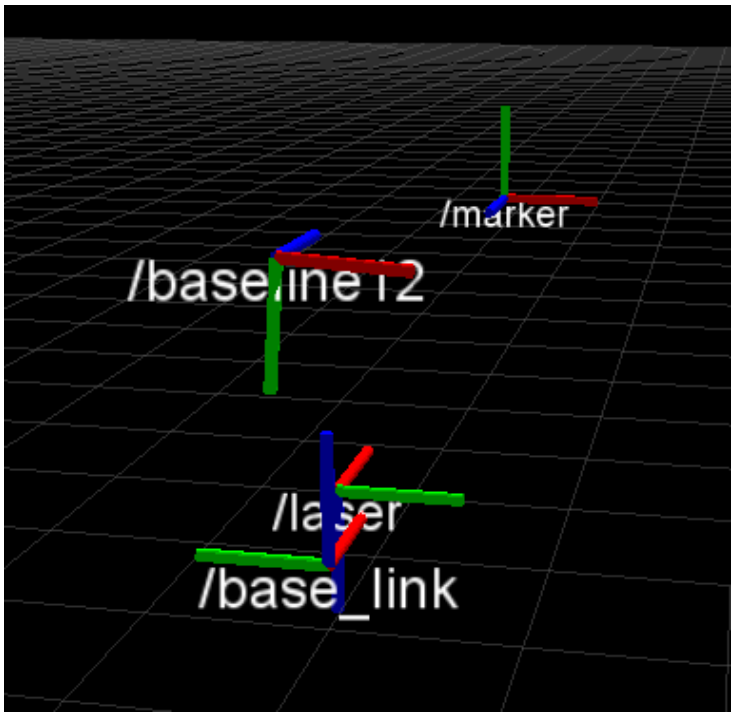


Figura 3.8: Disposizione delle terne utilizzate per la percezione.

Con questo nodo si conclude la prima macrofunzione svolta dal sistema sviluppato. Il passo successivo riguarda l'elaborazione della nuvola di punti generata utilizzando la Point Cloud Library, aggiungendovi i dati provenienti dal laser scanner.

3.3 Elaborazione della point cloud

Questa fase di elaborazione ha lo scopo di creare la point cloud che verrà utilizzata per la costruzione della mappa di navigazione, basandosi sui dati provenienti dalle telecamere e dal laser scanner opportunamente acquisiti al passo precedente. Il

nodo realizzato, denominato *camera_mapping*, svolge numerose operazioni sulla point cloud, riassunte in fig. 3.9.

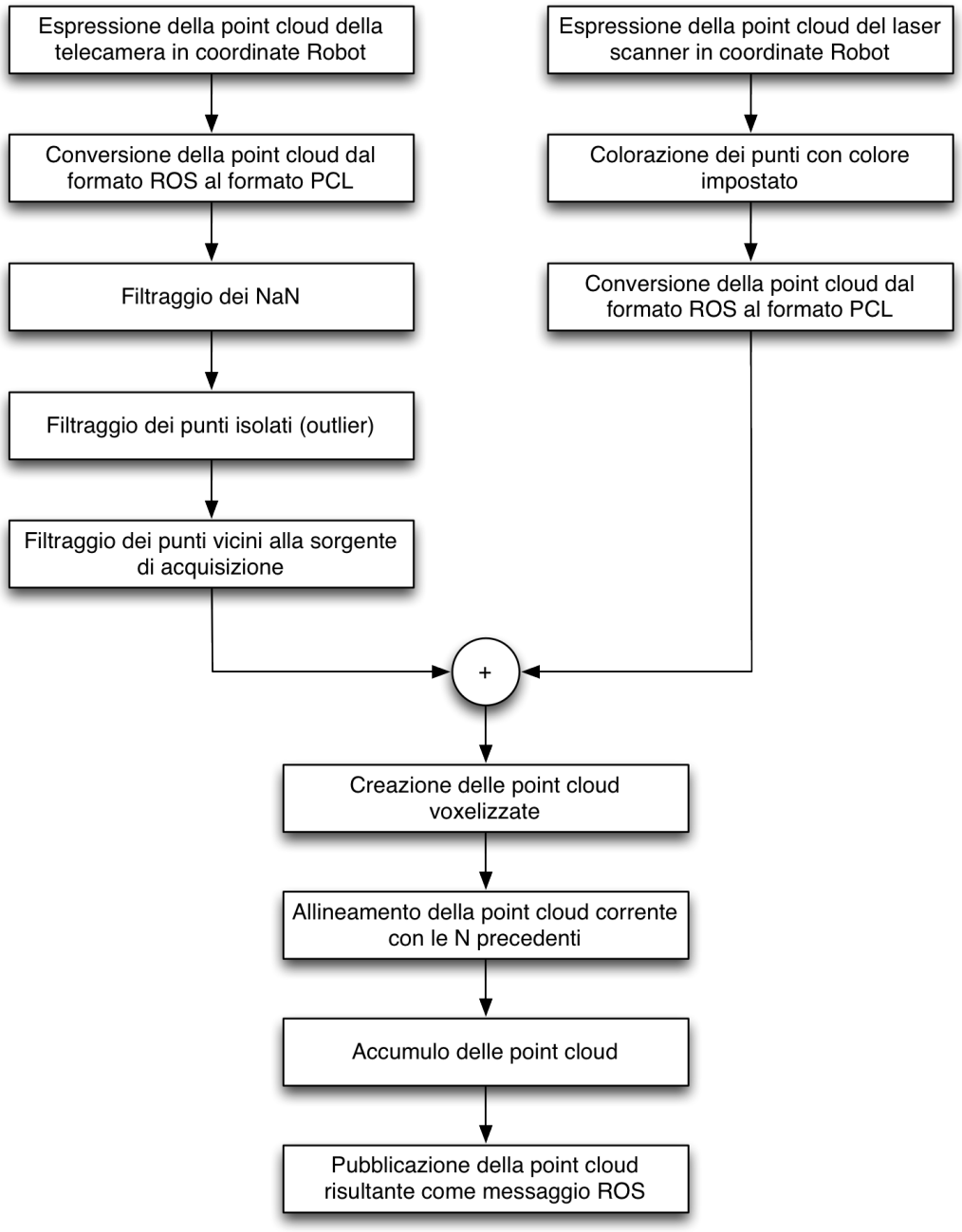


Figura 3.9: Struttura di elaborazione del nodo *camera_mapping*.

Come è possibile notare, il flusso di elaborazione procede inizialmente su due

binari diversi, a causa della presenza di due dati sensoriali, uno proveniente dalla testa stereo e uno proveniente dal laser scanner. Tali dati, dopo alcune operazioni di preparazione, vengono uniti con il processo di *fusione sensoriale*, al fine di generare, dopo alcune altre elaborazioni, la point cloud di percezione 3D che sarà poi utilizzata nelle successive operazioni di navigazione e identificazione di oggetti.

Il nodo si avvia per mezzo di un file di lancio dal quale è possibile specificare il valore dei parametri di configurazione per il suo corretto funzionamento. In particolare:

- *numPCLToStore*: numero di point cloud da accumulare;
- *voxelX*, *voxelY*, *voxelZ*: misura, nelle tre dimensioni, del singolo voxel in cui viene suddivisa la point cloud;
- *icpMaximumIterations*: numero massimo di iterazioni nell'allineamento delle point cloud;
- *icpMaxCorrespondenceDistance*: distanza massima dei punti considerati vicini nell'allineamento delle point cloud;
- *ktreeDistance*: soglia di distanza massima dai vicini per i punti della point cloud nell'operazione di accumulo;
- *ktreeThr*: percentuale minima di punti corrispondenti per l'accumulo della point cloud;
- *meanK*: numero di vicini da analizzare nella rimozione degli outlier;
- *stddev*: distanza massima dei punti dalla distanza media per non essere considerati outlier, espressa come moltiplicatore della deviazione standard;
- *robotFrame*: nome del sistema di riferimento del robot;
- *laser_topic*: nome del topic sul quale il laser pubblica i dati sensoriali;
- *pointsNode*: nome del topic sul quale il nodo di acquisizione pubblica la point cloud;

- *output_cloud_topic*: nome del topic sul quale pubblicare la point cloud filtrata;
- *PCDPath*: percorso nel quale salvare le point cloud di output;
- *PCDCreation*: attiva o disattiva il salvataggio su file delle point cloud;
- *laserEnabled*: attiva o disattiva l'utilizzo dei dati sensoriali del laser;
- *ROSrate*: frequenza di lavoro dell'applicazione (in Hz);
- *outlierFilter*: attiva o disattiva il filtraggio degli outlier;
- *laserR*, *laserG*, *laserB*: colore RGB da assegnare ai dati sensoriali del laser;
- *minX*, *maxX*: intervallo di valori della coordinata x del punto entro i quali rimuoverlo dalla point cloud;
- *minY*, *maxY*: intervallo di valori della coordinata y del punto entro i quali rimuoverlo dalla point cloud;
- *minZ*, *maxZ*: intervallo di valori della coordinata z del punto entro i quali rimuoverlo dalla point cloud.

Le diverse operazioni svolte dal nodo e riassunte in fig. 3.9 sono ripetute ad ogni ciclo di esecuzione. Una visione in pseudocodice delle operazioni svolte è mostrata nell'algoritmo 1.

La prima operazione effettuata dal nodo riguarda la conversione dei due flussi di dati in coordinate robot. Le matrici di trasformazione, come detto in precedenza, sono pubblicate dal pacchetto *tf* ad una frequenza decisa internamente e che non necessariamente è la stessa per tutte le trasformazioni. Per questo, si è reso necessario l'utilizzo di un *tf::MessageFilter*, un particolare tipo di dato predefinito di ROS che consente di ricevere e sincronizzare le informazioni sui sistemi di riferimento da un *tf::TransformListener*. In questo modo si evita che ROS cerchi di applicare, a causa del carico di lavoro del processore, una trasformazione di coordinate non ancora o non più disponibile.

Successivamente ha luogo la fase di preelaborazione riguardante la point cloud contenente i dati sensoriali provenienti dalle telecamere. La prima operazione

```
1  leggi i parametri di configurazione da file di lancio;
2  while ros::ok() do
3      leggi i dati sensoriali da ROS e memorizzali in una point cloud;
4      trasforma le point cloud nel sistema di riferimento robot;
5      converti le point cloud dal formato ROS al formato PCL;
6      filtra i valori NaN dalla point cloud della telecamera;
7      if rimozione outlier abilitata then
8          filtra gli outlier utilizzando StatisticalOutlierRemoval;
9      end
10     filtra i punti troppo vicini alla telecamera utilizzando PassThrough;
11     if laser abilitato & dimensioni della point cloud del laser ≠ 0 then
12         fusione sensoriale;
13     end
14     if è la prima point cloud ad essere acquisita then
15         inserisci la point cloud nel vettore;
16     else
17         if numero di point cloud da accumulare = 1 then
18             svuota il vettore di point cloud;
19             inserisci la point cloud nel vettore;
20         else
21             if il vettore di point cloud è pieno then
22                 elimina la point cloud più vecchia;
23             end
24             ricava la matrice di trasformazione per allineare la point cloud
               corrente all'ultima acquisita utilizzando ICP;
25             forall the point cloud do
26                 applica la trasformazione alla point cloud;
27             end
28             somma le point cloud contenute nel vettore;
29             converti la point cloud risultante in formato ROS;
30             pubblica la point cloud in un messaggio ROS;
31             if salvataggio su file abilitato then
32                 salva la point cloud in un file .pcd;
33             end
34         end
35     end
36 end
```

Algorithm 1: Pseudo codice delle operazioni svolte dal nodo *camera_mapping*.

consiste nel passaggio da coordinate camera a coordinate robot, al fine di poter esprimere tutti i dati sensoriali rispetto alla stessa terna di riferimento posizionata nel punto centrale del robot mobile. Infatti, un prerequisito fondamentale nel processo di fusione sensoriale è la compatibilità dei sistemi di riferimento dei dati che si vanno a fondere. Occorre quindi che sia la point cloud della telecamera, espressa nel sistema di riferimento *baseline12*, sia la point cloud del laser, espressa nel sistema di riferimento *laser*, vengano trasformate, tramite l'applicazione di un'apposita matrice di trasformazione ricavata dal nodo *tf*, nel sistema di riferimento *robot*.

