UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Integrazione tra robot Pepper e Robot Operating System: analisi del problema di localizzazione e validazione delle performance

Integration between Pepper robot and Robot Operating System: localization problem analysis and performance validation

Relatore:

PROF. ALESSANDRO FREDDI

Tesi di Laurea di:

CHIARA DI FELICE

Correlatore:

DOTT. DANIELE PROIETTI PAGNOTTA

Anno Accademico 2020 – 2021

Sommario

Questa tesi presenta i principali algoritmi di localizzazione esistenti in letteratura che vengono applicati al caso di studio del robot Pepper, cioè, viene utilizzata la sensoristica a bordo per effetturare mappatura dell'ambiente e localizzazione. Pepper è un robot da compagnia/sociale progettato per consentire l'interazione cognitiva e fisica con gli esseri umani. La localizzazione è una delle competenze fondamentali richieste da un robot autonomo poiché la conoscenza della posizione del robot è un precursore essenziale per prendere decisioni sulle azioni future. La navigazione permette al robot di navigare e operare autonomamente e in sicurezza nell'ambiente di lavoro ed è la funzione principale di robot mobili autonomi. Pertanto mappatura, localizzazione e navigazioni sono problemi strettamente correlati tra loro e continuano oggi giorno ad essere aree molto attive di ricerca.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

I robot di servizio rappresentano oggigiorno una tecnologia in forte sviluppo, per via dei loro campi di impiego, quali: cura degli anziani [1, 2], sorveglianza strutture pubbliche, musei ed attrazione turistiche [3], esplorazioni ambientali [4] e anche il settore educativo [5].

Che cosa sono, tuttavia, i robot di servizio? Ad oggi, varie istituzioni ed organizzazioni hanno provato a dare una definizione tangibile. La International Federation of Robotics (IFR) ha sviluppato una definizione nel 1997, secondo la quale, "I robot di servizio sono robot che operano in modo parzialmente o totalmente autonomo, per svolgere servizi atti al benessere dell'uomo e delle macchine. Sono di tipo mobile o manipolatori o una combinazione di entrambi". Questa affermazione si basa sulla definizione suggerita dal Fraunhofer Institute for Manufacting Engineering and Automation (IPA) nel 1994: "un robot di servizio è un dispositivo mobile liberamente programmabile che esegue servizi parzialmente o completamente automatici. I servizi sono compiti che non servono alla fabbricazione industriale diretta di beni, ma alla prestazione di servizi per l'uomo e le attrezzature ". Certamente nessuna delle due definizioni è generale, né comprende ogni aspetto. Una cosa è chiara, però: le aree di applicazione dei robot di servizio non sono facilmente definibili come quelle dei robot industriali. Ci sono due aree di applicazione principali nella robotica di base: una è la navigazione e l'altra è la manipolazione. La navigazione è la funzione principale di robot mobili autonomi: si costruisce un modello dell'ambiente attraverso l'osservazione dello stesso e con tale acquisizione si pianifica di muoversi da un punto di partenza ad uno di arrivo. La manipolazione, invece, è la funzione principale dei robot manipolatori e consiste nel saper afferrare, spostare, e rilasciare oggetti all'interno di uno spazio di lavoro. Numerosi manipolatori vengono utilizzati per l'assemblaggio di prodotti nelle industrie [6]. La robotica industriale ha poi sviluppato componenti chiave per costruire robot più simili a

quelli umani, con sensori e motori. La differenza più significativa tra robot industriali e quelli di servizio è 'l'interazione', infatti, negli ambienti quotidiani i robot incontrano gli umani e devono interagire con loro prima di svolgere il compito; piuttosto sarà proprio l'interazione il compito principale. L'interazione uomo-robot (Human Robot Interaction HRI) è attualmente un'area di ricerca e progettazione molto ampia e diversificata. Essa viene classificata sostanzialmente in quattro aree: supervisione dei robot nell'esecuzione di compiti di routine (includono la gestione delle linee di assemblaggio di produzione e telerobot), controllo remoto di veicoli per attività non di routine in ambienti ostili, veicoli automatizzati, e, infine, interazione sociale, compresi dispositivi per fornire assistenza [7]. Esistono poi anche i robot di compagnia che stanno prendendo sempre più piede. Un compagno robotico autonomo potrebbe essere visto come un tipo speciale di robot di servizio che è specificamente progettato per l'uso personale a casa. Sempre più spesso i robot sono destinati a impegnarsi nell'interazione sociale-umana, i compagni robotici in casa dovrebbero idealmente essere in grado di eseguire una vasta gamma di compiti tra cui funzioni educative, sicurezza domestica, compiti di agenda, intrattenimento e servizi di consegna dei messaggi, ecc [8].

1.1 Stato dell'arte degli algoritmi di localizzazione

Negli ultimi anni, la gamma di applicazioni dei robot mobili è aumentata significativamente e si possono trovare in diversi ambienti, come nelle case, nell'industria e nell'istruzione, dove sono in grado di svolgere una varietà di compiti. I miglioramenti effettuati sia nella percezione che nel calcolo hanno avuto un contributo importante per aumentare la gamma di ambienti in cui i robot mobili possono essere utilizzati. Per essere pienamente funzionale, un robot mobile deve essere in grado di navigare in modo sicuro in un ambiente sconosciuto e contemporaneamente svolgere il compito per cui è stato progettato. A tal fine, il robot deve essere in grado di costruire un modello o una mappa di questo ambiente precedentemente sconosciuto, di stimare la sua posizione attuale e l'orientamento facendo uso di questo modello, e di navigare verso i punti di destinazione. Mappatura, localizzazione [9] e navigazione [10] sono tre problemi classici della robotica mobile che hanno ricevuto molta attenzione in letteratura e continuano ad essere aree di ricerca molto attive attualmente. Trovare una soluzione robusta a questi tre problemi chiave è fondamentale per aumentare l'autonomia e l'adattabilità dei robot mobili a diverse circostanze e per espandere definitivamente la loro

gamma di applicazioni. Per affrontare i problemi di mappatura, localizzazione e navigazione, è necessario che il robot abbia alcune informazioni rilevanti sull'ambiente in cui si muove, che è a priori sconosciuto nella maggior parte delle applicazioni. I robot possono essere dotati di diversi sistemi sensoriali che permettono loro di estrarre le informazioni necessarie dall'ambiente per poter svolgere i loro compiti in modo autonomo. Così, dai primi lavori sulla robotica mobile, i sistemi di controllo hanno fatto uso delle informazioni raccolte dall'ambiente, in misura maggiore o minore. I metodi utilizzati per elaborare queste informazioni si sono evoluti man mano che sono stati sviluppati nuovi algoritmi e sono aumentate le capacità dei sistemi di percezione e di calcolo. Di conseguenza, sono emersi approcci molto diversi, considerando diversi tipi di informazioni sensoriali e tecniche di elaborazione.
Inizialmente si cercava di modellare la geometria dell'ambiente con precisione metrica, a partire da informazioni visive, e di organizzare le informazioni attraverso modelli CAD (Computer Assisted Design). Col progredire della ricerca, sono stati definiti diversi approcci che hanno lasciato il posto a modelli più semplici che rappresentano l'ambiente attraverso griglie di occupazione, mappe topologiche o anche sequenze di immagini. Tradizionalmente, i problemi di costruzione delle mappe e localizzazione sono stati affrontati utilizzando tre approcci diversi:

- Costruzione della mappa e successiva localizzazione: in questi approcci, prima il robot attraversa l'ambiente da mappare (di solito in modo teleoperato) e raccoglie alcune informazioni utili da una varietà di posizioni. Queste informazioni vengono poi elaborate per costruire la mappa. Una volta che la mappa è disponibile, il robot cattura informazioni dalla sua posizione attuale e, confrontando queste informazioni con la mappa, viene stimata la sua posa attuale (posizione e orientamento).

- Costruzione e aggiornamento continuo della mappa: in questo approccio, il robot è in grado di esplorare autonomamente l'ambiente per mappare e costruire o aggiornare un modello mentre è in movimento. Gli algoritmi SLAM (Simultaneous Localization And Mapping) rientrano in questo approccio.

- Sistemi di navigazione senza mappa: il robot naviga attraverso l'ambiente applicando alcune tecniche di analisi alle ultime scene catturate, come il flusso ottico, o attraverso memorie visive precedentemente registrate che contengono sequenze di immagini e azioni associate, ma nessuna relazione tra le immagini. Questo tipo di navigazione è associato, fondamentalmente, a comportamenti reattivi.

Per risolvere il problema della localizzazione è necessario avere un modello dell'ambiente costruito interamente a priori, oppure solo parzialmente e poi aggiornato di continuo, in modo tale che il robot possa stimare la sua posizione confrontando le sue informazioni sensoriali con le informazioni catturate nel modello. In generale, a seconda di come queste informazioni sono organizzate, questi modelli possono essere classificati in tre categorie: *metrici*, *topologici* o *ibridi*. Gli approcci metrici cercano di creare un modello con precisione geometrica, includendo alcune caratteristiche dell'ambiente rispetto a un sistema di riferimento. Questi modelli sono creati usando caratteristiche locali estratte da immagini e permettono di stimare metricamente la posizione del robot, fino a un errore specifico. Al contrario, gli approcci topologici cercano di creare modelli compatti che includono informazioni da diverse localizzazioni caratteristiche e le relazioni di connettività tra di esse. Sia le caratteristiche locali che l'aspetto globale possono essere usati per creare tali modelli. Essi permettono una localizzazione approssimativa del robot con un costo computazionale ragionevole. Infine, le mappe ibride combinano informazioni metriche e topologiche per cercare di sfruttare i vantaggi di entrambi i metodi. Le informazioni sono organizzate in diversi strati, con informazioni topologiche che permettono di effettuare una stima iniziale approssimativa e informazioni metriche per raffinare questa stima quando necessario [11].

La localizzazione basata su sensori è stata riconosciuta come un problema chiave nella robotica mobile [12, 13]. La localizzazione è una versione della stima di stato temporale on-line, in cui un robot mobile cerca di stimare la sua posizione rispetto ad un frame globale di coordinate. Il problema della localizzazione si presenta in due aspetti: localizzazione *globale* e *locale*. Il secondo è di gran lunga il problema più studiato; qui un robot conosce la sua posizione iniziale e deve "solo" adattarsi a piccoli errori nella sua odometria mentre si muove. Il problema della localizzazione globale coinvolge un robot a cui non viene comunicata la sua posizione iniziale; quindi, deve risolvere un problema di localizzazione molto più difficile, quello di stimare la sua posizione da zero (a volte ci si riferisce a questo come al problema del *robot sequestrato* come nel lavoro di *Engelson* [14]). La capacità di localizzarsi, sia a livello locale che globale, ha giocato un ruolo importante in una serie di applicazioni di robot mobili [15, 16]. Mentre la maggior parte dei primi lavori si è concentrata sul problema del *tracking* (ovvero il problema di monitorare la posizione del robot, conoscendo la sua posizione iniziale e adattandosi a piccoli errori nella sua odometria), diversi ricercatori hanno sviluppato quella che ora è una famiglia di approcci di grande successo in grado di risolvere entrambi i problemi di localizzazione: localizzazione di Markov [15, 17, 18]. L'idea centrale della localizzazione di Markov è di rappresentare la posizione presunta del robot (denominata *robot belief*) con una distribuzione di pro-

babilità delle posizioni possibili, e usare la regola di Bayes e l'operazione di convoluzione per aggiornarla ogni volta che il robot percepisce o si muove. L'idea della stima probabilistica dello stato risale ai filtri Kalman [19], che usano gaussiane multi-variate per rappresentare della posizione presunta del robot. Esso assume che la densità a posteriori in ogni istante di tempo sia gaussiana e quindi parametrizzabile da una media ed una covarianza, introducendo inevitabilmente alcune assunzioni stringenti. I rumori agenti sul sistema e sulle misure devono essere descritti da una distribuzione gaussiana, il modello che descrive il sistema deve essere noto e lineare e il modello dei sensori deve essere una funzione lineare nota dello stato. Ne deriva quindi un filtro ottimo nel caso gaussiano lineare, in quanto è in grado di minimizzare la covarianza d'errore stimata.