Per effettuare questa trasformazione, nel caso della point cloud della telecamera si utilizza il metodo *pcl_ros::transformPointCloud*, in cui occorre specificare il sistema di riferimento in cui trasformare la point cloud di ingresso e l'oggetto di classe *tf::TransformListener* dal quale recuperare la matrice di trasformazione. La point cloud risultante, in formato ROS *sensor_msgs::PointCloud2*, viene poi convertita nel formato *pcl::PointCloud<PointXYZRGB>*, ovvero in una nuvola di punti contenenti sia l'informazione di posizione che quella di colore, mediante l'istruzione *pcl::fromROSMsg*. Questo consentirà poi di elaborare la cloud utilizzando i metodi ed i tipi propri della libreria PCL.

Seguono poi tre differenti operazioni di filtraggio. La prima riguarda la rimozione dei punti che hanno valore *NaN* (Not a Number), in quanto inutilizzabili dal punto di vista computazionale. Questi punti identificano, nel processo di stereovisione, i punti non validi dell'immagine di profondità e, quindi, inutili al fine dell'elaborazione.

Il secondo filtro applicato consente la rimozione dei punti isolati, mediante la classe PCL predefinita *StatisticalOutlierRemoval*; tuttavia, dato che quest'ultima operazione è piuttosto onerosa dal punto di vista computazionale e non sempre apporta migliorie notevoli, si è reso possibile disattivarne la chiamata da file di lancio.

La terza operazione di filtraggio utilizza tre differenti filtri *PassThrough* per filtrare i punti che hanno coordinate (x, y, z) in un preciso intervallo di valori, in particolare per filtrare i punti troppo vicini alla telecamera e, quindi, soggetti a rumore di acquisizione. Infatti, la telecamera stereo ha una distanza minima prima della quale non riesce ad acquisire nulla in virtù dell'ottica del dispositivo: punti

che compaiono nella point cloud in questa zona cieca sono evidentemente frutto di rumore e, pertanto, occorre eliminarli.

La preelaborazione dei dati sensoriali del laser risulta più semplice, sia per la quantità di misure contenute in una scansione, sia per la più semplice struttura dei dati. Il primo passo risulta essere il cambio di sistema di riferimento in coordinate robot che, in questo caso, viene effettuato applicando la trasformazione ad ogni singolo punto, al fine di garantire la possibilità di applicare l'informazione di colore, non fornita dalla scansione laser, ad ogni punto, inserendo nei campi r, g e b i valori specificati da file di lancio. Il punto trasformato viene poi inserito nella point cloud di uscita del metodo che, successivamente, viene convertita dal formato ROS al formato PCL *PointCloud<PointXYZRGB>*.

Dopo queste prime fasi di preelaborazione, le due point cloud vengono fuse, dando la priorità al laser scanner, ovvero se lo stesso punto nello spazio è descritto da entrambe le point cloud, in virtù della maggior precisione del laser scanner, viene considerato il punto proveniente dalla point cloud del laser, rinunciando all'informazione precisa del colore. La fusione viene effettuata solamente se il laser è stato attivato da file di lancio. Infatti, al fine di rendere funzionante il sistema complessivo anche con l'utilizzo della sola testa stereo di telecamere, si è lasciata la possibilità all'utente di specificare, in fase di lancio del nodo *camera_mapping*, se attivare o meno i dati del laser e, quindi, le elaborazioni successive.

Al fine di accumulare le point cloud precedenti, che vengono memorizzate in un apposito vettore, con l'ultima point cloud acquisita, è necessario allinearle ad essa ricavando una matrice di trasformazione (operazione che, in letteratura, è definita *registration*). Ovviamente, questa fase non viene effettuata nel caso in cui, da file di lancio, si sia limitato ad 1 il numero di point cloud da memorizzare e, quindi, da accumulare.

L'operazione di allineamento, effettuata mediante la classe PCL *IterativeClosestPoint* (ICP), è piuttosto semplice dal punto di vista concettuale, ma meno dal punto di vista computazionale. Infatti, essa risulta piuttosto onerosa in assenza di informazioni di contorno utili per l'allineamento.

Sia N il numero di point cloud da memorizzare, specificato dal file di lancio, P_t^t la point cloud corrente espressa rispetto a sé stessa e P_{t-1}^i le point cloud al tempo passato i espresse rispetto all'ultima point cloud acquisita al tempo $t-1$,

con $i = t - N + 1, ..., t - 1$.

Nel caso semplice $N = 2$, occorre allineare P_{t-1}^{t-1} a P_t^t tramite la matrice di trasformazione T_t^{t-1}, ricavata tramite ICP passandogli come parametri di ingresso P_{t-1}^{t-1} e P_t^t. L'allineamento si riduce ad una moltiplicazione di una matrice per tutti i punti della point cloud:

$$P_t^{t-1} = T_t^{t-1} \cdot P_{t-1}^{t-1} \tag{3.1}$$

Ovviamente, alla point cloud corrente P_t non viene applicata nessuna trasformazione in quanto già allineata a sè stessa.

Nel caso $N = 3$, occorre allineare P_{t-1}^{t-2} e P_{t-1}^{t-1} alla point cloud corrente P_t^t. Come è possibile notare, la matrice di trasformazione da ricavare è sempre una sola, che viene poi applicata ad ogni point cloud passata, e cioè T_t^{t-1}. In formule:

$$\begin{aligned} P_t^{t-2} &= T_t^{t-1} \cdot P_{t-1}^{t-2} \\ P_t^{t-1} &= T_t^{t-1} \cdot P_{t-1}^{t-1} \end{aligned} \tag{3.2}$$

Risulta quindi immediato capire come, anche nel caso di N point cloud da allineare, la complessità computazionale abbia un aumento lineare $O(N)$, in quanto il calcolo della matrice di trasformazione tramite ICP, che fornisce il maggior contributo al carico di lavoro del processore, viene effettuato una sola volta.

Il listato 2 mostra le operazioni di memorizzazione e allineamento delle point cloud mediante l'uso di pseudocodice.

Dato che i tempi di elaborazione per ricavare la matrice di trasformazione T_t^{t-1} risultano direttamente proporzionali al numero di punti delle point cloud (complessità $O(N)$), al fine di velocizzare l'elaborazione, prima del calcolo le point cloud vengono discretizzate in voxel, ovvero suddivise in celle cubiche di dimensione assegnata. Ad ogni cella viene assegnato il valore medio dei pixel che cadono al suo interno. Questo passaggio non provoca perdita di informazioni, in quanto le point cloud vengono sottocampionate solamente per ricavare la matrice di trasformazione per poi riutilizzare quelle originali, ma ha notevoli vantaggi dal punto di vista dei tempi di elaborazione.

Le point cloud allineate vengono memorizzate in un vettore di dimensione N secondo una politica *FIFO* (First In First Out): quando il vettore si riempie e arriva una nuova point cloud da memorizzare, viene eliminata dal vettore la point cloud

```
1  if prima point cloud acquisita then
2  |  aggiungi la point cloud al vettore;
3  else if numero di point cloud da memorizzare = 1 then
4  |  svuota il vettore cloudsVector;
5  |  aggiungi la point cloud al vettore;
6  else
7  |  if cloudsVector pieno then
8  |  |  rimuovi point cloud più vecchia posizionata in testa;
9  |  end
10 |  calcola con ICP la matrice di trasformazione per allineare l'ultima point
   |  cloud inserita nel vettore alla point cloud corrente;
11 |  forall the point cloud nel vettore do
12 |  |  applica la matrice di trasformazione alla point cloud per allinearla
   |  |  alla point cloud corrente;
13 |  end
14 |  inserisci la point cloud corrente in coda al vettore;
15 end
```

Algorithm 2: Pseudo codice delle operazioni di allineamento delle point cloud.

temporalmente più vecchia e viene inserita, in coda, l'ultima arrivata. In questo modo, la point cloud accumulata rappresenta una memoria a breve termine dei dati sensoriali ottenuti dal robot durante la navigazione. Dato che, ad ogni nuovo arrivo, occorre riallineare e, quindi, modificare tutte le point cloud del vettore, va notato come esso non venga svuotato, in quanto le point cloud in esso contenute risultano tutte allineate alla penultima point cloud acquisita. Sarà quindi sufficiente applicare la matrice di trasformazione appena calcolata ad ogni singolo elemento del vettore P_{t-1}^{t-i} per allineare tutte le point cloud all'ultima acquisita, risparmiando il calcolo di $N-1$ matrici di trasformazione.

Una volta allineate tutte le point cloud, è possibile procedere all'accumulo, che consiste nel fondere le N point cloud in una unica che, a questo punto, conterrà maggiori informazioni e una memoria a breve termine. Infatti, l'accumulo serve proprio per cercare di riempire quelle zone della point cloud in cui non viene acquisita nessuna informazione in virtù della bassa disparità tra i punti della scena, ottenendo quindi una point cloud più densa.

Tuttavia, la fusione di point cloud non allineate sarebbe deleteria, in quanto andrebbe a sovrapporre informazioni sensoriali relative a scene diverse, generan-

do una nuvola di punti difficile da interpretare ed elaborare. Pertanto, prima di fondere due point cloud (P_t^{t-i} e P_t^t), viene effettuata un'associazione tra i punti di P_t^{t-i} e P_t^t, utilizzando l'implementazione inclusa in PCL dell'algoritmo KdTree. Se il numero di punti di P_t^{t-i} che hanno un corrispondente in P_t^t supera una certa soglia percentuale, impostata al 75 %, allora le due point cloud vengono fuse, altrimenti il passaggio viene saltato e si mantiene l'ultima point cloud acquisita.

Questo passaggio conclude le operazioni effettuate sulla point cloud, che a questo punto può essere pubblicata come messaggio ROS per essere poi utilizzata dal nodo descritto successivamente per la costruzione della mappa di navigazione. Infatti, grazie al nodo *camera_mapping*, si sono integrate in un'unica struttura dati *pcl::PointCloud<PointXYZRGB>* tutte le informazioni derivanti dai dati sensoriali delle telecamere e del laser scanner, opportunamente filtrate e ripulite dal rumore di acquisizione, e si sono uniti dati acquisiti in tempi diversi grazie all'allineamento e all'accumulo. La point cloud risultante, oltre a essere pubblicata in un messaggio ROS dopo essere stata convertita nel tipo di dato *sensor_msgs::PointCloud2* di ROS, può essere anche salvata in un file *PCD* (Point Cloud Data), abilitando l'apposita opzione dal file di lancio del nodo. Questa funzione è utile per poter visionare, in qualsiasi momento e *offline*, i dati sensoriali acquisiti ed elaborati dal sistema di percezione.

3.4 Costruzione della mappa 3D

Dopo che i nodi precedenti hanno acquisito e perfezionato i dati sensoriali, memorizzandoli in una point cloud, lo scopo di questa fase di elaborazione è quello di utilizzarli per la navigazione e, in particolare, per costruire una mappa tridimensionale dell'ambiente, consentendo al robot di individuare gli ostacoli da evitare e lo spazio libero in cui, invece, può muoversi in sicurezza. Questa mappa verrà poi utilizzata per pianificare il moto del robot verso il goal impostato al passaggio successivo.

La costruzione della mappa e la pianificazione del moto vengono effettuate dal nodo ROS predefinito *move_base*, che include il nodo *costmap_2d* per la costruzione della mappa e il nodo *nav_core* per la pianificazione. Di seguito si descriverà il nodo *costmap_2d* lasciando alla sezione 12 la descrizione del nodo

di pianificazione ai fini della navigazione. In questa operazione risulta di vitale importanza l'espressione della point cloud nel sistema di riferimento solidale al robot, al fine del calcolo corretto delle distanze del robot dagli ostacoli proiettati.

Il nodo *costmap_2d* prende in ingresso la point cloud 3D pubblicata dal nodo precedente e ne ricava una *occupancy grid*, ovvero una matrice bidimensionale in cui ogni cella viene classificata come occupata, indirettamente occupata o sconosciuta. In pratica, quello che viene fatto è una proiezione della nuvola di punti tridimensionale in uno spazio bidimensionale, proiettando gli ostacoli sul piano di navigazione del robot.

La mappa così costruita risulta essere un esempio di fusione sensoriale ottenuta per sovrapposizione. Infatti, l'operazione è svolta proiettando i dati provenienti da diverse sorgenti sensoriali sullo stesso piano. Sebbene il nodo *costmap_2d* preveda la possibilità di specificare diverse sorgenti sensoriali, come descritto in seguito, si è preferito attuare la fusione sensoriale separatamente, per poi fornire in ingresso al nodo la point cloud risultante come unica sorgente per la creazione della mappa di navigazione, al fine di garantire maggiore personalizzazione nei processi di preelaborazione e filtraggio.

Nonostante la mappa di navigazione risulti in buona parte ottenuta dai dati del laser, in virtù del fatto che il campo di vista della telecamera è molto più limitato, un importante contributo viene fornito dalla visione stereo nella zona frontale al robot. Infatti, in questa operazione si può notare il miglioramento introdotto dalla fusione sensoriale, e cioè che, mentre con il solo laser scanner alcuni ostacoli non verrebbero identificati (pensiamo al piano di un tavolo posto ad un'altezza differente dal piano di scansione del laser), l'aggiunta dei dati provenienti dalla testa stereo consente di completare la griglia di occupazione inserendovi ostacoli aggiuntivi. Ovviamente, nell'operazione di proiezione, il nodo tiene in considerazione anche l'altezza del robot, evitando di considerare come ostacoli oggetti posti troppo in alto, a cui il robot può passare sotto in sicurezza.

I parametri da impostare per l'utilizzo del nodo *costmap_2d*, specificabili da file *launch*, sono riassunti di seguito:

- *obstacle_range*: distanza massima, in metri, degli ostacoli per essere inseriti nella mappa (funzione della distanza massima a cui il sensore può percepire);

- *raytrace_range*: distanza massima, in metri, in cui il nodo cerca di pulire la mappa da ostacoli non più percepiti;

- *footprint*: letteralmente *impronta*, ovvero l'occupazione del robot nella mappa (descritta come poligono in metri), utile per il calcolo degli ostacoli indiretti;

- *inflation_radius*: raggio, in metri, di occupazione indiretta degli ostacoli. Infatti, visto che il robot viene approssimato, ai fini della navigazione, come un punto nello spazio, tenendo in considerazione la sua *impronta*, nelle zone vicine agli ostacoli il robot non può andare in quanto il suo *corpo*, più grande di un punto, si scontrerebbe con l'ostacolo. Questi sono gli *ostacoli indiretti* e questo raggio, direttamente proporzionale all'impronta del robot, delinea a che distanza il robot deve passare dagli ostacoli;

- *observation_sources*: nome dei sensori, descritti in seguito, i cui dati sensoriali verranno utilizzati per la costruzione della mappa;

- *nome_sensore*: impostazioni di ogni sorgente sensoriale e, in particolare:

  - *sensor_frame*: nome del sistema di coordinate in cui i dati sensoriali sono riferiti;
  - *data_type*: tipo di dato in cui il dato sensoriale viene pubblicato;
  - *topic*: nome del topic con il quale il dato sensoriale viene pubblicato;
  - *marking*: attiva o disattiva la possibilità per il nodo di marcare lo stato di un pixel della griglia di occupazione come *occupato*;
  - *clearing*: attiva o disattiva la possibilità per il nodo di marcare lo stato di un pixel della griglia di occupazione come *libero*;
  - *observation_persistence*: tempo, in secondi, dopo il quale un ostacolo non più identificato viene rimosso dalla griglia di occupazione;
  - *expected_update_rate*: frequenza di aggiornamento della griglia di occupazione.

- *local_costmap* o *global_costmap*: elenco dei parametri di configurazione della mappa locale, ovvero quella che tiene in considerazione il moto e le percezioni del robot nel breve periodo, o globale, ovvero quella che viene costruita utilizzando tutti i dati provenienti dai sensori dall'avvio del robot; in particolare:
  - *global_frame*: sistema di riferimento utilizzato dalla mappa locale;
  - *robot_base_frame*: sistema di riferimento posizionato sul robot mobile;
  - *update_frequency*: frequenza di aggiornamento (in *Hz*) della mappa locale;
  - *publish_frequency*: frequenza di pubblicazione (in *Hz*) della mappa locale;
  - *static_map*: attiva o disattiva l'utilizzo di una mappa statica;
  - *rolling_window*: attiva o disattiva l'utilizzo di finestre scorrevoli per l'aggiornamento della mappa, che consentono di eliminare gli ostacoli che, mano a mano che il robot si muove, non sono più percepiti nelle vicinanze; se attivata, la mappa rimane centrata sul robot;
  - *width*: larghezza massima, in metri, della mappa costruita;
  - *height*: altezza massima, in metri, della mappa costruita;
  - *resolution*: risoluzione, in metri, della mappa costruita.

Una volta avviato, il nodo pubblica messaggi di tipo *nav_msgs/GridCells* su tre differenti topic, contenenti il risultato dell'elaborazione:

- *obstacles*, mappa contenente le celle occupate da un ostacolo;
- *inflated_obstacles*, mappa contenente le celle occupate dai cosiddetti *ostacoli indiretti*, ovvero la porzione di spazio attorno agli ostacoli reali tale per cui, se il centro del robot si posizionasse su di essa con almeno uno specifico orientamento, il corpo del robot urterebbe gli ostacoli;
- *unknown_space*: mappa contenente tutte le altre celle, di cui non si conosce lo stato e che, pertanto, si considerano libere da ostacoli.

La mappa costruita apparirà, all'interno del visualizzatore Rviz, come mostrato in fig. 3.10. Ogni cella ha lato pari alla risoluzione della mappa impostata con il parametro *resolution*. Le celle rosse rappresentano gli ostacoli, pubblicate sul topic *obstacles*, mentre quelle blu gli ostacoli indiretti pubblicati sul topic *inflated_obstacles*.

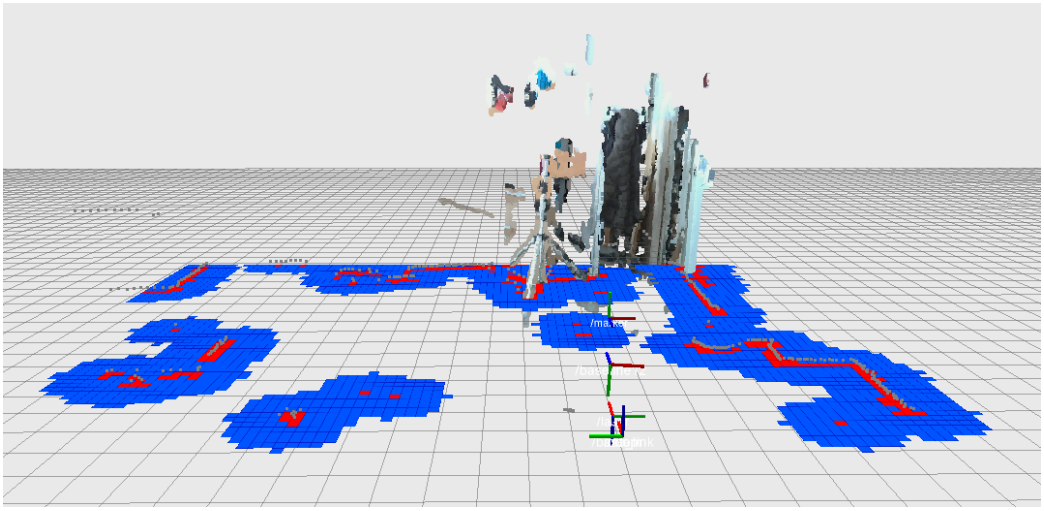


Figura 3.10: Visualizzazione in Rviz della mappa di navigazione costruita.

Questa mappa può essere utilizzata sia per la navigazione che per l'identificazione degli oggetti, comportamenti descritti in seguito.

3.5 Identificazione di oggetti

La point cloud creata nei passi precedenti non è utilizzata solamente ai fini della navigazione, ma anche per l'identificazione degli oggetti presenti sulla scena. Al fine di fornire al robot un'ulteriore autonomia, ovvero quella di impostare automaticamente la posizione da raggiungere, è stato sviluppato il nodo *cloud_clustering*, il cui compito principale è estrarre dalla point cloud gli oggetti presenti sulla scena, sotto la condizione che siano opportunamente separati, e individuare quello dominante in primo piano. L'oggetto ricercato dovrà soddisfare alcuni vincoli dimensionali impostati come parametri di configurazione del nodo.

La scelta di estrarre l'oggetto soddisfacente i vincoli impostati più vicino al robot è stata effettuata per garantire la migliore percezione possibile da parte dei sensori. Infatti, come già detto in precedenza, i risultati migliori nell'acquisizione dalla telecamera si hanno frontalmente al robot. Questa scelta, tuttavia, non è limitativa, in quanto se l'oggetto soddisfacente i vincoli non viene trovato nelle posizioni frontali vicine al robot, la ricerca continua con gli oggetti più lontani, ben sapendo che la percezione non è più ottimale e, quindi, la corretta identificazione risulterà più difficoltosa.

Una volta identificato l'oggetto, il nodo provvede a pubblicare su un apposito messaggio ROS il punto da raggiungere. In particolare, il *goal* viene fissato nella parte posteriore all'oggetto, con un'orientazione rivolta verso l'oggetto stesso, al fine di consentire al robot di acquisirne un'ulteriore vista.

Le operazioni svolte dal nodo *cloud_clustering* sono riassunte in fig. 3.11, mentre l'algoritmo 3 mostra i passi eseguiti in pseudocodice.

Tralasciando la descrizione del primo passo di conversione della point cloud di input in formato PCL, già descritto nei passi precedenti, il passo successivo consente di identificare il piano dominante presente sulla scena e rimuoverlo, dato che presumibilmente sarà il pavimento sul quale si muove il robot.

Per svolgere questa funzione, viene utilizzata la classe *SACSegmentation* inclusa nella libreria PCL, che consente di impostare il tipo e la direzione del piano ricercato, il metodo utilizzato (*RANSAC*), oltre ad altri parametri di configurazione, per restituire una point cloud contenente tutti i punti della point cloud originale non appartenenti al piano dominante.

Dato che, a seconda dell'orientazione della testa stereo di telecamere e del colore del pavimento, alcune volte la point cloud di ingresso non contiene nessun piano dominante, al fine di ridurre il tempo di elaborazione, è possibile disattivare l'utilizzo di questa funzione direttamente da file di lancio del nodo, attraverso il parametro booleano *remove_floor*.

L'estrazione vera e propria degli oggetti, cuore di questo nodo, viene effettuata tramite la classe PCL predefinita *EuclideanClusterExtraction* che, dopo aver impostato parametri di tolleranza e dimensione minima di ogni oggetto, consente di produrre un vettore di oggetti di classe *pcl::PointIndices* che permettono di dividere la point cloud di ingresso in cluster. La suddivisione della point cloud in