La localizzazione di Markov, invece, impiega rappresentazioni discrete, e multi-modali per rappresentare la posizione del robot, quindi può risolvere il problema della localizzazione globale. Infatti, si utilizzano distribuzioni multi-modali e ipotesi multiple proprio per evitare l'unicità della configurazione di partenza e per garantire un recupero dello stato del sistema anche in presenza di errori considerevoli. La capacità di approssimare distribuzioni multi-modali e di includere modelli non lineari per le misure e per il movimento, rende i filtri particolarmente robusti. A causa della natura reale e multidimensionale degli spazi di stato cinematici, questi approcci possono solo approssimare la robot belief, e un'approssimazione accurata di solito richiede quantità proibitive di calcolo e di memoria. In particolare, sono stati sviluppati metodi basati su griglia (*grid-based localization*) che approssimano lo spazio di stato cinematico con funzioni costanti a grana fine [20]. Per ambienti di dimensioni ragionevoli, questi approcci richiedono una memoria superiore a 100 MB un potenza di calcolo elevata. All'altro estremo, vari ricercatori hanno fatto ricorso a rappresentazioni topologiche a grana grossa, la cui granularità è spesso un ordine di grandezza inferiore a quella dell'approccio basato sulla griglia. Quando è necessaria un'alta resa, come per esempio, nel lavoro di *Fox et al.* [21], che usa la localizzazione per evitare collisioni con ostacoli statici che non possono essere rilevati dai sensori, tali approcci sono inapplicabili.

Viene poi presentata la localizzazione Monte Carlo (MCL). I metodi Monte Carlo sono stati introdotti negli anni Settanta [22], e riscoperti nella letteratura di target-tracking [23], statistica [24] e di computer vision [25], e sono stati anche applicati a reti probabilistiche dinamiche [26]. MCL usa tecniche di campionamento veloce per rappresentare la posizione presunta del robot. Quando il robot si muove o percepisce, viene applicato l'*importance resampling* [27] per stimare la distribuzione a posteriori. Uno schema di campionamento adattivo [28], che determina il numero di campioni al volo, è

impiegato per bilanciare calcolo e precisione. Come risultato, MCL usa molti campioni durante la localizzazione globale quando sono più necessari, mentre la dimensione del set di campioni è piccola durante quella locale, quando la posizione del robot è approssimativamente nota. Usando una rappresentazione basata sul campionamento, MCL ha diversi vantaggi chiave rispetto ai lavori precedenti nel campo:

- in contrasto con le tecniche esistenti basate sul filtraggio alla Kalman, è in grado di rappresentare distribuzioni multi-modali e quindi può localizzare globalmente un robot;
- riduce drasticamente la quantità di memoria richiesta rispetto alla localizzazione di Markov basata su griglia e può integrare le misure ad una frequenza considerevolmente più alta;
- è più accurata della localizzazione di Markov con una dimensione fissa della cella, poiché lo stato rappresentato nei campioni non è discretizzato;
- è molto più facile da implementare [29] .

L'approccio SLAM, termine coniato per la prima volta da Leonard e Durrant-Whyte [30], indica il processo per cui il robot è in grado contemporaneamente di costruire una mappa dell'ambiente e di localizzarsi in essa. L'ultimo decennio ha visto progressi rapidi ed entusiasmanti nella soluzione del problema SLAM insieme a molte interessanti implementazioni di tali metodi. La maggior parte del lavoro si è concentrata sul miglioramento dell'efficienza computazionale, garantendo allo stesso tempo stime coerenti e accurate per la mappa e la posa del veicolo. Tuttavia, c'è stata anche molta ricerca su questioni come la non linearità, l'associazione dei dati e la caratterizzazione dei punti di riferimento, che sono tutti vitali per ottenere un'implementazione SLAM pratica e robusta [31].

Un certo numero di approcci sono stati proposti per affrontare sia il problema SLAM che problemi di navigazione più semplificati in cui sono disponibili informazioni aggiuntive sulla mappa o sulla posizione del veicolo [32]. Notevoli sono i lavori di *Thrun et al.* [33] e *Yamauchi et al.* [34], che usano un approccio bayesiano alla costruzione delle mappe che non assumono distribuzioni di probabilità gaussiane come richiesto dal filtro di Kalman. Questa tecnica, pur essendo molto efficace per la localizzazione rispetto alle mappe, non si presta a fornire una soluzione incrementale allo SLAM in cui una mappa viene gradualmente costruita man mano che si ricevono informazioni dai sensori. In [34] *Yamauchi et al.* usano un approccio a griglia di prova

che richiede che l'ambiente sia decomposto in un certo numero di celle. Il lavoro iniziale di *Smith et al.* [35] e *Durrant-Whyte* [36] ha stabilito una base statistica per descrivere le relazioni tra i punti di riferimento e manipolare l'incertezza geometrica. Un elemento chiave di questo lavoro era dimostrare che ci deve essere un alto grado di correlazione tra le stime della posizione di diversi punti di riferimento in una mappa e che queste correlazioni sarebbero cresciute fino all'unità dopo osservazioni successive. Allo stesso tempo *Ayache & Faugeras* [37] e *Chatila & Laumond* [38] stavano intraprendendo i primi lavori sulla navigazione visiva di robot mobili utilizzando algoritmi del tipo filtro di Kalman. Questi due filoni di ricerca avevano molto in comune e sono sfociati poco dopo nell'articolo chiave di *Smith, Self e Cheeseman* [19]. È stato poi mostrato che, mentre un robot mobile si muove attraverso un ambiente sconosciuto prendendo osservazioni relative di punti di riferimento, le stime di questi punti di riferimento sono tutte necessariamente correlate tra loro a causa dell'errore comune nella posizione stimata del veicolo. L'articolo è stato seguito da una serie di lavori correlati che sviluppano una serie di aspetti del problema SLAM essenziale. La conclusione principale di questo lavoro era duplice. In primo luogo, tenere conto delle correlazioni tra i punti di riferimento in una mappa è importante se si vuole mantenere la coerenza del filtro. In secondo luogo, una soluzione SLAM completa richiede che un vettore di stato, che consiste in tutti gli stati del modello del veicolo e tutti gli stati di ogni punto di riferimento nella mappa, deve essere mantenuto e aggiornato dopo ogni osservazione se è richiesta una soluzione completa al problema SLAM. La conseguenza di ciò in qualsiasi applicazione reale è che il filtro di Kalman ha bisogno di impiegare un enorme vettore di stato (dell'ordine del numero di punti di riferimento mantenuti nella mappa) ed è in generale, computazionalmente intrattabile. Fondamentalmente, questo lavoro non ha esaminato le proprietà di convergenza della mappa o il suo comportamento allo stato stazionario. Infatti, è stato ampiamente assunto all'epoca che gli errori della mappa stimata non convergessero e che invece seguissero un comportamento casuale con una crescita illimitata dell'errore. Data la complessità computazionale del problema SLAM e senza la conoscenza del comportamento di convergenza della mappa, sono state proposte una serie di approssimazioni alla soluzione SLAM completa che presupponevano che le correlazioni tra i punti di riferimento potessero essere minimizzate o eliminate riducendo così il filtro completo a una serie di filtri disaccoppiati da punto di riferimento a veicolo, come nei lavori di *Renken* [39] e *Leonard e Durrant-Whyte* [30].

Recentemente, il metodo basato su grafi è diventato un problema importante [40]. Questo è un metodo per creare un grafo definendo la relazione di posizione tra il robot e le sue caratteristiche come vincoli e riducendo al mini-

mo l'errore accumulato nell'intero grafo. In precedenza, c'era il problema che le prestazioni in tempo reale erano scarse a causa di una grande quantità di calcoli, ma recentemente sono stati proposti vari metodi per superare questo problema. Quando si esegue lo SLAM in una vasta area, gli errori possono accumularsi nella mappa creata e il problema della mancata corrispondenza della mappa può verificarsi quando si torna nell'area precedentemente visitata. In un tale ambiente, è necessario eseguire la chiusura del ciclo. La stabilità della tecnologia SLAM è notevolmente migliorata e sono stati proposti anche vari metodi che utilizzano sensori a basso costo per garantire prestazioni in tempo reale nel sistema. Diversi gruppi di ricerca e ricercatori hanno lavorato e stanno attualmente lavorando sullo SLAM, e i sensori più comunemente usati possono essere categorizzati in sistemi basati su laser, sonar e visione. Altre fonti sensoriali sono utilizzate per percepire meglio le informazioni sullo stato del robot e il mondo esterno, come le bussole, la tecnologia a infrarossi e il Global Positioning System (GPS). Tuttavia, tutti questi sensori sono affetti da errori, spesso indicati come rumore di misurazione, e hanno anche diversi limiti di portata che rendono necessario navigare attraverso l'ambiente (ad esempio, la luce e il suono non possono penetrare i muri). Dal punto di vista della misura, i sistemi di misurazione laser sono i sensori più accurati. Comunemente funzionano secondo il principio del tempo di volo, inviando un impulso laser in un fascio stretto verso l'oggetto e misurando il tempo impiegato dall'impulso per essere riflesso dal bersaglio e restituito al mittente. I sistemi basati sul sonar sono veloci e forniscono misurazioni e capacità di riconoscimento con quantità di informazioni simili alla visione, ma con la mancanza di dati di aspetto e un alto rumore dovuto all'incertezza dei parametri ambientali come umidità e temperatura dell'aria. D'altra parte, i sistemi di visione hanno un lungo raggio e un'alta risoluzione, ma il costo di calcolo è considerevolmente alto e le buone caratteristiche visive sono più difficili da estrarre e abbinare. La visione viene usata per stimare la struttura 3D, la posizione delle caratteristiche e la posa del robot, per esempio per mezzo di coppie di telecamere stereo o telecamere monoculari in movimento.

La costruzione di mappe robotiche può essere fatta risalire a 25 anni fa, e dagli anni '90 gli approcci probabilistici (cioè i filtri di Kalman (KF), i filtri di particelle (PF) e la massimizzazione delle aspettative (EM)) sono diventati dominanti nello SLAM. Le tre tecniche sono derivazioni matematiche della regola ricorsiva di Bayes. La ragione principale di questa popolarità delle tecniche probabilistiche è il fatto che la mappatura dei robot è caratterizzata da incertezza e rumore dei sensori, e gli algoritmi probabilistici affrontano il problema modellando esplicitamente le diverse fonti di rumore e i loro effetti sulle misure.

Il metodo SLAM fornisce una soluzione alla competenza chiave della mappatura e della localizzazione per qualsiasi robot autonomo [41]. L'ultimo decennio, in particolare, ha visto progressi sostanziali nella comprensione del problema SLAM e nello sviluppo di algoritmi SLAM efficienti, coerenti e robusti. L'approccio standard allo spazio degli stati per SLAM è ora ben compreso e i principali problemi di rappresentazione, calcolo e associazione sembrano essere risolti. I metodi SLAM offrono un paradigma radicalmente nuovo per la mappatura e la stima della posizione senza la necessità di forti descrizioni geometriche dei punti di riferimento. Questi metodi stanno aprendo nuove direzioni e creando collegamenti con i principi fondamentali della percezione dei robot. Le sfide chiave per SLAM sono nelle implementazioni e nelle dimostrazioni più grandi. Mentre il progresso è stato sostanziale, la scala e la struttura di molti ambienti sono limitate. La sfida ora è quella di dimostrare soluzioni SLAM a grandi problemi in cui la robotica può davvero contribuire: guidare per centinaia di chilometri sotto la chioma di una foresta o mappare un'intera città senza ricorrere al sistema di posizionamento globale (GPS) e dimostrare una vera localizzazione e mappatura autonoma di strutture come la barriera corallina o la superficie di Marte. SLAM ha avvicinato queste possibilità.

1.2 Motivazioni

Mappatura, localizzazione e navigazione sono tre problemi classici della robotica mobile. La localizzazione del robot è il processo per determinare dove si trova un robot mobile rispetto al suo ambiente. Perché localizzare il robot è così importante? La localizzazione è una delle competenze fondamentali richieste da un robot autonomo poiché la conoscenza della posizione del robot è un precursore essenziale per prendere decisioni sulle azioni future. Quando si parla di posa o posizione, si intendono le coordinate x, y, e la direzione di rotta di un robot in un sistema di coordinate globale. Il robot avrà bisogno di avere almeno un'idea di dove si trova per essere in grado di operare e agire con successo. Per questo scopo vengono utilizzati sensori di posa che consentono di effettuare misurazioni relative allo stato del robot e al suo ambiente.

Questa tesi presenta uno studio di algoritmi di localizzazione e una valutazione delle perfomance del robot umanoide Pepper, della SoftBank Robotics. L'elaborato è suddiviso come segue: i fondamenti teorici degli algoritmi utilizzati sono presentati nel Capitolo 2; il Capitolo 3 presenta il software e l'hardware del dispositivo utilizzato, mentre i risultati sperimentali sono pre-

sentati nel Capitolo 4. Infine, le conclusioni e gli sviluppi futuri sono discussi nel Capitolo 5.