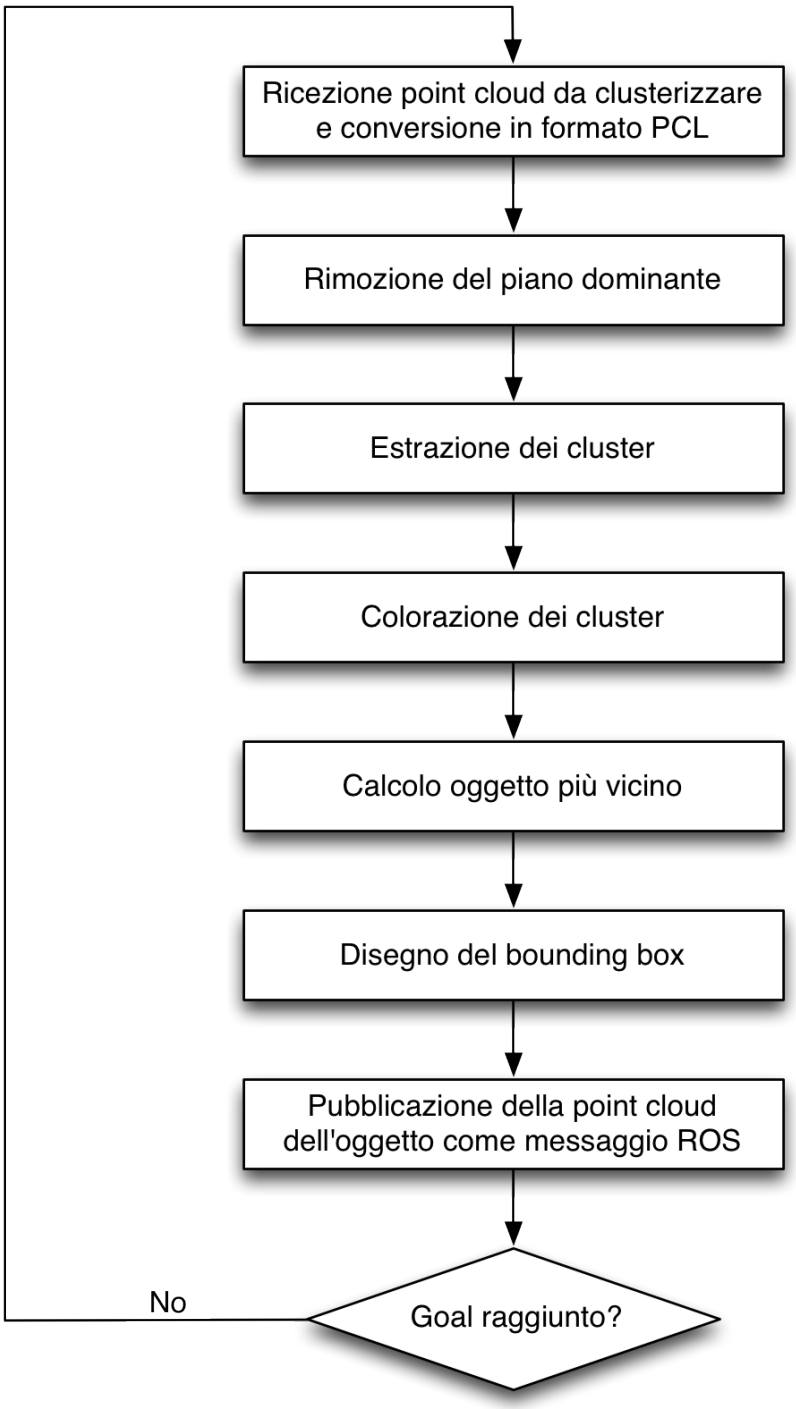


Figura 3.11: Diagramma delle operazioni svolte dal nodo *cloud_clustering*.

```
1  leggi i parametri di configurazione da file di lancio;
2  while ros::ok() do
3      leggi i dati sensoriali fusi da ROS e memorizzali in una point cloud;
4      if rimozione piano dominante attivata then
5          individua e rimuovi il piano dominante parallelo al terreno;
6      end
7      estrai i cluster della point cloud utilizzando EuclideanClusterExtracion;
8      forall the cluster do
9          inizializza una point cloud per il cluster e una per l'oggetto;
10         inizializza la distanza minima minDist = 100.0;
11         forall the pixel della point cloud do
12             if minDist > distanza euclidea del pixel then
13                 aggiorna minDist con la distanza del pixel;
14             end
15             inserisci il pixel nella point cloud del candidato oggetto;
16             genera un colore da associare al cluster e ricolora il pixel;
17             inserisci il pixel nella point cloud del cluster;
18         end
19         aggiungi il cluster alla point cloud contenente tutti i cluster;
20         calcola la dimensione del bounding box con getMinMax3D;
21         if non è stato trovato un oggetto & non navigo verso il goal then
22             if il cluster soddisfa i vincoli impostati da utente then
23                 considera il cluster corrente come oggetto ricercato;
24                 inizializza la point cloud contenente l'oggetto;
25             end
26         else if è stato trovato un oggetto & non navigo verso il goal then
27             if il cluster soddisfa i vincoli impostati da utente then
28                 considera il cluster corrente come oggetto ricercato;
29             end
30         end
31     end
32     disegna il bounding box attorno al cluster nella point cloud relativa;
33     if è stato trovato un oggetto then
34         calcola il goal e lo pubblica su ROS;
35     end
36     pubblica la point cloud contenente in cluster come messaggio ROS;
37 end
```

Algorithm 3: Pseudo codice del nodo *cloud_clustering*.

cluster avviene nel seguente modo:

1. inizializzazione di una lista vuota di cluster C e di una coda di punti da analizzare Q;
2. per ogni punto $p_i \in P$, esecuzione dei seguenti passi:
   - aggiunta di p_i alla coda Q;
   - per ogni punto $p_i \in Q$:
     - ricerca dell'insieme di punti P_k^i vicini a p_i in una sfera di raggio $r < d_{th}$, dove d_{th} è la tolleranza impostata da parametro di file di lancio;
     - per ogni vicino $p_i^k \in P_i^k$, controllo se è già stato elaborato, altrimenti aggiunta alla coda Q;
   - aggiunta di Q alla lista di cluster C e reset di Q a una lista vuota;
3. terminazione quando tutti i punti $p_i \in P$ sono stati elaborati e, quindi, sono ora parte della lista di cluster C, che diventa il risultato dell'elaborazione.

Per ogni cluster individuato, si procede alla colorazione dei pixel ad esso appartenenti tramite la funzione *valueToRGB*, che consente di generare una terna di parametri R, G, B a seconda dell'indice identificativo del cluster considerato. La colorazione dei pixel è utile per meglio verificare, a colpo d'occhio, il corretto funzionamento dell'algoritmo di estrazione dei cluster.

I cluster identificati vengono inseriti in una point cloud, che viene poi pubblicata come messaggio ROS al fine di agevolare la verifica del buon funzionamento del nodo. Un esempio di point cloud generata è mostrato in fig. 3.12.

Parallelamente, si procede alla ricerca dell'oggetto più vicino che soddisfi determinate condizioni relativamente alle dimensioni e alla distanza dal robot:

1. per ogni cluster c_i identificato:
   - calcolo del punto più vicino di coordinate (x_p, y_p, z_p), utilizzando la distanza euclidea $d = x_p^2 + y_p^2 + z_p^2$;
   - calcolo dei valori x, y, z minimi e massimi dei punti appartenenti a c_i utilizzando la funzione *pcl::getMinMax3D*;

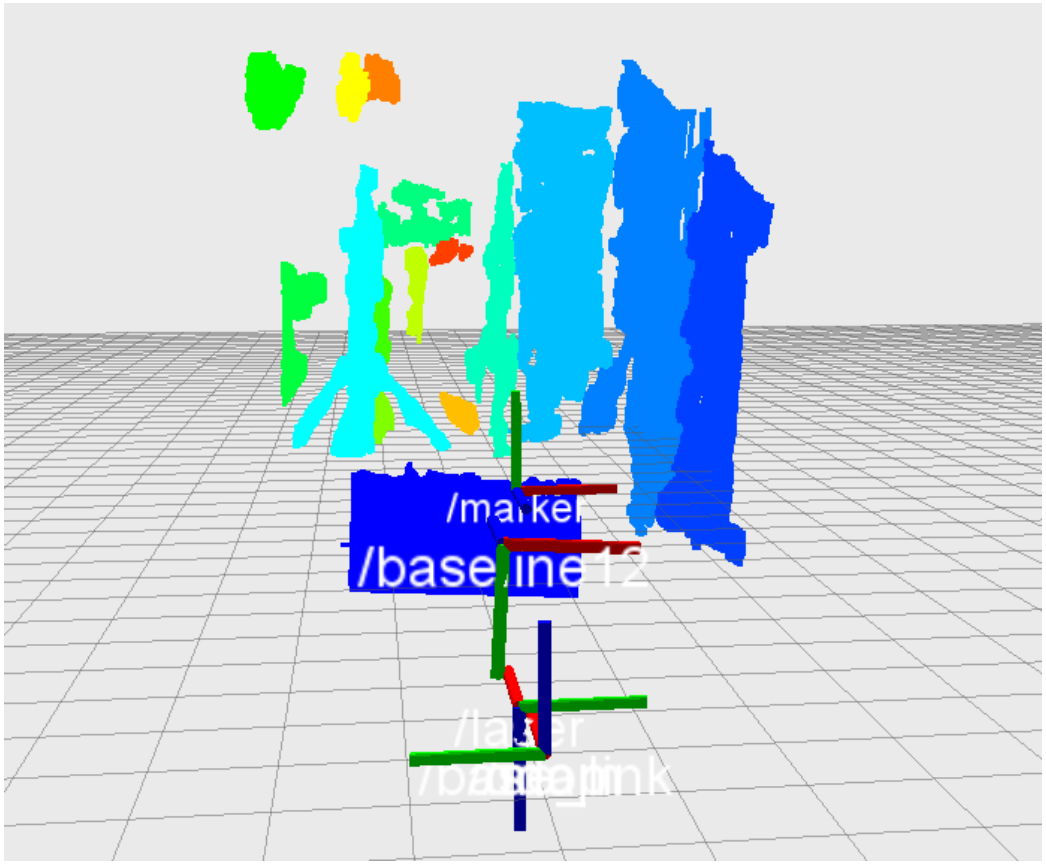


Figura 3.12: Point cloud contenente i cluster identificati.

- calcolo delle dimensioni dell'oggetto tramite differenza tra i valori massimi e i valori minimi appena trovati;
- controllo se le dimensioni dell'oggetto e la sua distanza sono compatibili con quelle impostate e, nel caso, inizializzazione della point cloud contenente l'oggetto

2. calcolo del centroide dell'oggetto più vicino al robot tramite la funzione predefinita *pcl::compute3DCentroid*;
3. aggiunta alla point cloud dell'oggetto di un bounding box, costruito aggiungendo punti sul contorno del parallelepipedo che circoscrive l'oggetto, distanziati di un valore impostato da file di lancio;
4. pubblicazione della point cloud come messaggio ROS.

Successivamente, la point cloud dell'oggetto più vicino identificato viene utilizzata per impostare il *goal*, ovvero il punto verso il quale far navigare il robot. Al fine di costruire una vista più completa dell'oggetto, infatti, il *goal* viene impostato in modo da far vedere l'oggetto al robot sia nella sua parte frontale che in quella posteriore. Pertanto, una volta identificato l'oggetto e, in particolare, le coordinate del suo centroide (x_c, y_c, z_c), e calcolate le sue dimensioni w_c, h_c, d_c (larghezza, altezza e profondità), la posizione del *goal* viene impostata alle seguenti coordinate:

$$\begin{aligned} x_{goal} &= x_c + d_c + x_0 \\ y_{goal} &= y_c \\ z_{goal} &= z_c \end{aligned} \tag{3.3}$$

dove x_0 è un parametro impostato da utente che indica la distanza dalla quale il robot si deve fermare dall'oggetto. Per quanto riguarda, invece, l'orientamento, esso viene rappresentato da un quaternione con i seguenti valori, che consentono di far arrivare il robot rivolto frontalmente all'oggetto.

$$\begin{aligned} q_x &= 0.0 \\ q_y &= 0.0 \\ q_z &= 1.0 \\ q_w &= 0.0 \end{aligned} \tag{3.4}$$

La posa del punto da raggiungere, struttura dati composta da posizione e orientamento, è poi pubblicata come messaggio ROS per le successive utilizzazioni.

Per svolgere tutte le operazioni precedentemente descritte, il nodo necessita dell'impostazione di svariati parametri di configurazione riassunti di seguito:

- *cloud_topic*: nome del topic contenente la Point Cloud dalla quale identificare gli oggetti;

- *clusters_topic*: nome del topic sul quale verrà pubblicato l'elenco dei clusters degli oggetti identificati;

- *object_cluster_topic*: nome del topic sul quale verrà pubblicato il cluster dell'oggetto più vicino identificato;
- *object_topic*: nome del topic sul quale verrà pubblicato l'oggetto più vicino identificato, racchiuso in un bounding box;
- *robot_frame*: nome del sistema di riferimento posizionato sul robot;
- *map_frame*: nome del sistema di riferimento della mappa globale;
- *planner_topic*: nome del topic sul quale vengono pubblicati i comandi per il robot da parte del pianificatore di traiettorie;
- *goal_topic*: nome del topic sul quale viene pubblicato il punto da raggiungere (*goal*);
- *sac_max_iterations*: numero massimo di iterazioni per l'algoritmo di identificazione del piano dominante;
- *sac_distance_thr*: soglia di distanza per l'identificazione del piano dominante;
- *cluster_toll*: distanza massima (in *m*) tra i pixel appartenenti allo stesso cluster;
- *min_cluster_size*: dimensione minima, in numero di punti, di un cluster per non esser scartato;
- *max_cluster_size*: dimensione massima, in numero di punti, di un cluster per non esser scartato;
- *remove_floor*: abilita o disabilita la rimozione del piano dominante;
- *savePCD*: abilita o disabilita il salvataggio dell'elenco dei cluster in un'immagine PCD;
- *bounding_box_resolution*: distanza (in *m*) dei punti identificanti il bounding box dell'oggetto;

- *distance_from_object_X*: distanza (in *m*) dall'oggetto identificato a cui impostare il *goal*;
- *objectMaxDistance*: distanza massima (in *m*) dell'oggetto da identificare come oggetto da aggirare;
- *objectMinWidth*: larghezza minima (in *m*) dell'oggetto;
- *objectMaxWidth*: larghezza massima (in *m*) dell'oggetto;
- *objectMinHeight*: altezza minima (in *m*) dell'oggetto;
- *objectMaxHeight*: altezza massima (in *m*) dell'oggetto;
- *objectMinDepth*: profondità minima (in *m*) dell'oggetto;
- *objectMinDepth*: profondità massima (in *m*) dell'oggetto;
- *objectSizeThreshold*: volume minimo (in m^3) dell'oggetto;
- *ROSrate*: frequenza di lavoro (in Hz) del nodo.

3.6 Navigazione

A questo punto, il robot ha a disposizione tutti i dati utili per procedere ad una navigazione sicura nello spazio circostante. Il comportamento di navigazione svolge due funzioni: la pianificazione del moto e l'invio dei comandi agli attuatori dei motori.

La pianificazione del moto viene effettuata dal nodo *nav_core*, parte integrante del package *navigation* di ROS sviluppato da Eitan Marder-Eppstein [6]. Questo nodo, in particolare, mette a disposizione un pianificatore di traiettorie locale e uno globale, facilmente configurabili, che, ricevendo in ingresso la posizione attuale del robot, ottenuta dal dato odometrico, e il punto da raggiungere (goal), impostato al passo precedente, tenendo in considerazione la mappa di navigazione appena costruita, consentono di calcolare una traiettoria che permetta al robot di muoversi nell'ambiente in sicurezza per raggiungere l'obiettivo. La traiettoria calcolata dovrà essere la più breve possibile e, allo stesso tempo, dovrà garantire

il passaggio più lontano possibile dagli ostacoli percepiti. La traiettoria impostata viene poi tradotta in velocità lineari e angolari, da applicare al robot per seguirla, pubblicate sul topic *planner*.

In fig. 3.13 viene mostrata l'interfaccia del nodo *nav_core*. Si noti che il nodo mette a disposizione anche un comportamento di recovery, non utilizzato in questo progetto di tesi.

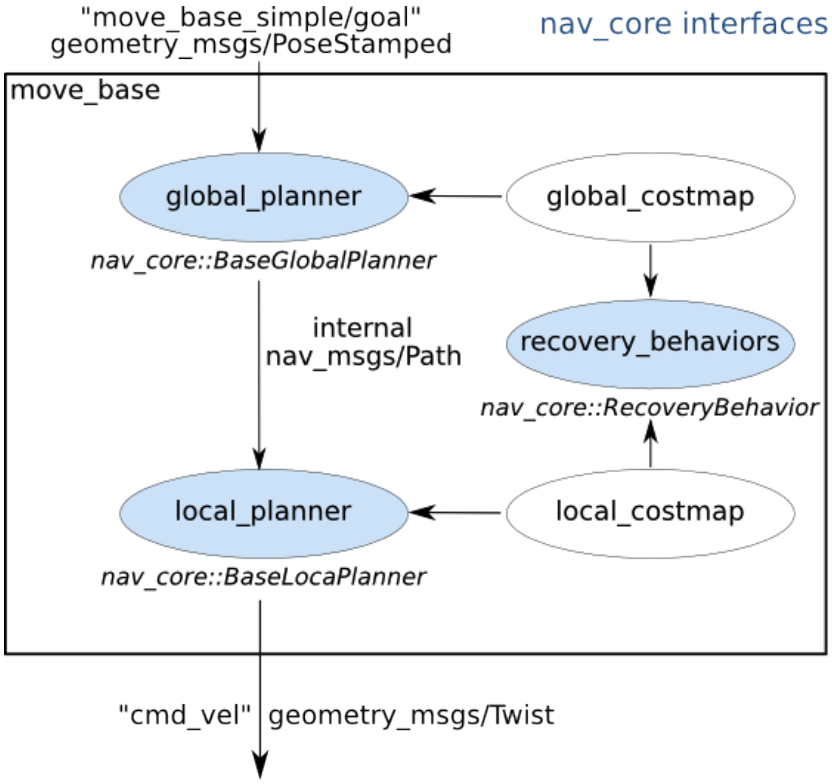


Figura 3.13: Interfaccia del nodo *nav_core*.

Per gli scopi di questa tesi, viene utilizzato prevalentemente il pianificatore locale, che utilizza un'implementazione dell'algoritmo di pianificazione Dynamic Window Approach (DWA) [22], i cui macropassi di esecuzione sono riassunti nell'algoritmo 4.

Il nodo *nav_core* viene impostato tramite file di lancio contenente i seguenti parametri, applicati al parametro *TrajectoryPlannerROS*:

```
1  while true do
2      campionamento discreto dello spazio di controllo (dx, dy, dθ) del robot;
3      forall the velocità campionate do
4          simulazione, dallo stato corrente del robot, dell'applicazione, per un
           breve periodo di tempo, della velocità corrente al robot, per capire
           cosa accadrebbe (punto di arrivo, urto con ostacoli, ecc.);
5          if la traiettoria incontra un ostacolo then
6              scarto della traiettoria;
7          else
8              valutazione, mediante l'assegnazione di un punteggio, della
               traiettoria seguita, usando una metrica che incorpori
               caratteristiche come la vicinanza agli ostacoli, la vicinanza
               all'obiettivo e la velocità;
9          end
10     end
11     salvataggio della traiettoria con il punteggio più alto e invio della
       velocità associata al robot mobile;
12 end
```

Algorithm 4: Pseudocodice delle elaborazioni eseguite dall'algoritmo DWA.

- *min_vel_x*: velocità lineare minima, in m/s, applicabile al robot lungo l'asse x;
- *max_vel_x*: velocità lineare massima, in m/s, applicabile al robot lungo l'asse x;
- *max_rotational_vel*: velocità rotazionale massima, in rad/s, applicabile al robot;
- *min_in_place_rotational_vel*: velocità rotazionale minima, in rad/s, applicabile al robot quando è fermo (velocità lineare nulla);
- *acc_lim_x*: accelerazione lineare massima, in m/s^2, applicabile al robot lungo l'asse x;
- *acc_lim_y*: accelerazione lineare massima, in m/s^2, applicabile al robot lungo l'asse y;

- *acc_lim_th*: accelerazione rotazionale massima, in rad/s^2, applicabile al robot;

- *holonomic_robot*: attiva o disattiva la pianificazione del moto per robot olonomi, ovvero quei robot mobili che possono muoversi istantaneamente in tutte le direzioni.

Una vista complessiva del sistema di navigazione è mostrata in fig. 3.14, dove si può vedere la mappa di navigazione presentata in precedenza, l'oggetto identificato come più vicino racchiuso in un bounding box, il *goal* da raggiungere con relativa orientazione (freccia gialla) e la traiettoria pianificata al passo precedente (in verde).

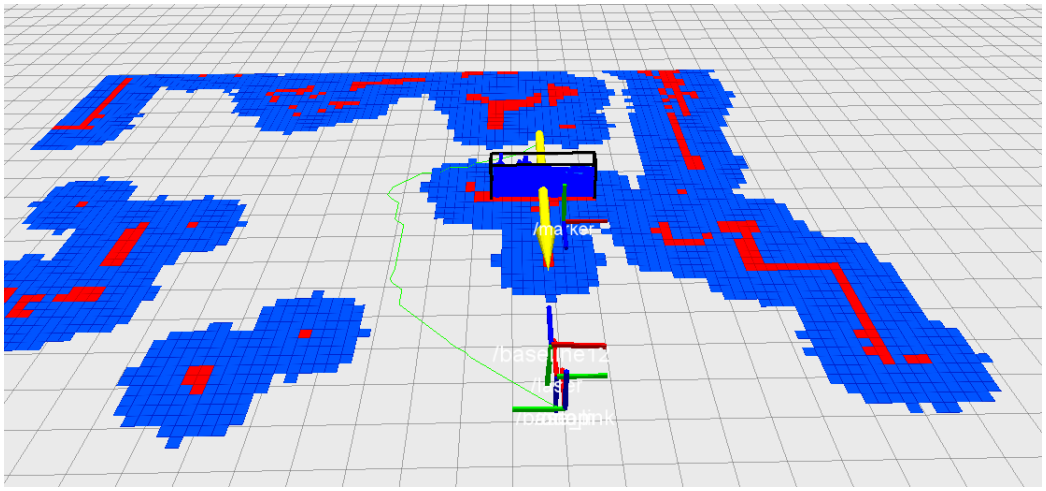


Figura 3.14: Mappa locale di navigazione con l'aggiunta del *goal* e della traiettoria pianificata per raggiungerlo.

La seconda funzione del behaviour di navigazione è svolta dal nodo *robot_move*, che prende in ingresso le velocità pubblicate dal pianificatore sul topic *planner*, controlla con un comportamento di sicurezza che utilizza i dati del laser scanner che il robot non stia andando a scontrarsi con un ostacolo e imposta i comandi di velocità al topic *cmd_vel*, che il driver del robot *p2os_driver* invierà sulla porta seriale del robot per farlo muovere di conseguenza. Il nodo viene avviato da un file di lancio, contenente anche i valori impostati nei seguenti parametri di configurazione:

- *lms100_topic*: nome del topic su cui il driver del laser LMS100 pubblica i dati sensoriali;
- *planner_topic*: nome del topic su cui il nodo di pianificazione pubblica le velocità da impostare al robot per inseguire la traiettoria;
- *cmd_vel_topic*: nome del topic su cui pubblicare i comandi di movimentazione del robot;
- *min_dist*: minima distanza di sicurezza (in m) da tenere dagli ostacoli rilevati dal laser scanner;
- *ROSrate*: frequenza di lavoro (in Hz) del nodo.

Il comportamento di sicurezza serve al fine di fermare il robot nel remoto caso in cui la mappa non identifichi un ostacolo sulla traiettoria impostata. Infatti, può accadere che, a causa della complessità di elaborazione della mappa, il robot si muova utilizzando dati sensoriali riferiti al passato (quando l'elaborazione è iniziata), non acquisendo quindi i nuovi ostacoli che, eventualmente, si sono inseriti sulla scena. Dato che il laser è molto più rapido a pubblicare i dati sensoriali rispetto al nodo di creazione della mappa, si è preferito inserire un comportamento di *wall avoidance* che utilizza i soli dati sensoriali del laser per evitare gli ostacoli e, nel caso, fermare il robot. Tuttavia, come si vedrà, nelle prove sperimentali questo comportamento entra in funzione un numero molto limitato di volte.

Il funzionamento del comportamento di *wall avoidance* risulta piuttosto semplice: il nodo riceve i dati del laser scanner sotto forma di vettore di numeri in virgola mobile in cui, ogni valore, rappresenta la distanza dell'ostacolo più vicino in una particolare direzione (angolo) del laser. Lo spazio di scansione del laser viene suddiviso in due parti (destra e sinistra) e per ciascuna viene calcolato l'ostacolo più vicino. Se l'ostacolo più vicino ha una distanza inferiore ad una certa soglia (*min_dist*) definita tra i parametri del nodo, il robot si ferma e inizia a girare dalla parte opposta, ovvero quella libera. In caso contrario, trasmette agli attuatori dei motori i comandi impostati dal pianificatore di traiettoria per navigare verso il *goal*.

Una volta arrivato a destinazione, il robot si fermerà e attenderà nuovi comandi, come ad esempio l'identificazione di un nuovo oggetto da aggirare.

Questo passo, conclude le operazioni svolte dal sistema realizzato in questa tesi. Nel capitolo successivo si mostreranno i risultati ottenuti da alcune prove condotte in laboratorio, al fine di verificarne il reale funzionamento.

Capitolo 4

Prove sperimentali

Per testare il sistema complessivo realizzato, sono state condotte alcune prove sperimentali presso il laboratorio di Robotica (RimLab) del Dipartimento di Ingegneria dell'Informazione dell'Università di Parma. Negli esperimenti è stata utilizzata la piattaforma robotica Pioneer 3DX dotata di telecamere stereo e laser scanner, descritta nel capitolo 2.1. Per l'elaborazione si è utilizzato un computer portatile MacBook Pro 13" dotato di processore Intel Core i7 dual core da 2.7 GHz, 8 GB di memoria RAM DDR3 1333 MHz, su cui si è installato Ubuntu 12.04 64 bit e ROS Fuerte.

Al fine di quantificare la precisione delle misurazioni di distanze effettuate dalla point cloud contenente le informazioni provenienti da entrambe le sorgenti sensoriali, si è posto un generico oggetto (scatola di dimensioni 58.2x30.3x8.6 cm) in posizione frontale al robot e si è misurata la sua distanza lineare rilevata dall'apposito nodo. Le prove sono state effettuate con l'oggetto a distanze diverse, al fine di verificare la dipendenza dell'errore di misura dalla distanza. I risultati sono riassunti in tab. 4.1.