Capitolo 2

Localizzazione

Due problemi chiave nella robotica mobile sono la stima della posizione globale e il tracciamento della posizione locale [42]. Definiamo la stima della posizione globale come la capacità di determinare la posizione del robot in una mappa creata in precedenza, senza altre informazioni se non che il robot è da qualche parte all'interno della mappa. Se non è disponibile una mappa, molte applicazioni permettono di costruirne una man mano che il robot esplora il suo ambiente. Il problema del position tracking consiste nel tenere traccia della posizione del robot nel tempo, una volta che esso è stato localizzato all'interno della mappa. Entrambe queste capacità sono necessarie per permettere a un robot di eseguire compiti utili. Conoscendo la sua posizione globale, il robot può fare uso delle mappe esistenti, il che gli permette di pianificare e navigare in modo affidabile in ambienti complessi. Il tracciamento locale, d'altra parte, è utile per una navigazione efficiente e per compiti di manipolazione locale. Entrambi questi sotto problemi sono di fondamentale importanza per costruire robot veramente autonomi.

Esistono diversi approcci al problema della localizzazione: Filtro di Kalman, Filtro di Kalman Esteso, Localizzazione Markov, localizzazione Monte Carlo [29]. Quelli probabilistici sono tra i candidati più promettenti per fornire una soluzione completa e in tempo reale. L'idea principale è quella di modificare la posizione stimata, chiamata in letteratura *belief*, ossia la funzione densità di probabilità che descrive la posizione del robot, aumentando la probabilità di trovarsi nella configurazione che maggiormente si avvicina alle misure ricavate in precedenza e diminuendo allo stesso tempo le meno probabili. Le tecniche basate sul filtro di Kalman hanno dimostrato di essere robuste e accurate per tenere traccia della posizione del robot. Tuttavia, un filtro di Kalman non può rappresentare le ambiguità e manca della capacità di localizzare globalmente il robot nel caso di errori di localizzazione. Anche se il filtro di Kalman può essere modificato in vari modi per far fronte ad

alcune di queste difficoltà, approcci recenti hanno adottato schemi più ricchi per rappresentare l'incertezza, allontanandosi dall'assunzione di densità gaussiana limitata inerente al filtro di Kalman. La localizzazione Markov, invece, è basata su approcci di tipo grid-based, che compiono un integrazione numerica su una griglia di punti uniformemente spaziati, che può rappresentare densità di probabilità arbitrariamente complesse. Tuttavia, l'onere computazionale e i requisiti di memoria di questo approccio sono considerevoli. Inoltre, la dimensione della griglia, e quindi anche la precisione con cui può rappresentare lo stato, deve essere fissata.

L'approccio Monte Carlo è diverso: anziché descrivere la funzione di densità di probabilità per rappresentare l'incertezza, la si rappresenta mantenendo un insieme di campioni che vengono estratti da essa. Usando una rappresentazione basata sul campionamento, si ottiene un metodo di localizzazione che ha diversi vantaggi rispetto ai precedenti:

- In contrasto con le tecniche basate sul filtraggio di Kalman, è in grado di rappresentare distribuzioni multimodali e quindi può localizzare globalmente un robot.
- Riduce drasticamente la quantità di memoria richiesta rispetto alla localizzazione Markov basata sulla griglia, e può integrare misure ad una frequenza considerevolmente più alta.
- È più accurata della localizzazione di Markov con una dimensione fissa della cella, poiché lo stato rappresentato nei campioni non è discretizzato.
- È facile da implementare.

Nelle prossime sezioni verrà approfondito il metodo Monte Carlo e verranno presentati altri approcci presenti in letteratura.

2.1 Localizzazione Monte Carlo

Nell'ultimo decennio, la localizzazione basata su sensori è stata riconosciuta come un problema chiave nella robotica mobile. La localizzazione è una versione della stima di stato temporale on-line, in cui un robot mobile cerca di stimare la sua posizione in un sistema di riferimento globale. Il problema della localizzazione si presenta in due aspetti: localizzazione globale e tracciamento della posizione. Il secondo è di gran lunga il problema più studiato; qui un robot conosce la sua posizione iniziale e deve correggere errori nella sua odometria mentre si muove. Il problema della localizzazione globale

coinvolge un robot a cui non viene comunicata la sua posizione iniziale; deve risolvere un problema di localizzazione molto più difficile, quello di stimare la sua posizione da zero. I primi metodi Monte Carlo risalgono agli anni settanta e di recente sono stati riscoperti e utilizzati in problemi statistici, di tracciamento della posizione, di computer vision ecc. Attualmente questo metodo è di sicuro uno dei metodi piu utilizzati, in quanto in grado di risolvere problemi di localizzazione globale e non solo; affronta anche il problema del rapimento del robot chiamato in letteratura *kidnapped robot problem*.

L'idea chiave della localizzazione Monte Carlo è rappresentare le ipotesi a posteriori mediante un insieme di N campioni casuali e pesati, dette particelle, il cui insieme $S = s_i|i = 1..N$ costituisce un'approssimazione discreta della densità di probabilità. La particella i-esima è cosi descritta: $(\langle x^i, y^i, \theta^i\rangle, W^i)$ ove $\langle x^i, y^i, \theta^i\rangle$ indica le coordinate x, y e orientamento del robot, e $W^i \geq 0$ il fattore di importanza o peso assegnatogli. Si assume che $\sum_{n=1}^{N} W^n = 1$. Essa prevede due fasi:

Robot motion Quando il robot comincia a muoversi, l'algoritmo di localizzazione prevede la generazione di N nuove particelle che approssimano la posizione in seguito al movimento effettuato. Ogni particella è generata in maniera casuale, attingendo all'insieme di campioni determinato all'istante precedente. Indicando con l' la configurazione $(\langle x^i, y^i, \theta^i\rangle, W^i)$ di una generica particella, la nuova posizione l viene determinata mediante il modello del sistema, $p(l|l', u)$, usando il controllo u come osservazione. Il nuovo valore del peso W^i assegnato alla particella è N^{-1}. In Figura (2.1), ad esempio, si può osservare l'effetto della tecnica di campionamento adottata. Partendo da una posizione iniziale nota e facendo eseguire al robot le azioni indicate dalla linea, l'insieme delle particelle approssima la distribuzione di probabilità con crescente incertezza, che rappresenta la graduale perdita di informazione sulla posizione, dovuta allo slittamento e all'imprecisione nel movimento del robot.

Sensor readings Le letture sensoriali sono incluse nella ri-pesatura dell'insieme dei campioni, come avviene con la regola di Bayes. Sia (l, W^i) con $i = 1, .., N$ una generica particella, allora $W^i \leftarrow \alpha p(z|l)$, dove z contiene le misure dei sensori e α è una costante di normalizzazione che garantisce la condizione $\sum_{n=1}^{N} W^n = 1$. L'inserimento delle letture sensoriali viene solitamente portato a termine in due fasi; nella prima la W^i è moltiplicata per $p(z|l)$, mentre nella seconda i valori ottenuti per le probabilità di ogni singola particella vengono normalizzati [29]. Nel contesto della localizzazione, la rappresentazione delle particelle ha una serie di caratteristiche che la distingue:

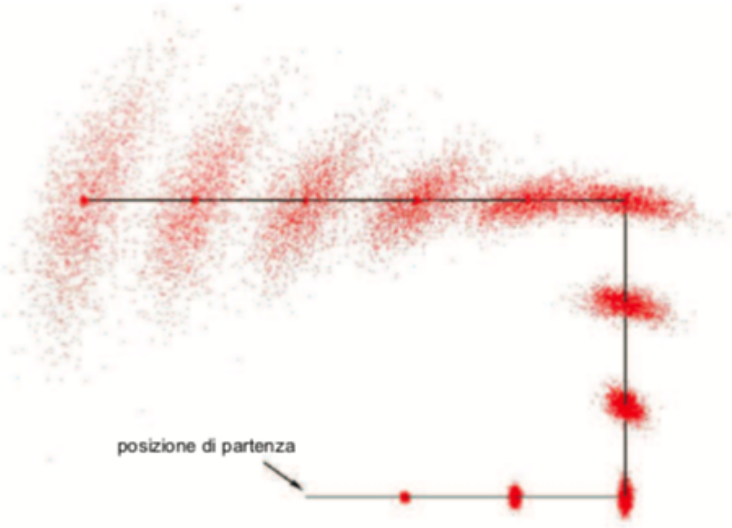


Figura 2.1: Approssimazione della distribuzione di probabilità basata sul campionamento per un robot privo di misurazioni sensoriali. Immagine da [29]

- I filtri di particelle possono tenere in considerazione caratteristiche del sensore (quasi) arbitrarie, dinamiche di movimento e distribuzioni di rumore.
- filtri di particelle sono approssimatori di densità universali, indebolendo i presupposti restrittivi sulla forma della densità posteriore rispetto ai precedenti approcci parametrici.
- I filtri di particelle concentrano le risorse computazionali nelle aree che sono più rilevanti, campionando in proporzione alla probabilità a posteriori.
- Controllando il numero di campioni on-line, i filtri a particelle possono adattarsi alle risorse computazionali disponibili. Lo stesso codice può, quindi, essere eseguito su computer con velocità molto diverse senza modifiche.
- Infine, i filtri a particelle sono sorprendentemente facili da implementare, il che li rende un paradigma attraente per la localizzazione di robot mobili.

2.1.1 Filtro di Bayes

Monte Carlo Localization (MCL) è un filtro di Bayes ricorsivo che stima la distribuzione a posteriori delle pose dei robot condizionata dai dati dei sensori. I filtri di Bayes affrontano il problema della stima dello stato x di un sistema dinamico (catena di Markov parzialmente osservabile) dalle misure dei sensori. Per esempio, nella localizzazione dei robot mobili, il sistema dinamico è il modello di un robot mobile e del suo ambiente, lo stato è la posizione del robot (spesso specificata da una posizione in uno spazio cartesiano bidimensionale e dall'orientamento del robot), e le misure possono includere misure di distanza, immagini della telecamera e letture odometriche. I filtri di Bayes assumono che l'ambiente sia Markov, cioè che i dati passati e futuri siano (condizionatamente) indipendenti se si conosce lo stato attuale. L'idea chiave del filtraggio di Bayes è di stimare una densità di probabilità sullo spazio di stato condizionata dai dati. Questa probabilità a posteriori è tipicamente chiamata *belief* ed è denotata da:

$$Bel(x_t) = p(x_t|d_{0..t}) \tag{2.1}$$

Qui x denota lo stato, x_t è lo stato al tempo t, e $d_0...t$ denota i dati a partire dal tempo 0 fino al tempo t. Per i robot mobili, distinguiamo due tipi di dati: i dati percettivi come le misurazioni della distanza laser, e i dati odometrici, che portano informazioni sul movimento del robot. Denotando i primi con o (per osservazione) e i secondi con a (per azione), abbiamo

$$Bel(x_t) = p(x_t|o_t, a_{t-1}, o_{t-1}, a_{t-2}, ..., o_0). \tag{2.2}$$

Senza perdita di generalità, assumiamo che le osservazioni e le azioni arrivino in una sequenza alternata. Notate che useremo a_{t-1} per riferirci alla lettura odometrica che misura il movimento avvenuto nell'intervallo di tempo $[t-1;t]$, per illustrare che il movimento è il risultato dell'azione di controllo affermata al tempo $(t-1)$. I filtri di Bayes stimano la $Bel(x_t)$ in modo ricorsivo. All'istante iniziale essa è caratterizzata dalla conoscenza iniziale dello stato del sistema. In assenza di tale conoscenza, è tipicamente inizializzata da una distribuzione uniforme sullo spazio di stato. Nella localizzazione dei robot mobili, una distribuzione uniforme iniziale corrisponde al problema della localizzazione globale, dove la posizione iniziale del robot è sconosciuta. Per derivare un'equazione di aggiornamento ricorsiva, osserviamo che l'espressione (2.2) può essere trasformata dalla regola di Bayes in

$$Bel(x_t) = \frac{p(o_t|x_t, a_{t-1}, ..., o_0)p(x_t|a_{t-1}, ..., a_0)}{p(o_t|a_{t-1}, ..., o_0)}. \tag{2.3}$$