Distanza reale $[m]$	**Distanza misurata** $[m]$	**Errore** $[\%]$
1.00	0.99	1.000
2.03	2.00	1.478
2.52	2.48	1.587

Tabella 4.1: Errore percentuale medio di misurazione.

Come è possibile notare, la presenza del laser scanner nella dotazione senso-

riale consente di ottenere misurazioni di distanze piuttosto precise, con un errore percentuale che si mantiene nettamente sotto al 2 % anche per distanze medie. Su distanze superiori a 2.5 m interviene un limite progettuale del sistema di visione stereo, tarato per una maggior precisione sulle brevi distanze a discapito delle grandi distanze. Infatti, ai fini della navigazione e della costruzione della mappa di occupazione locale, risulta più interessante la percezione corretta di ostacoli relativamente vicini al robot.

Appurato il corretto funzionamento del sistema di misurazione, si è proceduto ad effettuare una prova del sistema complessivo. La prova consiste nell'esplorazione dell'ambiente mostrato in fig. 4.1, nell'identificazione di un oggetto target e nella pianificazione di una traiettoria per posizionare il robot dal lato opposto dell'oggetto. L'oggetto target è selezionato secondo criteri di attenzione, ossia scegliendo quello maggiormente visibile al robot. Negli esperimenti è stata utilizzata una scatola di dimensioni 58.2x30.3x8.6 cm, già utilizzata nel test precedente.

Figura 4.1: Scena reale in cui si muove il robot.

Il robot è stato posto ad una distanza di circa 2 m dal target e orientato in modo tale che la scatola sia visibile nel lato destro dell'immagine acquisita. Al fine di verificare i reali vantaggi derivanti dall'utilizzo della fusione sensoriale dei dati provenienti dai due sensori nella navigazione, è stato inserito nell'ambiente un ostacolo (sedia) non percepibile dal laser scanner, in quanto composto da parti posizionate su un piano diverso dal piano di scansione, ma tale da costituire comunque un pericolo di urto con il robot. Infatti, lo spazio sotto alla sedia è sufficientemente largo, e quindi non identificato come ostacolo dal laser scanner, ma non sufficientemente alto per far sì che il robot vi passi. Pertanto, l'ausilio della telecamera per percepire l'ostacolo nella sua completezza è fondamentale.

Sulla scena erano presenti ulteriori oggetti, come vari scatoloni, un attaccapanni, un cestino, poco interessanti ai fini del test, ma utili a rendere più generale possibile l'ambiente in cui il robot si muove. Tali oggetti sono stati considerati come ostacoli, contribuendo alla costruzione della mappa di occupazione utilizzata per la navigazione.

Una volta preparato l'ambiente, il robot è stato acceso e si è iniziato il processo di setup del sistema realizzato.

4.1 Calibrazione e setup del sistema

Il primo passo riguarda l'avvio di tutti i nodi sviluppati tramite i rispettivi file di lancio, in particolare: *baseline12* per l'acquisizione dei dati dalla telecamera, *calibration* per il caricamento dei parametri di calibrazione e la creazione dei sistemi di riferimento, *map_creator* per la creazione della point cloud, *base_nav* per la creazione della mappa di navigazione e la pianificazione delle traiettorie verso il goal, *cloud_clust* per l'identificazione di oggetti e l'impostazione del goal. Al fine di rendere visibili le elaborazioni dei singoli nodi, è stato lanciato anche il visualizzatore *Rviz* integrato in ROS.

In fig. 4.2 è possibile vedere l'output della prima fase di elaborazione, ovvero la point cloud acquisita dal sistema stereo di telecamere che sarà poi oggetto di successive elaborazioni. Per agevolare la comprensione, è stato inserito nella visualizzazione un piano rappresentante il pavimento sul quale il robot si muove.

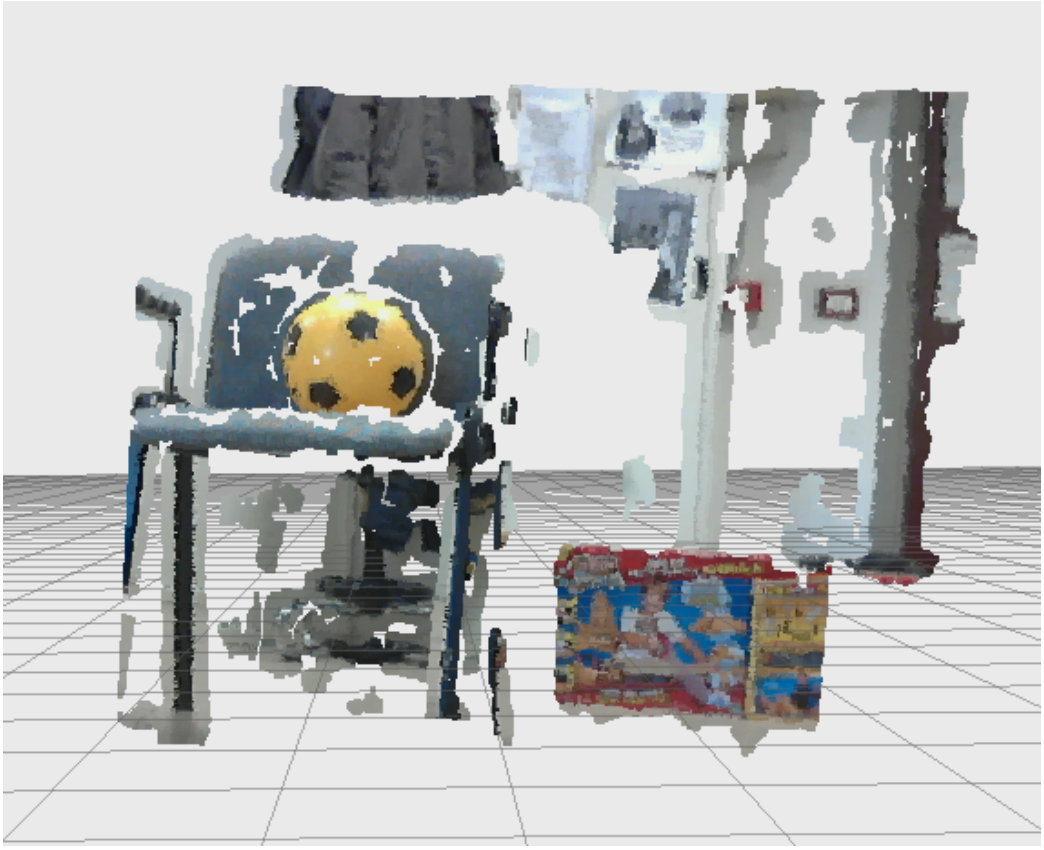


Figura 4.2: Point cloud ottenuta dal nodo di acquisizione di visione stereo.

Successivamente, vengono create le terne dei sistemi di riferimento rispetto alle quali tutte le misure saranno riferite. In Rviz, ciascuna terna è visualizzata con l'asse X in rosso, l'asse Y in verde e l'asse Z in blu.

Come si può vedere, la point cloud acquisita risulta poco densa, presentando numerose zone cieche che non agevolano la navigazione. Infatti, a fronte di una risoluzione delle telecamere di 640 x 480 pixel, i pixel effettivamente acquisiti variano tra gli 80 000 e i 100 000 a fronte di un numero potenziale di pixel pari a 307 000. Queste zone cieche sono causate dal fatto che l'immagine di disparità, da cui viene desunta la nuvola di punti, viene filtrata dei punti con un valore di disparità inferiore ad una certa soglia, al fine di garantirne la robustezza.

Al fine di ridurre il problema di zone cieche, che potrebbe portare a non percepire alcuni ostacoli, si è attuato un processo di accumulo delle ultime N point

cloud, operazione svolta dal nodo *map_creator*. Questo nodo si occupa anche della rimozione degli outlier e della fusione sensoriale della point cloud con i dati del laser scanner. Un'immagine della point cloud risultante è mostrata in fig. 4.3.

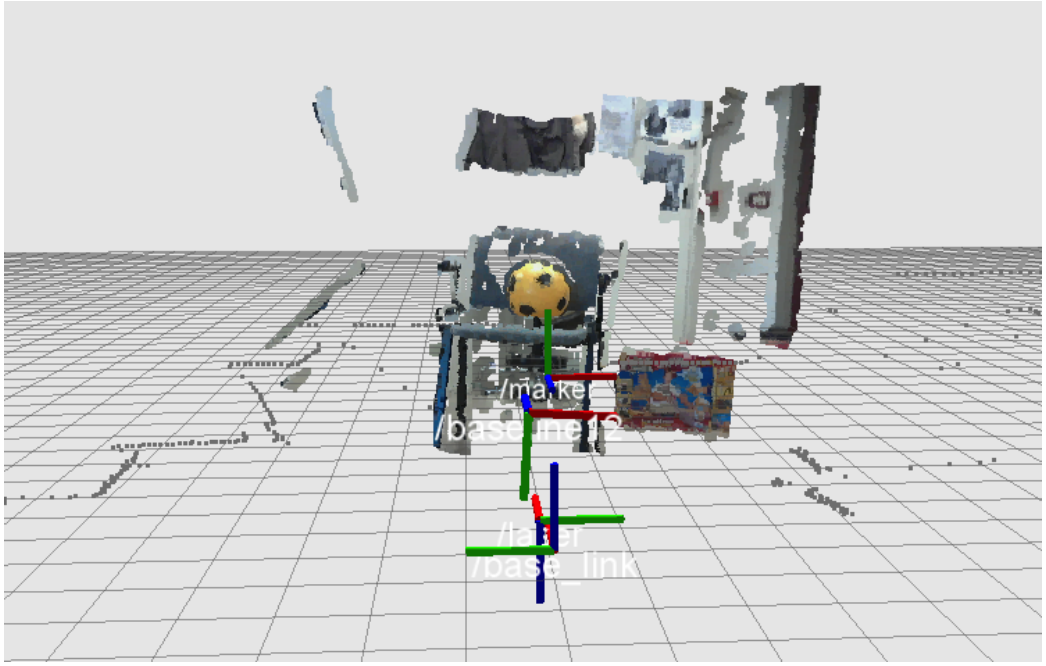


Figura 4.3: Point cloud ottenuta dalla fusione sensoriale e dall'accumulo. E' possibile notare la presenza di punti privi di colore sul piano di scansione, ai lati dell'immagine.

L'avvio del nodo *base_nav* consente di creare, a partire dalla point cloud, la vera e propria mappa di navigazione, mostrata in fig. 4.5, dalla quale è possibile notare la corrispondenza tra i pixel rossi (occupati) e gli ostacoli mostrati nella point cloud. In particolare, si noti come la sedia, presente nella parte centrale dell'immagine, venga proiettata sulla mappa di navigazione anche se posta su un piano diverso dal piano di scansione del laser scanner, grazie al contributo che la telecamera fornisce alla percezione. Questo consente di percepirla come ostacolo nella sua interezza e, quindi, di evitare che il robot si scontri con alcune sue parti.

Durante questo processo di costruzione della mappa, l'ambiente è stato lievemente modificato rimuovendo il pallone giallo presente sulla sedia e spostando più a destra l'oggetto target. Questo ha consentito di verificare il processo di aggiornamento della mappa locale prima di attivare i motori del robot mobile.

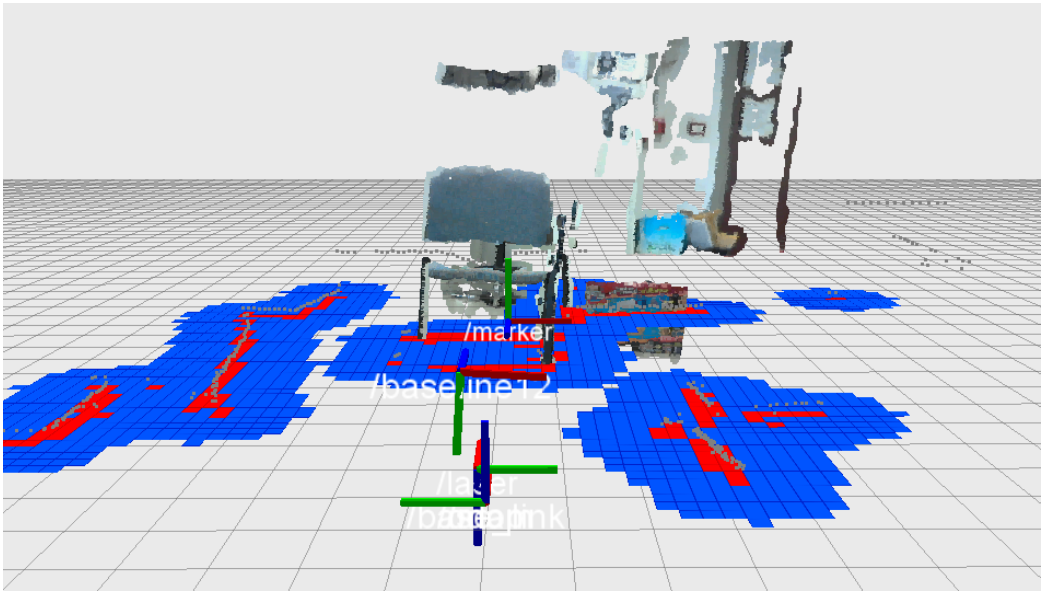


Figura 4.4: La mappa di navigazione creata dal nodo *base_nav*.

La stessa point cloud accumulata al passo precedente viene poi utilizzata dal nodo *cloud_clustering* per l'identificazione degli oggetti sulla scena. In particolare, da file di lancio si sono impostati gli intervalli di dimensioni ammissibili per l'oggetto da cercare, come da tabella 4.2. Inoltre, si è impostata la distanza dal retro dell'oggetto alla quale il nodo dovrà impostare la configurazione goal da raggiungere pari a 1 m.

Dimensione	**Intervallo**
Larghezza	$[0.55 - 0.65]$ m
Altezza	$[0.20 - 0.40]$ m
Profondità	$[0.08 - 0.18]$ m

Tabella 4.2: Intervalli dimensionali dell'oggetto da ricercare

Il nodo individua correttamente numerosi oggetti presenti sulla scena, in particolare quelli sufficientemente distanziati gli uni dagli altri e composti da un numero sufficiente di punti nella point cloud. Tra questi, dopo alcune iterazioni di stabilizzazione e sincronizzazione dei dati, il nodo individua correttamente l'oggetto target, calcola il centroide e provvede ad impostare la configurazione goal

da raggiungere. In fig. 4.5 è possibile vedere l'oggetto che soddisfa i vincoli impostati da file di lancio, racchiuso nel suo bounding box.

4.2 Pianificazione e navigazione verso il goal

L'impostazione della configurazione goal consente al nodo *base_nav* di calcolare la traiettoria per raggiungerla, che viene poi pubblicata in un messaggio ROS sul topic */move_base/TrajectoryPlannerROS/local_plan*. Tale traiettoria è composta da comandi di velocità lineare e angolare da inviare agli attuatori dei motori del robot, operazione effettuata dal nodo *robot_move*.

In fig. 4.5 è possibile vedere in verde la traiettoria calcolata che, come richiesto, passa sufficientemente lontano dagli ostacoli individuati dalla mappa di occupazione ma, allo stesso tempo, riduce al minimo la distanza percorsa.

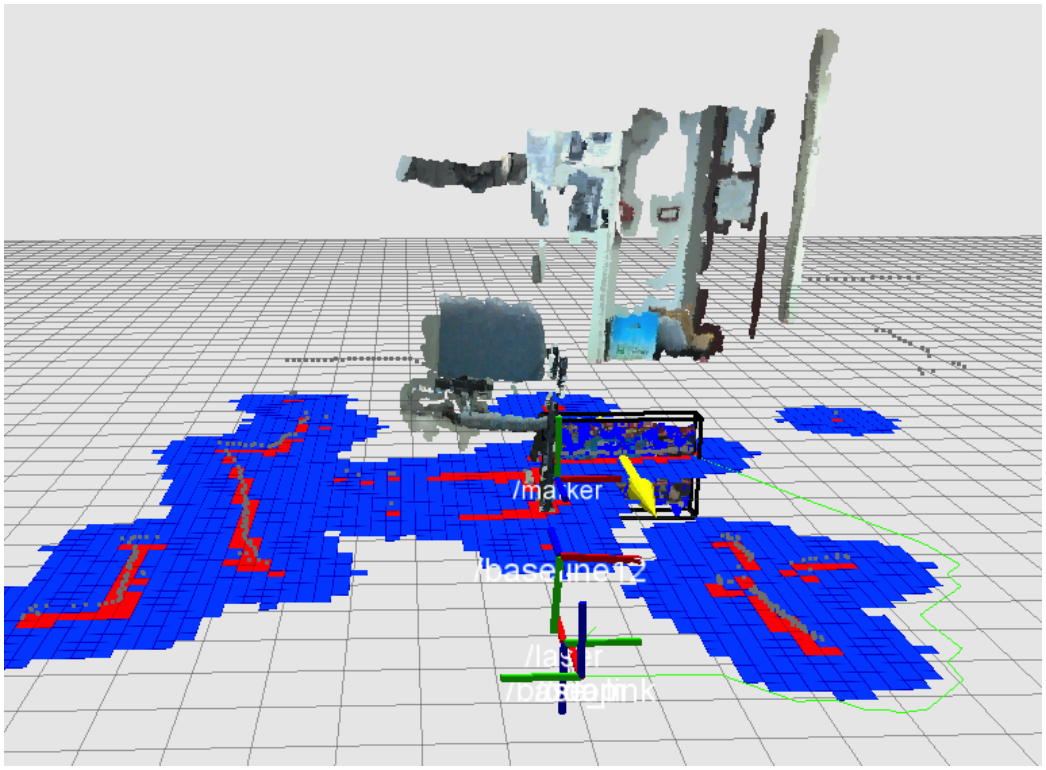


Figura 4.5: Identificazione dell'oggetto, racchiuso in un bounding box nero, impostazione del goal (freccia gialla) e aggiunta della traiettoria calcolata verso la parte posteriore dell'oggetto (in verde).

A questo punto, sono stati attivati i motori del robot tramite la *dashboard*, una semplice interfaccia grafica predisposta dal nodo ROS *p2os* del Pioneer. Il robot ha subito iniziato il suo moto verso il target, procedendo ad una velocità media di circa 0.2 m/s. Sporadicamente, il robot si è arrestato per qualche secondo a causa dell'alto carico computazionale del sistema complessivo, per poi riprendere correttamente il suo tragitto.

Dal punto di vista qualitativo, il sistema si comporta in modo corretto e raggiunge correttamente il punto impostato come *goal*, con una precisione dell'ordine di circa 2 centimetri nella posizione e di decimi di grado nell'orientamento. Piuttosto precisa anche la misurazione delle distanze di oggetti e ostacoli, grazie all'importante contributo del laser scanner e all'accumulo delle point cloud acquisita dalla testa stereo di telecamere.

La stima effettuata solo qualitativamente rende comunque necessari ulteriori test per trarre conclusioni più complete.

4.3 Valutazione dei tempi di elaborazione

Da un punto di vista valutativo, è interessante analizzare la differenza dei tempi medi di elaborazione nelle varie fasi del sistema, che vengono riassunti nella tabella seguente.

Nodo	**Tempo medio**
baseline12	65.71 *ms*
calibration	0.08 *ms*
map_creator	6437.97 *ms*
base_nav	154.32 *ms*
cloud_clustering	7037.94 *ms*

Tabella 4.3: Tempi medi di elaborazione dei nodi del sistema.

Il tempo di elaborazione del nodo *baseline12* è stato stimato utilizzando i dati contenuti nella tesi di laurea di Fabio Oleari [19] in cui è stato sviluppato, dato che si è utilizzato il nodo senza particolari modifiche. La misurazione degli altri tempi è stata condotta calcolando la media dei tempi impiegati per l'esecuzione dei singoli nodi su 5 esecuzioni consecutive.

Il nodo di calibrazione risulta essere il più veloce, in virtù del fatto che si limita a leggere alcuni parametri da file, a creare i sistemi di riferimento e le relative trasformazioni tramite operazioni geometriche e a pubblicarle utilizzando *tf*. I tempi di esecuzione nella sua versione che effettua la calibrazione con la staffa montata, utilizzando *ar_pose*, risultano poco interessanti, in virtù del fatto che la navigazione del robot non può avvenire con la staffa di calibrazione installata per ovvi motivi di ingombro. Pertanto, i tempi misurati si riferiscono alla calibrazione offline che legge i parametri di configurazione da un file precedentemente salvato.

Il nodo *map_creator*, interamente sviluppato nell'ambito di questo progetto di tesi, risulta uno dei più onerosi dal punto di vista computazionale, in quanto la sua pipeline di elaborazione risulta essere piuttosto lunga. Pertanto, risulta utile analizzare più nel dettaglio i tempi di elaborazione delle singole operazioni da esso svolte.

Operazione	**Tempo medio**
Lettura dati della telecamera e trasformazione in terna *robot*	512.63 *ms*
Lettura dati del laser e trasformazione in terna *robot*	475.88 *ms*
Filtraggio valori NaN	517.23 *ms*
Filtraggio outlier	2993.39 *ms*
Filtraggio punti vicini alla telecamera	401.28 *ms*
Voxelizzazione delle point cloud	854.52 *ms*
Calcolo della matrice di trasformazione per l'allineamento	2589.73 *ms*
Allineamento con le point cloud precedenti	394.74 *ms*
Accumulo delle point cloud	1435.25 *ms*
Pubblicazione come messaggio ROS	437.72 *ms*

Tabella 4.4: Tempi medi di elaborazione del nodo *map_creator*

Da questa tabella è possibile notare come le operazioni più onerose siano il filtraggio degli outlier e l'allineamento con le point cloud precedenti (necessario per l'accumulo). Per questo, è stato effettuato lo stesso tipo di prova disattivando, da file di lancio, la rimozione degli outlier e impostando a 1 il numero di point cloud da accumulare. Quest'ultima impostazione consente, indirettamente, di non disattivare la discretizzazione in voxel, il calcolo della matrice di trasformazione e la sua applicazione per l'allineamento delle point cloud.

Durante questa seconda sessione di prove si è vista, come era atteso, una netta riduzione dei tempi di elaborazione. Infatti, mentre prima il tempo di elaborazione

del nodo era, in media, pari a 6437.97 *ms*, con queste funzioni disattivate si scende a soli 869.77 *ms*, con un calo pari all'86.49 %. Nonostante si siano ridotte le operazioni di filtraggio ed elaborazione svolte dal nodo, i risultati della percezione non peggiorano in modo significativo.

Le prestazioni del nodo integrato *base_nav*, che si occupa di creare la mappa di occupazione e pianificare la traiettoria verso il goal, sono accettabili e non compromettono il corretto funzionamento del sistema complessivo.

Diverso, invece, il caso del nodo *cloud_clustering* che si occupa dell'identificazione dell'oggetto target. Sebbene il tempo di elaborazione sia piuttosto elevato, questo non rallenta le prestazioni del sistema complessivo in quanto non è necessario che la ricerca dell'oggetto venga fatta in modo rapido. Infatti, mentre il nodo di creazione della mappa locale è necessario si accorga di ulteriori ostacoli nel più breve tempo possibile, al fine di non compromettere la sicurezza del robot, l'identificazione dell'oggetto target può essere effettuata sporadicamente. Questi vincoli temporali rilassati sono possibili grazie al fatto che gli oggetti sono visibili già da una buona distanza e, pertanto, un ritardo nell'identificazione non compromette l'esito del task complessivo.

Risulta comunque interessante notare che larga parte del tempo impiegato per l'elaborazione di questo nodo è utilizzata per l'estrazione dei cluster dalla point cloud, operazione effettuata tramite un metodo predefinito della Point Cloud Library su cui, pertanto, non si può intervenire con significative ottimizzazioni.

4.4 Distribuzione del carico computazionale

Le prove sono state effettuate su un computer che, sebbene fosse un notebook, era piuttosto performante. Su computer di fascia più bassa o su tablet, la pipeline di elaborazione potrebbe risultare piuttosto pesante. Si è reso quindi necessario studiare una soluzione alternativa per consentire l'esecuzione del sistema anche da computer meno performanti.

Per risolvere questo problema, si è sfruttata la possibilità fornita da ROS di distribuire il carico di lavoro, dovuto all'esecuzione dei diversi nodi del sistema, su più elaboratori connessi alla rete locale. Il funzionamento è piuttosto semplice,

basandosi su un'architettura di tipo client/server composta da un client, posto a bordo del robot mobile, e uno o più server remoti connessi alla rete locale.