Poiché il denominatore è una costante relativa alla variabile x_t, la regola di Bayes solitamente scritta come:

$$Bel(x_t) = \eta p(o_t|x_t, a_{t-1}, ..., o_0)p(x_t|a_{t-1}, ..., o_0), \tag{2.4}$$

ove η è la costante di normalizzazione

$$\eta = p(o_t|a_{t-1}, ..., o_0)^{-1}. \tag{2.5}$$

Come notato sopra, i filtri di Bayes si basano sull'assunzione che i dati futuri sono indipendenti dai dati passati, data la conoscenza dello stato attuale, un'assunzione a cui ci si riferisce tipicamente come l'assunzione di Markov. Detto matematicamente, l'assunzione di Markov implica

$$p(o_t|x_t, a_{t-1}, ..., o_0) = p(o_t|x_t) \tag{2.6}$$

quindi l'espressione (2.4) può essere semplificata:

$$Bel(x_t) = \eta p(o_t|x_t)p(x_t|a_{t-1}, ..., o_0). \tag{2.7}$$

Ora espandiamo il termine più a destra integrando sullo stato al tempo $[t-1]$:

$$Bel(x_t) = \eta p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1}, ..., o_0)p(x_{t-1}|a_{t-1}, ..., o_0)dx \tag{2.8}$$

di nuovo possiamo usare l'approssimazione di Markov $p(o_t|x_t, a_{t-1}, ..., o_0) = p(o_t|x_t)$ per semplificare:

$$p(x_t|x_{t-1}, a_{t-1}, ..., o_0) = p(x_t|x_{t-1}, a_{t-1}) \tag{2.9}$$

la quale ci dà la seguente espressione:

$$Bel(x_t) = \eta p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1})p(x_{t-1}|a_{t-1}, ..., o_0)dx. \tag{2.10}$$

Sostituendo la definizione base della *credenza* nella (2.10) otteniamo quindi l'importante equazione ricorsiva:

$$Bel(x_t) = \eta p(o_t|x_t) \int p(x_t|x_t, a_{t-1})Bel(x_{t-1})dx. \tag{2.11}$$

Questa è l'equazione di aggiornamento ricorsivo dei filtri di Bayes. Insieme alla posizione iniziale, definisce uno stimatore ricorsivo per lo stato di un sistema parzialmente osservabile. Questa equazione è di importanza, poiché è la base per vari algoritmi MCL. Per implementare la (2.11), è necessario conoscere due densità condizionate: la probabilità $p(x_t|x_{t-1}, a_{t-1})$, che chiameremo densità di stato successivo o semplicemente modello di movimento, e la densità $p(o_t|x_t)$, che chiameremo modello percettivo o modello del sensore. Entrambi i modelli sono tipicamente stazionari, cioè non dipendono dal tempo specifico t . Questa stazionarietà ci permette di semplificare la notazione denotando questi modelli $p(x^{'}|x, a)$, e $p(o|x)$, rispettivamente [43].

2.1.2 Modelli probabilistici per la localizzazione

La natura dei modelli $p(x'|x,a)$ e $p(o|x)$ dipende dallo specifico problema di stima. Nella localizzazione di robot mobili, entrambi i modelli sono relativamente semplici e possono essere implementati in poche righe di codice. Il modello di movimento, $p(x'|x,a)$, è una generalizzazione probabilistica della cinematica dei robot. Come notato sopra, per un robot che opera nel piano le pose x e x' sono variabili tridimensionali. Ogni posa comprende le coordinate cartesiane bidimensionali di un robot e la sua direzione di marcia (orientamento). Il valore di a può essere una lettura odometrica o un comando di controllo, entrambi i quali caratterizzano il cambiamento di posa. In robotica, il cambiamento di posa è chiamato cinematica. Le equazioni cinematiche convenzionali, tuttavia, descrivono solo la posa prevista x' che un robot ideale, privo di rumore, raggiungerebbe partendo da x, e dopo essersi mosso come specificato da a. Naturalmente, il movimento fisico del robot è disturbato; quindi, la posa x' è incerta. Per rendere conto di questa incertezza intrinseca, il modello di moto probabilistica $p(x'|x,a)$ descrive una densità a posteriori sui possibili successori x'. L'incertezza è tipicamente modellata da un rumore gaussiano centrato sullo zero che viene aggiunto alle componenti di traslazione e di rotazione nelle misure odometriche. Così, $p(x'|x,a)$ generalizza l'esatta cinematica dei robot mobili tipicamente descritta nei libri di testo sui robot con una componente probabilistica [44].

In Figura (2.2) viene mostrato un esempio di come l'incertezza dipenda dal movimento complessivo. Rivolgiamo ora la nostra attenzione al modello percettivo, $p(o|x)$. I robot mobili sono comunemente dotati di range finders, come trasduttori ultrasonici (sensori sonar) o laser (per determinare la distanza di un oggetto, solitamente funziona secondo il principio del tempo di volo, inviando un impulso laser verso l'oggetto e misurando il tempo impiegato dall'impulso per essere riflesso dal bersaglio e restituito al mittente). Il problema del calcolo di $p(o|x)$ viene decomposto in tre parti:

1. il calcolo del valore che un sensore privo di rumore genererebbe;
2. la modellazione del rumore del sensore; e
3. l'integrazione di molti fasci di sensori individuali in un singolo valore di densità.

Supponiamo che la posa del robot sia x, e che o_i denoti un raggio del singolo sensore con rilevamento α_i rispetto al robot. Lasciamo che $g(x,\alpha_i)$ denoti la misura di un sensore ideale, privo di rumore, il cui rilevamento relativo è α_i e assumiamo che al robot sia data una mappa dell'ambiente $g(x,\alpha_i)$, come quella mostrata nella Figura (2.3(a)).

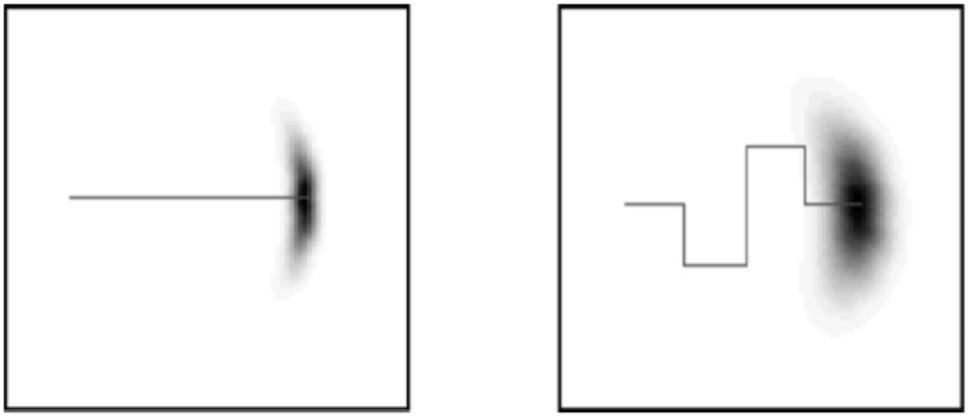


Figura 2.2: La figura mostra due esempi di $p(x^{'}|x,a)$. In entrambi gli esempi, la posa iniziale x è mostrata sulla sinistra, e la linea continua rappresenta i dati odometrici misurati dal robot. L'area ombreggiata in grigio sulla destra rappresenta la densità $p(x^{'}|x,a)$: più scura è una posa e più probabile è. Un confronto di entrambi i diagrammi mostra che l' incertezza dipende dal movimento complessivo: anche se le pose di un robot senza rumore sono le stesse per entrambi i percorsi eseguiti, l'incertezza nel diagramma di destra è più grande a causa della maggiore distanza complessiva percorsa dal robot. Immagine da [44]

Assumiamo che questa distanza "attesa" $g(x,\alpha_i)$ sia una statistica sufficiente per la probabilità $p(o_i|x)$, cioè

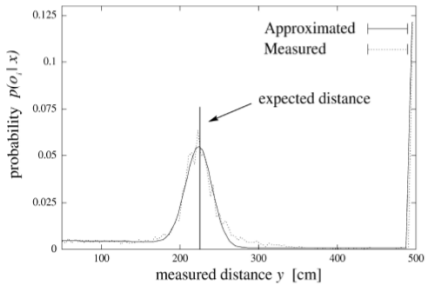
$$p(o_i|x) = p(o_i|g(x,\alpha_i)). \tag{2.12}$$

La densità esatta $p(o_i|x)$ è mostrata in Figura (2.3(b)). Questa densità è una miscela di tre densità: una gaussiana centrata su $g(x,\alpha_i)$ che modella l'evento di misurare la distanza corretta con un piccolo rumore gaussiano aggiunto, una densità esponenziale che modella le letture casuali come spesso causate dalle persone, e infine una grande probabilità discreta (modellata matematicamente da una densità uniforme) che modella le misurazione di max-range, che frequentemente si verificano quando un sensore di distanza non riesce a rilevare un oggetto. Infine, i valori di densità individuali $p(o_i|x)$ sono integrati in modo moltiplicativo, assumendo l'indipendenza condizionale tra le misure individuali:

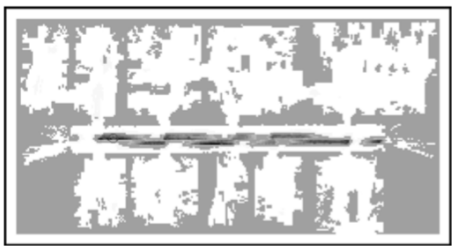
$$p(o_i|x) = \prod{}^{i} p(o_i|x). \tag{2.13}$$

Chiaramente, questa indipendenza condizionale può essere violata in presenza di persone (che spesso bloccano più di un raggio del sensore). In questi

(a) *Laser scan e mappa*



(b) *sensor model* $p(o_i|x)$



(c) *Distribuzione di probabilità per diverse pose*

Figura 2.3: (a) Scansione laser della distanza, proiettata in una mappa. (b) La densità $p(o_i|x)$, dove il picco corrisponde alla distanza che un sensore ideale, privo di rumore, misurerebbe. (c) Mostra la $p(o_i|x)$ per la scansione mostrata in (a). Sulla base di una singola scansione del sensore, il robot segnala un'alta probabilità di essere nel corridoio principale. Immagini da [44]

casi, potrebbe essere consigliabile sotto-campionare le letture dei sensori e utilizzare un set ridotto per la localizzazione. La Figura (2.3(c)) rappresenta $p(o_i|x)$ per la scansione del sensore mostrata in Figura (2.3(a)) e la mappa mostrata in grigio in Figura (2.3(c)). La maggior probabilità si trova nella regione del corridoio, il che riflette il fatto che è più probabile che la misura specifica del sensore abbia avuto origine nel corridoio piuttosto che in una delle stanze. La densità mostrata in Figura (2.3(c)) è equivalente alla probabilità a posteriori di un robot che localizza globalmente dopo aver percepito una sola misura del sensore.

2.1.3 Approssimazione particellare

Se lo spazio di stato è continuo, come nel caso della localizzazione di robot mobili, l'implementazione della (2.11) non è una questione banale soprattutto se ci si preoccupa dell'efficienza. L'idea di MCL (e di altri algoritmi di filtro a particelle) è di rappresentare la $Bel(x)$ con un insieme di m campioni pesati distribuiti secondo:

$$Bel(x) \approx (x^{(i)}, \omega^{(i)})_{i=1..m} \tag{2.14}$$

Qui ogni $x^{(i)}$ è un campione della variabile casuale x, cioè uno stato ipotizzato (posa). I parametri numerici non negativi $w^{(i)}$ sono chiamati *fattori di importanza*, la cui somma sarà pari a 1 (anche se questo non è strettamente richiesto nel filtraggio da particelle). Come suggerisce il nome, i fattori di importanza determinano il peso (importanza) di ogni campione. L'insieme dei campioni, quindi, definisce una funzione di probabilità discreta che approssima la probabilità continua $Bel(x)$. L'insieme iniziale di campioni rappresenta la conoscenza iniziale $Bel(x_0)$ sullo stato del sistema dinamico. Per esempio, nella localizzazione globale dei robot mobili, la stima della posizione iniziale è un insieme di pose disegnate secondo una distribuzione uniforme sull'universo del robot, annotato dal fattore di importanza uniforme $1/m$. Se la posa iniziale è nota fino a un piccolo margine di errore, $Bel(x_0)$ può essere inizializzata da campioni tratti da una gaussiana stretta centrata sulla posa corretta. L'aggiornamento ricorsivo viene eseguito in 3 passi:

1. i campioni $x_{t-1}^{(i)} \sim Bel(x_{t-1})$ dal set ponderato di campioni che rappresenta la $Bel(x_{t-1})$. Ciascuna di queste particelle è distribuita in accordo con la distribuzione della $Bel(x_{t-1})$.

2. $x_t^{(i)} \sim p(x^i | x_{(t-1)}^{(i)}, a_{(t-1)})$. Ovviamente, la coppia $\langle x_t^{(i)}, x_{(t-1)}^{(i)} \rangle$ è distribuita secondo la distribuzione del prodotto

$$q_t := p(x_t | x_{t-1}, a_{t-1}) x Bel(x_{t-1}). \tag{2.15}$$

   In accordo con la letteratura sull'algoritmo SIR (Sampling Importance Resampling), ci riferiremo a questa distribuzione q_t come la proposta di distribuzione. Il suo ruolo è quello di "proporre" campioni della distribuzione posteriore desiderata, che è data in (2.11); tuttavia, non è equivalente alla distribuzione a posteriori desiderata.

3. Infine la correzione della mancata corrispondenza tra la distribuzione proposta q_t e la distribuzione desiderata specificata in (2.11) è qui proposta per la coppia di campioni $\langle x_t^{(i)}, x_{(t-1)}^{(i)} \rangle$:

$$\eta p(o_t | x_t^{(i)}) p(x_t^{(i)} | x_{t-1}^{(i)}, a_{(t-1)}) Bel(x_{(t-1)}^{(i)}). \tag{2.16}$$

Questa discrepanza è spiegata dal seguente fattore di importanza (non normalizzato), che viene assegnato al campione i-esimo:

$$\omega^{(i)} = p(o_t|x_i^{(i)}). \tag{2.17}$$

Il fattore di importanza è ottenuto come quoziente della distribuzione obiettivo (2.16) e della distribuzione proposta (2.15), ottenendo un fattore di scala costante:

$$\eta p(o_t|x_t^{(i)}) = \frac{\eta p(o_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^i, a_{t-1})Bel(x_{t-1}^{(i)})}{p(x_t^{(i)}|x_{t-1}^{(i)}, a_{t-1})Bel(x_{t-1}^{(i)})} \tag{2.18}$$

Si noti che questo quoziente è proporzionale a $\omega_{(i)}$, poiché η è una costante. La routine di campionamento viene ripetuta m volte, producendo un insieme di m campioni ponderati $x_t^{(i)}$ (con $i = 1, ..., m$). In seguito i fattori di importanza sono normalizzati in modo che la loro somma sia pari a 1, e quindi definiscono una distribuzione di probabilità discreta. DI seguito viene riassunto l'algoritmo MCL. Noto che anche sotto ipotesi blande, l'insieme di campioni converge al vero a posteriori $Bel(x_t)$ man mano che m va ad infinito, con una velocità di convergenza in $O(1/\sqrt{m})$. La velocità potrebbe variare di un fattore costante, che può variare drasticamente a seconda della distribuzione proposta. Bisogna porre l'attenzione al numero di campioni, ad esempio un numero estremamente piccolo (meno di 10), aumenterebbe la distorsione essendo essa inversamente proporzionale al set di campioni.

Algoritmo MCL
(X, a, o)
$X' = \emptyset$
For $i = 0$ to m
generate random x from X according to $\omega_1, ..., \omega_m$
generate random $x' \sim p(x'|a, x)$ $\omega' = p(o|x')$
add $\langle x', \omega' \rangle$ to X'
normalize the importance factor ω' in X'
return X'

2.1.4 Proprietà della localizzazione Monte Carlo

La localizzazione Monte Carlo è un approccio potente e popolare nella localizzazione dei robot mobili. Una delle caratteristiche dell'algoritmo Monte Carlo è che è in grado di approssimare distribuzioni di probabilità del tutto arbitrarie. Non viene fatto alcun tipo di assunzione sulla natura della funzione di distribuzione di probabilità e questo ha favorito il largo impiego che ha avuto la localizzazione Monte Carlo rispetto alle altre tecniche di

localizzazione. Come mostrato in [45], la varianza converge a zero con una velocità pari a $1/\sqrt{N}$ (N numero di campioni) in condizioni che sono vere per MCL. La grandezza dell'insieme delle particelle naturalmente influisce sull'accuratezza e sulla complessità computazionale dell'algoritmo. Il vero vantaggio, comunque, si trova nel modo in cui la MCL organizza le risorse computazionali; l'attenzione è sempre focalizzata in regioni dello spazio con alta probabilità, dove realmente c'è la possibilità di trovare il robot.

L'algoritmo MCL può essere implementato online in quanto si presta bene a una implementazione in qualunque istante di tempo come mostrato in [46]. Infatti esso rientra negli algoritmi any-time, algoritmi che possono generare una soluzione valida anche se viene interrotto anticipatamente in qualsiasi momento. Esso si presta ad una semplice implementazione e gli algoritmi che lo utilizzano sono in grado di ottenere una risposta al problema della localizzazione in ogni istante; in particolare la qualità della soluzione cresce con il trascorrere del tempo. La fase di campionamento in MCL può essere terminata in qualsiasi momento. Così, quando arriva una lettura del sensore, o viene eseguita un'azione, il campionamento viene terminato e il set di campioni risultante viene utilizzato per l'operazione successiva.

In pratica, il numero di campioni richiesti per raggiungere un certo livello di precisione varia drasticamente. Durante la localizzazione globale, il robot è completamente ignaro di dove si trova; quindi, la sua distribuzione di probabilità copre uniformemente tutto il suo spazio di stato tridimensionale. Durante il tracciamento della posizione, d'altra parte, l'incertezza è tipicamente piccola e spesso concentrata su mappe di dimensioni inferiori. Così, molti più campioni sono necessari durante la localizzazione globale per approssimare accuratamente la vera densità, rispetto a quelli necessari per il tracciamento della posizione. MCL determina la dimensione del set di campioni mentre è in esecuzione. Come in [28], l'idea è di usare la divergenza di $P(l)$ e $P(l|s)$, cioè la distribuzione di probabilità prima e dopo il rilevamento, per determinare i set di campioni. Più specificamente, sia i dati di movimento che i dati del sensore sono incorporati in un unico passo, e il campionamento viene fermato ogni volta che la somma dei pesi (prima della normalizzazione) supera una soglia. Se la posizione predetta dall'odometria è ben in sintonia con la lettura del sensore, ogni singolo peso è grande e il set di campioni rimane piccolo. Se, invece, la lettura del sensore porta molta innovazione, come è tipicamente il caso quando il robot è globalmente incerto o quando ha perso traccia della sua posizione, i singoli valori dei pesi sono piccoli e l'insieme del campione è grande.
Nel confronto con il filtro di Kalman, c'è convergenza anche se non si è in presenza di distribuzioni gaussiane e di sistemi lineari. Rispetto, invece, ai

metodi di Markov basati su griglie sono molto efficienti dato che l'attenzione è focalizzata sulle particelle con più alta probabilità

2.2 Simoultaneous Localization And Mapping (SLAM)

In questa sezione viene presentata un'altra tecnica di localizzazione che prevede la contemporanea localizzazione e mappatura dell'ambiente. Creare mappe e contemporaneamente localizzarsi è una capacità essenziale per i robot mobili che viaggiano in ambienti sconosciuti dove non sono disponibili dati di posizione accurati a livello globale (ad esempio GPS). In particolare, i robot mobili hanno mostrato capacità promettenti per l'esplorazione remota, andando in luoghi che sono troppo lontani, troppo pericolosi, o semplicemente troppo costosi per consentire l'accesso umano. Se i robot devono operare autonomamente in ambienti estremi sottomarini, sotterranei e sulle superfici di altri pianeti, devono essere in grado di costruire mappe e navigare in modo affidabile in base a queste mappe. Anche in ambienti benigni come gli interni degli edifici, spesso è difficile acquisire mappe accurate e preventive. La capacità di mappare un ambiente sconosciuto permette a un robot di essere distribuito con un'infrastruttura minima. Questo è particolarmente importante se l'ambiente cambia nel tempo. Le mappe prodotte dagli algoritmi SLAM servono tipicamente come base per la pianificazione e l'esplorazione del movimento. La localizzazione e la mappatura simultanea è una tecnica applicata nei robot mobili per un'auto-esplorazione in numerosi ambienti geografici. SLAM è diventata un'area di ricerca fondamentale negli ultimi anni in quanto è una soluzione promettente per risolvere la maggior parte dei problemi che riguardano il campo dei robot mobili di intelligenza artificiale orientati all'auto-esplorazione, come per esempio, la capacità di esplorare senza alcuna conoscenza preliminare dell'ambiente e senza alcuna interferenza umana. La caratteristica unica di SLAM è che il processo di mappatura e localizzazione è fatto simultaneamente e ricorsivamente. Dall'introduzione di SLAM, molti algoritmi sono stati proposti per applicare tale tecnica nella pratica reale con l'obiettivo di trovare la migliore soluzione per rendere il robot capace di navigare autonomamente senza alcuna conoscenza preliminare dell'ambiente che esplora [47]. Le caratteristiche principali sono tre: mappatura, localizzazione e navigazione.

- Mappatura (rappresentazione dell'ambiente): prima che il robot mobile inizi a esplorare o navigare in un ambiente sconosciuto, richiede una mappa dell'ambiente che vuole esplorare come conoscenza preliminare.

Il processo di mapping permette al robot mobile di generare una mappa dell'ambiente utilizzando il set sensoriale. Dai dati, viene generata una mappa e i tipi di rappresentazione della mappa sono topologica, geometrica, a griglia o mista. Poi, sarà usata dal robot mobile per localizzare e riconoscere la propria posizione.

- Localizzazione (stima della posizione): la localizzazione è una delle caratteristiche di SLAM, mostra come il robot mobile sia in grado di calcolare e stimare la sua posizione sulla base della mappa generata dal processo di mappatura. La localizzazione rende il robot capace di riconoscere la propria posizione, l'ambiente circostante ed evitare gli ostacoli vicini.

- Navigazione (pianificazione del percorso): questa caratteristica combina sia la mappatura che la localizzazione, dove il robot mobile fa una pianificazione appropriata del percorso dalle informazioni ricevute durante il processo di mapping e localizzazione. Mentre il robot mobile naviga nell'ambiente, i processi di mappatura e localizzazione vengono eseguiti ricorsivamente per aggiornare la conoscenza del robot mobile sull'ambiente circostante, per garantire che il robot sia in grado di navigare nell'ambiente circostante in modo efficiente. Le caratteristiche della pianificazione della navigazione fatta dal robot mobile sono fare un percorso appropriato basato sulle informazioni ricevute, rispondere all'ambiente circostante ed essere in grado di tornare al punto di origine o al punto di partenza dopo l'esplorazione.

2.2.1 Il problema SLAM

Definizione del problema

La posa del robot al tempo t indica le coordinate x, y e l'orientamento θ dello stesso. Le pose evolvono secondo la legge probabilistica chiamata *robot motion* : $p(s_t|u_t, s_{t-1})$ ove s_t è la funzione di probabilità condizionata dall'azione di controllo u_t del robot, e, dalla sua posa precedente s_{t-1} [48]. Nella robotica mobile, il modello di movimento è solitamente una generalizzazione dei modelli tempo-invarianti della cinematica. L'ambiente del robot possiederà K punti di riferimento nello spazio (*landmark*), ed ognuno di essi è caratterizzato dalla sua posizione nello spazio indicata con θ_k ove $\theta = \left[x^L y^L\right]^T$ per $k = 1, \ldots, K$. Senza perdita di generalità, è possibile pensare ai punti di riferimento come punti nel piano, in modo che le posizioni siano specificate da due valori numerici. Per mappare il suo ambiente, il robot percepisce dei punti di riferimento: per esempio, può essere in grado

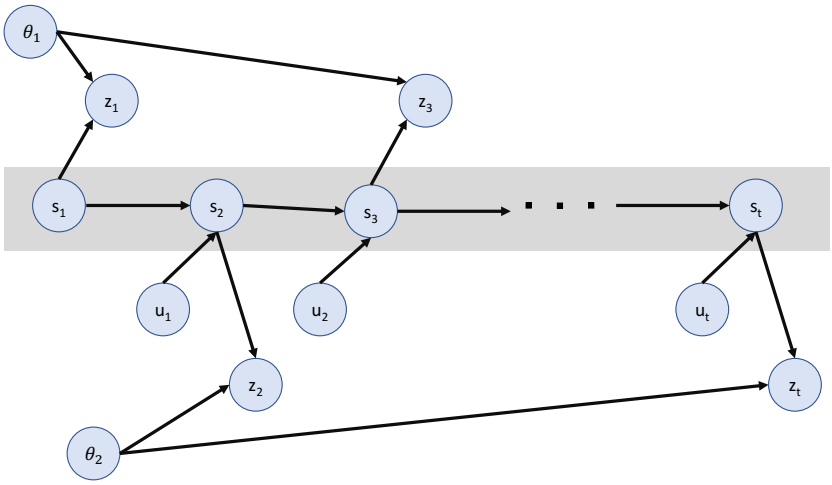


Figura 2.4: Il problema SLAM: il robot si muove dalla posa s_1 attraverso una sequenza di azioni di controllo, u_1, u_2, . . . , u_t. Mentre si muove, osserva i punti di riferimento vicini. Al tempo $t = 1$, osserva il punto di riferimento θ_1. La misura è indicata con z_1 (distanza e rilevamento). Al tempo $t = 2$, osserva l'altro punto di riferimento, θ_2, e al tempo $t = 3$, osserva nuovamente θ_1. Il problema SLAM riguarda la stima delle posizioni dei punti di riferimento e del percorso del robot a partire dai comandi u e dalle misure z. L'ombreggiatura grigia illustra una relazione di indipendenza condizionata.

di misurare la distanza da un punto di riferimento, rispetto al suo sistema di coordinate locali. La misura al tempo t sarà indicata con z_t. Mentre i robot possono spesso percepire più di un landmark alla volta, verrà seguita la notazione comune assumendo che le misure dei sensori siano riferite a un solo punto di riferimento. Questa convenzione è adottata solo per comodità matematica e non pone alcuna restrizione, in quanto gli avvistamenti di più punti di riferimento in un singolo passo temporale possono essere elaborati in modo sequenziale.