Il client, al quale sono connessi tutti i dispositivi di acquisizione dei dati sensoriali, si preoccupa di quelle operazioni poco onerose e di quelle che non è possibile distribuire, in virtù del fatto che necessitano del collegamento fisico con uno o più dispositivi. Nel caso del sistema sviluppato, il client si occuperà dell'acquisizione dei dati della testa stereo e del laser scanner, del calcolo dell'immagine di disparità e dell'applicazione dei comandi di velocità al robot mobile. I dati acquisiti saranno pubblicati come messaggi ROS per essere poi letti ed elaborati dal server.

Il server (o l'insieme dei server) ha come unico compito l'elaborazione di quei nodi molto esigenti di risorse hardware e, pertanto, occorre sia una macchina piuttosto performante. Il server riceve i messaggi ROS contenenti i dati di input acquisiti dai sensori dal client, avvia i nodi corrispondenti per elaborarli e poi pubblica un messaggio ROS contenente l'output. Particolarmente rilevante è l'utilizzo del server per l'esecuzione dei nodi intermedi alla pipeline di elaborazione, consentendo quindi di risparmiare la pubblicazione di risultati intermedi, che potrebbero sovraccaricare inutilmente la rete locale.

L'architettura di rete utilizzata per le prove è mostrata in fig. 4.6.

Ogni nodo del sistema sviluppato è stato predisposto per essere lanciato ed elaborato da un computer remoto, specificando da file di lancio il nome dell'host (server) che andrà ad eseguire il nodo.

Le prove condotte in laboratorio in tal senso hanno visto l'utilizzo, come server, di una workstation di recente acquisizione (*fenena*), dotata di processore Intel Core i7 Quad-Core 3,6 GHz, 8 GB di RAM e ROS Fuerte. Come client, invece, è stato utilizzato un portatile Acer Aspire 1810TZ, dotato di processore Intel Core 2 Duo 1,3 GHz, 2 GB di RAM e ROS Fuerte.

I risultati sono stati soddisfacenti: i tempi di elaborazione si sono ridotti di circa il 30 % rispetto all'elaborazione considerata nel caso sopra descritto. Tuttavia, in virtù della grossa mole di dati acquisiti dalle telecamere e successivamente scambiati, attraverso la rete locale, tra client e workstation, la distribuzione del calcolo computazionale richiede una rete cablata, preferibilmente Gigabit Ethernet, al fine di non saturare la banda a disposizione e rendere inutilizzabile il sistema. Con lo sviluppo delle nuove tecnologie senza fili, in particolare la nuova connes-

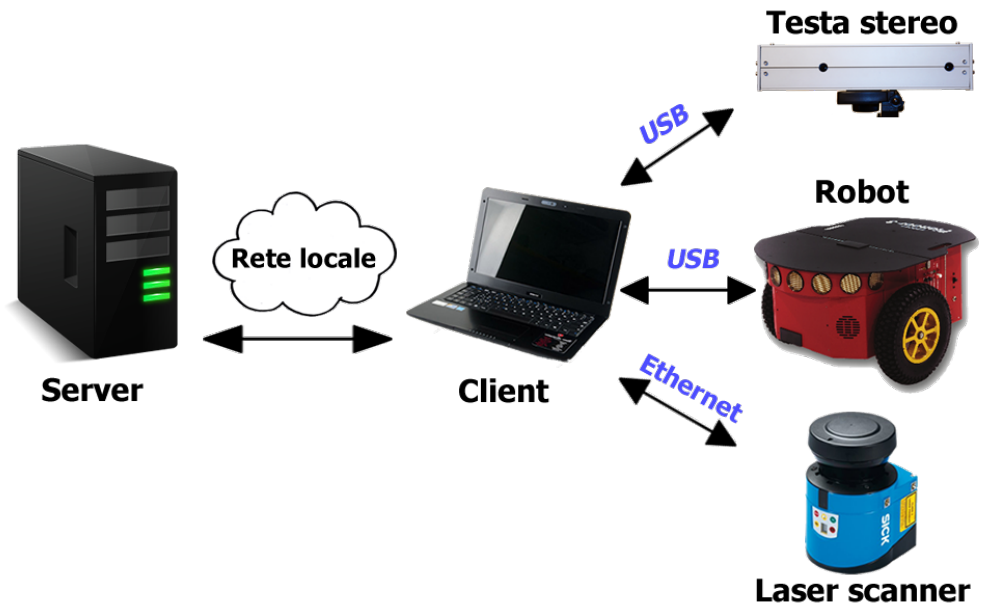


Figura 4.6: Architettura di rete per la distribuzione dell'elaborazione.

sione Wi-Fi 802.11n a 150 Mbps, sarà però presto possibile utilizzare anche la rete senza fili e agevolare maggiormente la suddivisione del carico, magari tra più nodi server, mantenendo un host poco potente a bordo del robot.

Conclusioni e sviluppi futuri

In questo lavoro di tesi è stato sviluppato un sistema di percezione tridimensionale, utile alla navigazione di robot mobili da un lato e all'individuazione di oggetti dall'altro. Il primo problema che ci si è prefissi di risolvere è quello di rappresentare gli ostacoli in modo tridimensionale sfruttando il sistema di visione stereo. Con sensori planari, infatti, la presenza di oggetti sporgenti al di fuori del piano di scansione, quali ad esempio sedie e tavoli, rende non pienamente sicura la navigazione del robot mobile nell'ambiente.

Il problema è stato risolto integrando il dato sensoriale di una testa stereo di telecamere, al fine di aumentare la percezione frontale. L'integrazione di un sensore di visione nel sistema robot rende necessario l'utilizzo di opportune procedure di calibrazione che aumentano la complessità del sistema. L'aumento della complessità computazionale ha reso necessaria la razionalizzazione delle risorse di calcolo e l'utilizzo di funzioni correttive solo ove assolutamente necessario.

Tuttavia, la visione stereoscopica genera una nuvola di punti a bassa densità e su un ridotto campo visivo, limitando il contributo che la testa stereo fornisce al sistema di percezione del robot. Per risolvere questo problema si è provveduto ad accumulare più point cloud, derivanti dalle ultime acquisizioni della testa stereo. L'accumulo ha reso necessario un'operazione di *registration*, ovvero di allineamento di più viste della scena tramite l'applicazione di un'opportuna matrice di trasformazione geometrica.

La successiva integrazione nel sistema di strumenti come il nodo di costruzione della mappa locale con memoria a breve termine consente di fornire al robot le informazioni di occupazione spaziale necessarie per la navigazione in sicurezza nell'ambiente. La mappa locale viene ottenuta tramite una proiezione planare della point cloud risultante dal passo precedente. La successiva pianificazione del

moto verso una configurazione goal verrà effettuata utilizzando questa mappa.

Un'altra funzionalità molto importante nell'ambito della robotica è quella di identificare correttamente gli oggetti in un ambiente, al fine di interagire con essi (afferrarli, raggiungerli, ecc.). Nell'ambito di questa tesi si è affrontato il problema dell'individuazione e del raggiungimento di un oggetto target da parte del robot mobile, tramite pianificazione della traiettoria verso una configurazione goal.

La funzione di raggiungimento dell'oggetto da parte del robot, in virtù del buon funzionamento degli altri nodi e della loro corretta integrazione, si limita all'estrazione degli oggetti con tecniche elementari e all'impostazione di un punto da raggiungere dietro all'oggetto target. La scelta dell'oggetto da raggiungere viene attuata con metodi di selezione sulla point cloud segmentata che tengano in considerazione criteri di attenzione, ovvero privilegiando gli oggetti più vicini al robot che soddisfino determinate condizioni dimensionali. Una possibile integrazione della funzionalità offerta potrebbe essere una vera e propria catalogazione degli oggetti presenti nella scena, al fine di attuare metodi di selezione più vicini alla semantica dell'oggetto piuttosto che alle sue caratteristiche strutturali.

Si è quindi realizzato e fornito all'utente un sistema completo che aumenta l'autonomia di un robot mobile e apre la strada all'integrazione di ulteriori funzionalità, senza doversi focalizzare sugli aspetti di basso livello, come evitare gli ostacoli o acquisire dati sensoriali, già trattati nel progetto sviluppato. Alcuni esperimenti di navigazione ed osservazione di un oggetto da molteplici punti di vista sono stati condotti con successo, validando il corretto funzionamento del sistema complessivo.

Una prima miglioria futura potrà essere quella di ridurre ulteriormente la complessità computazionale, intervenendo su quei metodi che si sono dimostrati troppo onerosi, attraverso una reingegnerizzazione o un cambio di approccio.

Interessante risulta essere anche il completamento dell'accumulo di diverse viste per la costruzione di una point cloud tridimensionale dell'oggetto che ne fornisca una vista più completa, con possibilità di identificarne anche i singoli particolari. L'obiettivo finale di questa integrazione potrebbe essere la costruzione di una mappa 3D navigabile di un ambiente, in cui si possano visualizzare tutti gli oggetti presenti già etichettati e classificati.

La robotica mobile è un ambito molto studiato negli ultimi tempi e questa tesi ha voluto entrarvi in modo deciso per fornire il proprio contributo all'evolvere della ricerca in questo settore, ben sapendo che il lavoro da fare è ancora tanto.

Bibliografia

[1] H. Moravec. Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids. *IEEE International Conference on Robotics and Automation (ICRA)*, September 1996.

[2] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[3] Open Perception Foundation. Sito ufficiale della Point Cloud Library. http://www.pointcloud.org.

[4] M. Herbert, C. Caillas, E. Krotkov, I. S. Kweon and T. Kanade. Terrain Mapping for a Roving Planetary Explorer. *IEEE International Conference on Robotics and Automation (ICRA)*, May 1989.

[5] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, February 2013.

[6] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey and K. Konolige. The Office Marathon: Robust Navigation in an Indoor Office Environment. *IEEE International Conference on Robotics and Automation (ICRA)*, May 2010.

[7] R. Kummerle, D. Hahnel, D. Dolgov, S. Thrun and W. Burgard. Autonomous Driving in a Multi-level Parking Structure. *IEEE International Conference on Robotics and Automation (ICRA)*, May 2009.

[8] K. Steinbach, J. Kuffner, T. Asfour and R. Dillmann. Efficient Collision and Self-Collision Detection for Humanoids Based on Sphere Tree Hierarchies. *Proceedings of the IEEE International Conference on Humanoid Robots (Humanoids)*, September 2006.

[9] A. Hornung, M. Phillips, E. Gil Jones, M. Bennewitz, M. Likhachev and S. Chitta. Navigation in Three-Dimensional Cluttered Environments for Mobile Manipulation. *IEEE International Conference on Robotics and Automation (ICRA)*, May 2012.

[10] P. Gvozdjak and L. Ze-Nian. From Nomad to Explorer: Active Object Recognition on Mobile Robots. *IEEE International Conference on Systems, Man, and Cybernetics*, October 1995.

[11] D. H. Ballard. Generalizing the Hough Transform to Detect Arbitrary Shapes. *Pattern Recognition*, 1981.

[12] P. Jensfelt S. Ekvall and D. Kragic. Integrating Active Mobile Robot Object Recognition and SLAM in Natural Environments. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, October 2006.

[13] S. Ekvall and D. Kragic. Receptive Field Cooccurrence Histograms for Object Detection. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, August 2005.

[14] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, November 2004.

[15] P. E. Forssén, D. Meger, K. Lai, S. Helmer, J. J. Little and D. G. Lowe. Informed Visual Search: Combining Attention and Object Recognition. *IEEE International Conference on Robotics and Automation (ICRA)*, May 2008.

[16] J. Westell and P. Saeedi. 3d object recognition via multi-view inspection in unknown environments. *11th International Conference on Control Automation Robotics Vision (ICARCV)*, December 2010.

[17] M. Bjorkman L. Nalpantidis and D. Kragic. YES - YEt Another Object Segmentation: Exploiting Camera Movement. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, October 2012.

[18] A. Escande T. Foissotte, O. Stasse and A. Kheddar. A Next-Best-View Algorithm for Autonomous 3D Object Modeling by a Humanoid Robot. *8th IEEE-RAS International Conference on Humanoid Robots*, December 2008.

[19] F. Oleari. Riconoscimento di oggetti 3D tramite visione stereo e allineamento di caratteristiche FPFH. *Tesi di laurea in Ingegneria Informatica, Università degli Studi di Parma*, December 2012.

[20] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. *2nd IEEE and ACM International Workshop on Augmented Reality*, October 1999.

[21] Wikipedia. Bayer filter. http://en.wikipedia.org/wiki/Bayer_filter, 2012.

[22] D. Fox, W. Burgard and S. Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics Automation Magazine*, March 1997.