Le misure dei sensori sono governate da una legge probabilistica, spesso chiamata *modello di misura*: $p(z_t|s_t, \theta, n_t)$ ove $\theta = \theta_1, .., \theta_k$ è l'insieme dei landmark, e $n_t \in 1, ..., K$ indica l'insieme degli indici dei landmark percepiti al tempo t e spesso viene chiamata *corrispondenza*. In Figura (2.4) vengono mostrati $n_1 = 1$, $n_2 = 2$, $n_3 = 1$, poichè il robot osserva prima il punto di riferimento θ_1 poi θ_2 e poi di nuovo θ_1 per una seconda volta.La

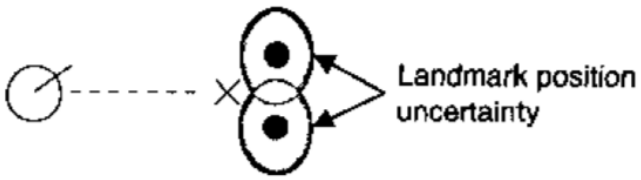


Figura 2.5: Ambiguità di misura: Due punti di riferimento (mostrati come cerchi neri) sono abbastanza vicini che l'osservazione (mostrata come una x) potrebbe plausibilmente provenire da uno dei due. Immagine da [48]

maggior parte dei lavori teorici in letteratura presuppone la conoscenza della corrispondenza o, detto diversamente, che i punti di riferimento siano identificabili in modo univoco. Le implementazioni pratiche usano stimatori di massima verosimiglianza per stimare la corrispondenza, che funzionano bene se i punti di riferimento sono sufficientemente distanziati. In maniera generica quindi, il problema SLAM consiste nel determinare la posizione di tutti i landmarks θ e le posizione del robot s_t dalle misure $z^t = z_1, .., z_t$ e dal controllo $u^t = u_1, .., u_t$. In termini probabilistici, questo viene espresso dalla probabilità $p(s^t, \theta | z^t, u^t)$, dove si utilizza l'apice ^{t} per indicare il set di variabili dall'istante iniziale al tempo t. Se la corrispondenza è nota, il problema SLAM è semplificato e la probabilità diventa $p(s^t, \theta | z^t, u^t, n^t)$. Nei problemi SLAM nel mondo reale, la corrispondenza n_t tra osservazioni e punti di riferimento è raramente nota ed anche il numero K di landmarks è sconosciuto. Ogni volta quindi che un robot esegue una lettura, questa deve essere correlata con un landmark esistente o considerata come proveniente da un punto di riferimento non visto in precedenza. Se questo però non è ovvio, scegliere l'associazione sbagliata può far sì che un filtro diverga. Due sono i fattori che contribuiscono all'incertezza: il primo è il rumore di misura ed il secondo è il rumore del movimento. Ognuno porta a un diverso tipo di ambiguità nell'associazione dei dati; il rumore nel modello di misura comporterà una maggiore incertezza nelle posizioni dei landmark, che porteranno ad una ambiguità di misura o confusione tra punti di riferimento vicini e questo errore avrà un effetto relativamente piccolo sull'errore di stima perché l'osservazione potrebbe plausibilmente provenire da entrambi i punti di riferimento, come mostrato in Figura (2.5) . L'ambiguità dovuta al rumore del movimento può avere conseguenze molto più gravi: se questa incertezza è abbastanza alta, diverse pose plausibili del robot porteranno a ipotesi di associazione dati drasticamente diverse per le osservazioni successive. Questa è facilmente indotta se c'è una significativa incertezza angolare nella stima della posa del robot come mostrato in Figura (2.6).

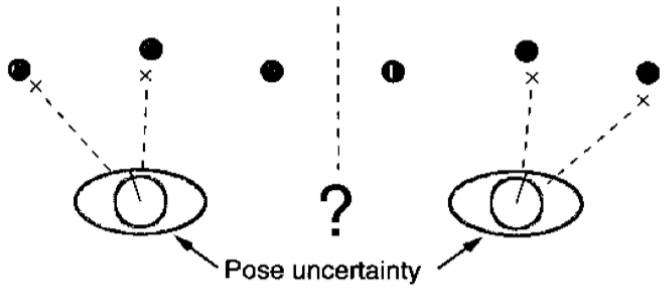


Figura 2.6: Ambiguità di movimento: osservazioni potrebbero essere associate a landmark completamente differenti se l'orientamento del robot cambia leggermente. Immagine da [48]

Se vengono incorporate più osservazioni per passo temporale, il movimento del robot metterà in relazione le associazioni di tutti i punti di riferimento e se un algoritmo SLAM sceglie l'associazione sbagliata per un singolo punto di riferimento, a causa dell'ambiguità del movimento, ci sarà un' alta probabilità che anche il resto delle associazioni sarà sbagliato. Questo aggiungerà una grande quantità di errore alla posa del robot, e spesso causerà una divergenza del filtro [49]. La seguente sezione mostrerà un approccio alternativo al problema SLAM, presentando un algoritmo che può campionare su più ipotesi di associazione dati.

Soluzione teorica di stima al problema

Questa sezione è dedicata alla descrizione del quadro matematico impiegato nello studio del problema SLAM. L'impostazione del problema è quella di un veicolo con modello cinematico noto, che parte da una posizione iniziale non nota e si muove attraverso un ambiente che contiene una popolazione di *feature* (punti di riferimento o punti caratteristici). Il veicolo è dotato di sensori che possono prendere le misure della posizione relativa tra ogni singolo punto di riferimento e il veicolo stesso, come mostrato in Figura (2.7).

Le posizioni assolute dei punti di riferimento non sono disponibili. Lo stato del sistema di interesse consiste nella posizione e nell'orientamento del veicolo insieme alla posizione di tutti i punti di riferimento. Lo stato del veicolo ad un passo temporale k è indicato con $x_v(k)$. Si adotta un modello lineare (sincrono) a tempo discreto dell'evoluzione del veicolo e delle osservazioni dei punti di riferimento. Sebbene il movimento del veicolo e l'osservazione dei punti di riferimento siano quasi sempre non lineari e asincroni in qualsiasi problema di navigazione reale, l'uso di modelli lineari sincroni non influisce sulla validità, se non per richiedere le stesse ipotesi di linearizzazione normal-

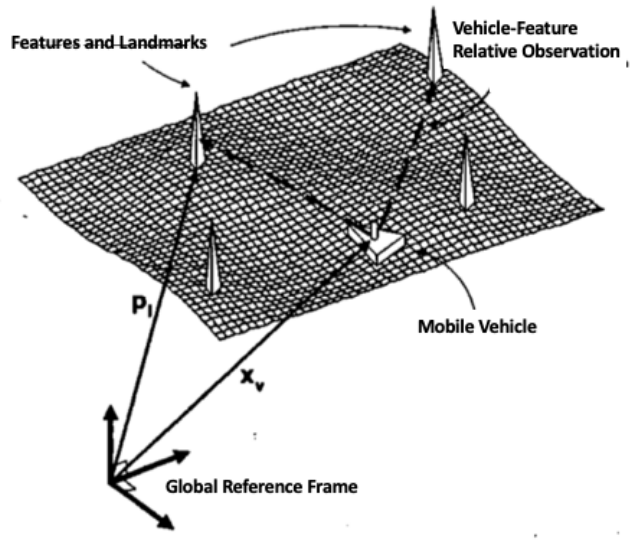


Figura 2.7: La struttura del problema SLAM. Un veicolo prende osservazioni relative di punti di riferimento dell'ambiente. La posizione assoluta dei punti di riferimento e del veicolo sono sconosciuti. Immagine da [50]

mente impiegate nello sviluppo di un filtro di Kalman esteso. Il movimento del veicolo attraverso l'ambiente è modellato da un'equazione di transizione di stato discreta in tempo reale convenzionale o da un modello di processo della forma:

$$x_v(k+1) = F_v(k)x_v(k) + u_v(k+1) + v_v(k+1), \tag{2.19}$$

dove $F_v(k)$ è la matrice di transizione dello stato, $u_v(k)$ è il vettore degli ingressi di controllo, $v_v(k)$ il vettore di errore di rumore di processo non correlati al tempo con media zero e covarianza $Q_v(k)$. La posizione dell'i-esimo landmark è nota come p_i. L'equazione di transizione di stato per l'i-esimo landmark è:

$$p_i(k+1) = p_i(k) = p_i, \tag{2.20}$$

assumendo che i punti di riferimento siano stazionari. Il modello di transizione di stato aumentato per il sistema completo può ora essere scritto

come:

$$\begin{bmatrix} x_v(k+1) \\ p_1 \\ . \\ . \\ p_N \end{bmatrix} = \begin{bmatrix} F_v(k) & 0 & ... & 0 \\ 0 & I_{p1} & \dots & 0 \\ . & . & . & 0 \\ . & . & . & 0 \\ . & . & . & 0 \\ 0 & 0 & 0 & I_{pN} \end{bmatrix} \begin{bmatrix} x_v(k) \\ p_1 \\ . \\ . \\ p_N \end{bmatrix} + \begin{bmatrix} u_v(k+1) \\ 0_{p1} \\ . \\ . \\ 0_{pN} \end{bmatrix} + \begin{bmatrix} v_v(k+1) \\ 0_{p1} \\ . \\ . \\ 0_{pN} \end{bmatrix} \tag{2.21}$$

$$x(k+1) = F(k)x(k) + u(k+1) + v(k+1) \tag{2.22}$$

ove N indica il numero dei punti di riferimento, $I_{pi} \in \mathbb{R}^{\dim(p_i)\times\dim(p_i)}$ indica la matrice identità, mentre $0_{pi} \in \mathbb{R}^{\dim(p_i)\times\dim(p_i)}$ è la matrice nulla [50].
Il veicolo è dotato di sensori che possono ottenere osservazioni della posizione relativa dei punti di riferimento rispetto al veicolo. Il modello di osservazione per l'i-esimo punto di riferimento è scritto nella forma:

$$z_i(k) = H_i x(k) + w_i(k) = H_{pi} p - H_v x_v(k) + w_i(k) \tag{2.23}$$

dove $w_i(k)$ è un vettore di errori di osservazione non correlati temporalmente, con media zero e varianza $R_i(k)$. La struttura dell'equazione (2.23) riflette il fatto che le osservazioni sono "relative" tra il veicolo e il punto di riferimento, spesso sotto forma di posizione relativa. Il termine H_i è la matrice di osservazione e mette in relazione l'uscita del sensore $z_i(k)$ con il vettore di stato $x(k)$ quando si osserva il punto di riferimento. Importante è notare che il modello di osservazione stesso per il punto di riferimento è scritto nella forma:

$$H_i = [-H_v, 0...0, H_{pi}, 0...0]\,. \tag{2.24}$$

Nella formulazione teorica della stima del problema SLAM, il filtro di Kalman è usato per fornire stime della posizione del veicolo e dei punti di riferimento. Riassumiamo brevemente la notazione e le fasi principali di questo processo. Il filtro di Kalman calcola ricorsivamente le stime per uno stato $x(k)$ che sta evolvendo secondo il modello di processo in (2.21) e che viene osservato secondo il modello di osservazione in (2.23). Il filtro di Kalman calcola una stima che è equivalente alla media condizionata $\hat{x}(p|q) = E[x(p)|Z^q]$ $(p \geq q)$, dove Z^q è la sequenza di osservazioni prese fino al tempo q. L'errore nella stima è indicato come $\tilde{x}(p|q) = \hat{x}(p|q) - x(p)$. Il filtro di Kalman fornisce anche una stima ricorsiva della covarianza $P(p|q) = E[\tilde{x}(p|q)\tilde{x}(p|q)^T|Z^q]$ nella stima $\tilde{x}(p|q)$. L'algoritmo del filtro di Kalman procede ora ricorsivamente in tre fasi:

1. Predizione: dato che i modelli descritti in (2.21) e (2.23) sono validi, e che esiste una stima $\tilde{x}(k|k)$ dello stato $x(k)$ al tempo k insieme ad una stima della covarianza $P(k|k)$, l'algoritmo genera prima una previsione per la stima dello stato, l'osservazione (relativa all'i-esimo punto di riferimento) e la covarianza stimata dello stato al tempo $(k+1)$ secondo:

$$\hat{x}(k+1|k) = F(k)\hat{x}(k|k) + u(k) \tag{2.25}$$

$$\hat{z}_i(k+1|k) = H_i(k)\hat{x}(k+1|k) \tag{2.26}$$

$$P(k+1|k) = F(k)P(k|k)F^T(k) + Q(k) \tag{2.27}$$

2. Osservazione: dopo la predizione, l'osservazione $z_i(k+1)$ dell'i-esimo punto di riferimento del vero stato $x(k+1)$ viene fatta seguendo la (2.23). Supponendo una corretta associazione dei punti di riferimento, l'innovazione è calcolata come segue:

$$v_i(k+1) = z_i(k+1) - \hat{z}_i(k+1|k) \tag{2.28}$$

   insieme ad una matrice di covarianza dell'innovazione associata, data da

$$S_i(k+1) = H_i(k)P(k+1|k)H_i^T(k) + R_i(k+1) \tag{2.29}$$

3. Aggiornamento: la stima di stato e la corrispondente covarianza di stato sono poi aggiornate secondo

$$\hat{x}(k+1|k+1 = \hat{x}(k+1|k) + W_i(k+1)v_i(k+1) \tag{2.30}$$

$$P(k+1|k+1) = P(k+1|k) - W_i(k+1) \times S(k+1) \times W_i^T(k+1) \tag{2.31}$$

   dove la matrice del guadagno $W_i(k+1)$ è data da

$$W_i(k+1) = P(k+1|k)H_i^T(k)S^{-1}(k+1) \tag{2.32}$$

L'aggiornamento della matrice di covarianza della stima dello stato è di fondamentale importanza per il problema SLAM: verrà esaminata adesso l'evoluzione della matrice di covarianza $P(k)$ che risulta dalla struttura della matrice $H_i(k)$ data nell'equazione (2.23). Si dimostra che quando un punto di riferimento non è più visibile dalla posizione corrente del robot, può essere

cancellato dalla mappa senza influenzare la coerenza statistica del processo di stima sottostante. Viene anche mostrato che una volta cancellato, tutte le informazioni accumulate finora sul punto di riferimento devono essere scartate e che il punto di riferimento deve essere reinizializzato quando torna in vista. Viene proposto un processo per selezionare i punti di riferimento da rimuovere.

La matrice di covarianza, in qualsiasi istante k può essere scritta come:

$$P(k/k) = \sum_{j=1,2...n} \sum_{l=1,2...n} p(k/k)_{jl} \tag{2.33}$$

dove P_{vv} è la matrice degli errori associata alla stima dello stato del veicolo, $P_{jl}, j = 1..n, l = 1..n$ sono le covarianze associate alle stime degli stati di riferimento, e $P_{vl}, l = 1..n$ sono le cross-varianze tra il veicolo e i punti di riferimento. Tralasciando l'indice k per chiarezza e utilizzando l'equazione (2.23), la covarianza dell'innovazione dopo lche il punto di riferimento è stato osservato è data da:

$$S_i = H_v P_{vv} H_v^T - H_{pi} P_{vi} H_v^T - H_v P_{vi} H_{pi}^T + H_{pi} P_{vi} H_{pi}^T. \tag{2.34}$$

Chiaramente S_i è funzione delle varianze e delle cross-varianze del veicolo e dei landmark i, ed è indipendente da tutte le altre cross-covarianze dei punti di riferimento.

Durante la fase di aggiornamento che segue l'osservazione della feature i, il cambiamento della matrice di covarianza e il guadagno di Kalman può essere scritto come:

$$W_{ij} = \left[-P_{vj} H^{t_v} + P_{ij} H_{pi}^T\right] S_i^{-1}, J = 1, 2...n \tag{2.35}$$

$$\delta P_{jl} = \left[-P_{vj} H_v^T + P_{ji} H_{pi}^T\right] S_i^{-1} \left[-H_v P_{vl} + H_{pi} P_{li}\right] \tag{2.36}$$

L'effetto dell'osservazione di un dato punto di riferimento i sull'evoluzione degli stati del veicolo e dei punti di riferimento e la matrice di covarianza degli stati può ora essere spiegata come segue:

1. I guadagni di Kalman associati allo stato del veicolo e a tutti gli stati dei punti di riferimento sono diversi da zero. Quindi la stima di tutti gli stati dei punti di riferimento sono aggiornati anche quando molti di questi punti di riferimento non sono visibili dalla posizione corrente del robot.

2. Tutti gli elementi della matrice di covarianza di stato devono essere modificati dopo ogni osservazione. Non è sufficiente aggiornare gli elementi della matrice di covarianza di stato corrispondenti al veicolo e al punto di riferimento attualmente osservato. Questa è la ragione principale della complessità computazionale del processo SLAM.

3. Il guadagno di Kalman e le modifiche agli elementi della matrice di covarianza di stato associata al j-esimo punto di riferimento sono una funzione delle covarianze e delle cross-covarianze associate allo stato del veicolo e agli stati i e j del punto di riferimento.

4. Gli aggiornamenti degli stati del veicolo e del punto di riferimento e la matrice di covarianza associata dopo l'osservazione di un punto di riferimento i, non sono influenzati dalla rimozione dello stato corrispondente a qualsiasi punto di riferimento j e dalle corrispondenti righe e colonne della matrice di covarianza dello stato. Perciò qualsiasi punto di riferimento che attualmente non viene osservato può essere spostato nuovamente dalla mappa senza influenzare la consistenza statistica del processo di costruzione della mappa.

5. Quando un punto di riferimento rimosso viene osservato di nuovo, la posizione del veicolo e gli stati degli altri punti di riferimento non possono essere aggiornati usando lo stato e le covarianze associate del punto di riferimento nell'istante in cui è stato rimosso, poiché questi non sono più validi, dato che gli aggiornamenti menzionati nei passi 1 e 2 non vengono effettuati dopo la cancellazione. Il punto di riferimento deve essere reinizializzato [32].

2.2.2 FastSLAM

É stato introdotto un approccio alternativo al problema SLAM che lo fattorizza in un problema di localizzazione e K problemi di stima dei punti di riferimento indipendenti, condizionati dalla stima della posa del robot. Questo algoritmo chiamato FastSLAM, usa una modifica del filtro particellare per stimare a posteriori il percorso del robot. Ogni particella possiede K filtri di Kalman indipendenti che stimano le posizioni dei punti di riferimento condizionate dal percorso della particella. L'algoritmo risultante, è un'istanza del filtro particellare $Rao-Blackwellized$ [49], rappresentando le particelle come alberi binari di filtri di Kalman, le osservazioni possono essere incorporate in FastSLAM con una complessità temporale $O(MlogK)$, dove M rappresenta il numero di particelle e K il numero di landmarks. Inoltre questo algoritmo è stato dimostrato funzionare fino a 100.000 punti di riferimento, problemi ben

oltre la portata del filtro di Kalman esteso. Dato che ogni particella rappresenta una diversa ipotesi di posa del robot, l'associazione dei dati può essere considerata separatamente per ognuna di esse, portando a due vantaggi: il primo è che il rumore del movimento del robot non influenza la precisione dell'associazione dei dati mentre il secondo, se le osservazioni sono associate correttamente ad alcune particelle ed altre erroneamente ad altre, quest'ultime riceveranno una probabilità più basse e saranno rimosse nei futuri passi di ricampionamento. In questo modo FastSLAM può "dimenticare" le associazioni del passato quando altre associazioni corrette giustificano meglio i dati.

FastSLAM con associazioni dati nota

La Figura (2.4) come già spiegato, illustra un modello probabilistico (rete dinamica di Bayes) che descrive il problema SLAM. Da questo diagramma è chiaro che il problema SLAM contiene importanti indipendenze condizionali. In particolare, la conoscenza del percorso del robot $s_i, ..., s_t$ rende indipendenti le misure dei singoli punti di riferimento. Così, per esempio, se un oracolo ci fornisse il percorso esatto del robot, il problema di determinare la posizione dei punti di riferimento potrebbe essere disaccoppiato in K problemi di stima indipendenti, uno per ogni punto di riferimento. Questa indipendenza condizionale è la base dell'algoritmo FastSLAM. Questa indipendenza condizionale implica che la probabilità a posteriori $p(s^t, \theta | z^t, u^t, n^t)$ può essere fattorizzata come segue, in un posteriore del percorso del robot e un prodotto delle probabilità a posteriori dei singoli punti di riferimento condizionati sul percorso del robot:

$$p(s^t, \theta | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_{i=1}^{K} p(\theta_i | s^t, z^t, u^t, n^t). \tag{2.37}$$

FastSLAM usa un filtro a particelle modificato, con i filtri di Kalman indipendenti per ogni particella.

Stima del percorso con filtro particellare

FastSLAM stima la probabilità a posteriori del percorso del robot nell'equazione (2.4) usando un filtro a particelle, in un modo che è simile (ma non identico) all'algoritmo di localizzazione Monte Carlo Localization (MCL). Ad ogni istante di tempo, FastSLAM mantiene un insieme di particelle che rappresentano $p(s^t | z^t, u^t, n^t)$, denotato S_t. Ogni particella $s^{t,[m]}$ che appartiene a S_t, rappresenta una "ipotesi" del percorso del robot:

$$S_t = s_m^{t,[m]} = s_1^{[m]}, s_2^{[m]}, .., s_t^{[m]} \tag{2.38}$$

si usa la notazione $^{[m]}$ che indica la m-esima particella nel campione.
L'insieme delle particelle S_t è calcolato in modo incrementale dall'insieme S_{t-1} al tempo $t-1$, dalle azioni di controllo u_t e dalle misurazioni z_t. Primo, per ogni particella $s^{t,[m]}$ in S_{t-1} è usata per generare un'ipotesi probabilistica della posa del robot al tempo t:

$$s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]}). \tag{2.39}$$

Questa ipotesi è ottenuta per campionamento del modello di movimento probabilistico. Questa stima viene poi aggiunta ad un insieme temporale di particelle, insieme al percorso $s^{t-1,[m]}$. Con l' assunzione che il set di particelle in S_{t-1} è distribuito secondo la legge $p(s^{t-1}|z^{t-1}, u^{t-1}, n^{t-1})$ (la quale è un'approssimazione asintoticamente corretta), la nuova particella è distribuita secondo:

$$p(s^t|z^{t-1}, u^t, n^{t-1}). \tag{2.40}$$

Questa distribuzione è comunemente indicata come la distribuzione proposta di filtraggio delle particelle. Dopo aver generato M particelle in questo modo, il nuovo insieme S_t è ottenuto per campionamento dal temporaneo insieme di particelle. Ogni particella $s^{t,[m]}$ è estratta (con sostituzione) con una probabilità proporzionale al cosiddetto fattore di importanza $\omega_t^{[m]}$, calcolato come segue:

$$\omega_t^{[m]} = \frac{distribuzioneobiettivo}{distribuzioneproposta} = \frac{p(s^{t,[m]}|z^t, u^t, n^t)}{p(s^{t,[m]}|z^{t-1}, u^t, n^{t-1})}. \tag{2.41}$$

I pesi vengono calcolati:

$$\omega_t^{[m]} \propto \frac{p(s^{t,[m]}|z^t, u^t, n^t)}{p}(s^{t,[m]}|z^{t-1}, u^t, n^{t-1}) = \int p(z_t|\theta_{n_t}^{[m]}, s_t^{[m]})p(\theta_{n_t}^{[m]})d\theta_{n_t} \tag{2.42}$$

Questa quantità può essere calcolata in forma chiusa perché gli stimatori del punto di riferimento sono filtri di Kalman. L'insieme di campioni risultante S_t è distribuito secondo un'approssimazione al percorso desiderato $p(s^t|z^t, u^t, n^t)$, un'approssimazione che è corretta quando il numero di particelle M va all'infinito. Notiamo anche che solo la più recente stima della posa del robot $s^{[m]_{t-1}}$ è usata quando si genera il set di particelle S_t. Questo permette di "dimenticare" tutte le altre stime di posa, rendendo la misura di ogni particella indipendente dall'indice temporale t.

Stima della posizione dei landmark

FastSLAM rappresenta la stima condizionata dei punti di riferimento $p(\theta_i|s^t, z^t, u^t, n^t)$ in (2.37) usando dei filtri di Kalman. Poiché questa stima è condizionata dalla posa del robot, i filtri di Kalman sono collegati alle singole particelle di posa in S_t. Più specificamente, la probabilità a posteriori completa sui percorsi e le posizioni dei punti di riferimento nell'algoritmo FastSLAM è rappresentata dall'insieme di campioni

$$S_t = s^{t,[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, ..., \mu_K^{[m]}, \Sigma_K^{[m]}. \tag{2.43}$$

Qui, $\mu_i^{[m]}$ e $\Sigma_i^{[m]}$ sono rispettivamente la media e la covarianza della Gaussiana che rappresenta l'i-esimo punto di riferimento θ_i, collegato alla m-esima particella. Nello scenario di navigazione del robot, ogni media $\mu_i^{[m]}$ è un vettore di due elementi e ogni covarianza $\Sigma_i^{[m]}$ è una matrice 2x2. Il calcolo della probabilità a posteriori della posizione dell'i-esimo riferimento dipende dal fatto che il punto di riferimento sia stato osservato o meno. Se il punto di riferimento è osservato, si ottiene:

$$p(\theta_{i=n_t}|s^t, z^t, u^t, n^t) \propto p(z_t|\theta_{nt}, s_t, n_t)p(\theta_{nt}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}). \tag{2.44}$$

Se i punti di riferimento non sono osservati, viene lasciata semplicemente la Gaussiana invariata:

$$p(\theta_{i\neq nt}|s^t, z^t, u^t, n^t) = p(\theta_i|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}). \tag{2.45}$$

L'algoritmo FastSLAM implementa l'equazione di aggiornamento (2.44) usando il filtro di Kalman esteso (EKF). Come negli approcci EKF esistenti per lo SLAM, questo filtro usa una versione linearizzata del modello percettivo $p(z_t|s_t, \theta_{nt}, n_t)$. Una differenza significativa tra l'uso di filtri di Kalman da parte di FastSLAM e quello dell'algoritmo dello SLAM tradizionale è che gli aggiornamenti nell'algoritmo FastSLAM coinvolgono solo una Gaussiana di dimensione due (per i due parametri di localizzazione dei punti di riferimento). Nell'approccio SLAM basato su EKF deve essere aggiornata una gaussiana di dimensione $2K + 3$ (con K punti di riferimento e 3 parametri di posa del robot). Questo calcolo può essere fatto in tempo costante in FastSLAM, mentre richiede un tempo quadratico in K nel metodo EKF [49].

FastSLAM con dati sconosciuti associati

Se la corrispondenza tra osservazioni e punti di riferimento è nota, l'algoritmo FastSLAM effettua il campionamento sui percorsi del robot e calcola

le stime condizionali dei punti di riferimento in modo analitico per ogni campione di percorso. Quando questa corrispondenza non è nota, FastSLAM può essere esteso per campionare le associazioni di dati possibili e i percorsi dei robot. Ci sono diversi modi in cui questo campionamento può essere fatto.

L'approccio più semplice al campionamento sulle associazioni di dati è quello di adottare la procedura di assegnazione della massima verosimiglianza usata da EKF, ma su una base per particella. Le particelle che scelgono la corretta associazione di dati riceveranno alte probabilità perché spiegano bene le misurazioni. Le particelle che assegnano le osservazioni in modo errato riceveranno probabilità più basse e saranno rimosse nei futuri passi di ricampionamento. Questa procedura può essere scritta come:

$$n_t^{[m]} = \underset{n}{\operatorname{argmax}}\, tp(z_t|s_t^{[m]}, \theta_n t, n_t). \tag{2.46}$$

L'associazione di dati per particella ha due chiare conseguenze. In primo luogo, il rumore di movimento del robot non influenza l'efficacia dell'associazione dei dati, dato un numero appropriato di particelle. Questo fatto da solo si traduce in prestazioni significativamente migliorate in situazioni di sostanziale ambiguità di movimento. Se applicato allo scenario mostrato in Figura (2.6), alcune delle particelle rappresenteranno la posa a sinistra e sceglieranno la prima ipotesi di associazione dei dati, mentre altre parti sceglieranno la seconda ipotesi. La seconda conseguenza dell'associazione dei dati per particella è un processo decisionale integrato e ritardato. In qualsiasi momento, una frazione delle particelle riceverà associazioni di dati plausibili, ma sbagliate. In futuro, potrebbero essere ricevute nuove osservazioni che confutano chiaramente queste assegnazioni precedenti. Queste particelle riceveranno una bassa probabilità e saranno rimosse dal filtro. Di conseguenza, l'effetto delle associazioni sbagliate fatte in passato può essere rimosso dal filtro in un secondo momento.

Questo è in netto contrasto con l'EKF, in cui l'effetto di un'associazione errata di dati non può mai essere rimosso una volta incorporato. Inoltre, nessuna euristica è necessaria per gestire la rimozione delle vecchie associazioni. Questo viene fatto in modo statisticamente valido, semplicemente come una sequenza del passo di ricampionamento del filtro a particelle. Naturalmente, il campionamento dei percorsi dei robot e delle associazioni di dati richiederà più particelle che il campionamento dei soli percorsi dei robot [48].

2.2.3 RGB-D SLAM

Uno dei più recenti sviluppi nella tecnologia di rilevamento [51] è arrivato sotto forma di telecamere a luce strutturata; Microsoft Kinect e ASUS

Xtion Pro sono entrambi esempi di sensori basati sulla tecnologia PrimeSense. Questi sensori sono stati classificati come telecamere RGB-D, per il fatto che forniscono due tipi di immagini: un'immagine primaria a colori basata su RGB e un'immagine secondaria di profondità (dove ogni pixel è rappresentativo di una misura di distanza). Possono essere ulteriormente classificati come sensori attivi, poiché il sensore deve emettere costantemente una struttura di luce proiettata.

I recenti approcci allo SLAM RGB-D si sono concentrati esclusivamente sull'uso della profondità grezza o su una combinazione di profondità e informazioni RGB per prevedere l'ego-motion della telecamera. Per ego-motion si intende il movimento 3D di una telecamera all'interno di un ambiente. Nel campo della computer vision, l'ego-motion si riferisce alla stima del movimento di una telecamera rispetto a una scena in cui gli elementi che la compongono sono fermi [52]. Un esempio di stima dell'ego-motion potrebbe essere la stima della posizione di un'auto in movimento rispetto alle linee sulla strada o ai segnali stradali osservati dall'auto stessa [53]. La stima dell'ego-motion è importante nelle applicazioni di navigazione autonoma dei robot. Le tecniche del tipo di combinazione hanno mostrato generalmente più robustezza poiché prendono sia le informazioni visive che quelle geometriche nella stima del movimento. Tuttavia, quando si ha a che fare con ambienti interni complessi, possono verificarsi due scenari che fanno fallire l'algoritmo tradizionale di tipo RGB-D. Si tratta di grandi scene spaziali in cui si ottengono poche o nessuna informazione sulla profondità e di scene planari, in cui la struttura geometrica è molto carente.

Gli algoritmi di mappatura che utilizzano le informazioni RGB-D sono stati un campo di ricerca attivo per molto tempo, riportato alla ribalta dopo i recenti progressi della tecnologia RGB-D, principalmente la telecamera a luce strutturata PrimeSense. Un lavoro significativo è stato dedicato all'utilizzo di questi sensori per affrontare il problema dello SLAM utilizzando sia approcci basati sulle caratteristiche che sull'aspetto. Il lavoro pionieristico fatto da Nister [54] ha combinato l'estrazione delle caratteristiche e la corrispondenza con RANSAC per ottenere una stima accurata della posa usando le telecamere stereo. *Izadi et al.* [55] hanno affrontato SLAM dividendo il tracking e la mappatura in problemi separati. In questo lavoro la telecamera viene tracciata su un modello globale dell'ambiente attraverso un algoritmo ICP (Iterative Closed Points). Il modello dell'ambiente viene reso più denso e raffinato nel corso di diverse osservazioni in modo da migliorare il tracciamento. Sfortunatamente l'ICP puro si basa su ambienti molto strutturati, limitando così questa tecnica solo a uno scenario di tipo stanza e non a un caso generale di interni. Inoltre, tenere traccia di un modello denso può essere computazionalmente costoso e non si scala molto bene. *Henry et al.* [56]

propongono un metodo per ottimizzare congiuntamente le caratteristiche visive basato su RANSAC con ICP. Questo approccio è di natura semplicistica ma in grado di funzionare in modo molto intuitivo con le telecamere RGB-D. Durante l'ottimizzazione, la regolazione del bundle viene applicata per ottenere mappe coerenti. *Audras et al.* [57] si sono opposti al metodo basato sulle caratteristiche a causa della loro natura incerta nella corrispondenza e nel rilevamento, introducendo un approccio basato sull'aspetto. La tecnica di tracciamento stereo denso, convertita per lavorare con la telecamera RGB-D, predice il movimento relativo della telecamera minimizzando una funzione di deformazione tra i fotogrammi data l'informazione della nuvola di punti 3D.

Affinché tutti questi metodi funzionino con successo nel caso generale degli interni, devono essere fornite sufficienti informazioni sulla profondità. Il pacchetto software di ROS più utilizzato per l'implementazione del RGB-D Slam è 'RTAB-Map' distribuito come libreria open source dal 2013, iniziato come un approccio di rilevamento della loop closure basato sull'aspetto con gestione della memoria per affrontare operazioni online su larga scala e a lungo termine [58]. Prima di illustrare il pacchetto utilizzato, verrà mostrato l'algoritmo sul quale esso si basa: Graph SLAM.

Graph SLAM

Graph SLAM è un algoritmo SLAM che risolve il problema SLAM completo, come mostrato in [59]. Ciò significa che l'algoritmo recupera l'intero percorso e la mappa, invece che solo la posa e la mappa recenti. Questa differenza gli permette di considerare le dipendenze tra le pose attuali e precedenti. Uno dei vantaggi del graph SLAM è la ridotta necessità di una significativa capacità di elaborazione a bordo. Un altro è la maggiore accuratezza del graph SLAM rispetto al fast SLAM. Il Fast SLAM usa le particelle per stimare la posizione più probabile del robot. Tuttavia, in qualsiasi momento, è possibile che non ci sia una particella nella posizione più probabile. Infatti, le possibilità sono minime o nulle, specialmente in ambienti grandi. Poiché il graph SLAM risolve il problema SLAM completo, ciò significa che può lavorare con tutti i dati in una volta sola per trovare la soluzione ottimale.

Nel graph SLAM, l'idea è di organizzare le informazioni in un grafo. Un nodo nel grafo rappresenta o una posa del robot x_t ad un passo temporale specifico t o la posizione di una caratteristica nell'ambiente denotata come $m^{(i)}$ con $i = 1, \dots$. Un arco nel grafo rappresenta o un vincolo di misura tra una posa e una caratteristica o un vincolo di movimento tra due pose successive. Poiché i vincoli spaziali sono soft, possono essere considerati come molle che collegano due masse. In questa analogia, il problema SLAM completo può essere risolto come un problema di ottimizzazione globale del

grafo. La configurazione ottimale del grafo è quella in cui le molle sono rilassate, e le forze su ciascuno dei nodi sono minimizzate. Il principio di massima verosimiglianza (MLE) è usato per ottimizzare il grafo. Quando viene applicato allo SLAM, la verosimiglianza cerca di stimare la configurazione più probabile delle posizioni di stato e delle caratteristiche date le osservazioni di movimento e di misurazione. L'aggiornamento delle misure al passo temporale t è dato da:

$$\overline{z}^t := x_t + m_t^{(i)} \tag{2.47}$$

che rappresenta per esempio un laser range finder che misura la distanza dal punto di riferimento $m^{(i)}$. Equivalentemente, un aggiornamento del movimento può essere definito come:

$$\overline{x}^t := x_{t-1} + u_t \tag{2.48}$$

che potrebbe essere realizzato come un comando di controllo che istruisce il robot a muoversi di una certa distanza u_t . Si suppone che l'aggiornamento abbia un rumore gaussiano. Le distribuzioni di probabilità corrispondenti sono date da

$$p_u(x_t) = \frac{1}{\sigma_m\sqrt{2\pi}} e^{-(z_t-\overline{z}_t)^2/2\sigma_m^2} \tag{2.49}$$

$$p_m(z_t) = \frac{1}{\sigma_u\sqrt{2\pi}} e^{-(x_t-\overline{x}_t)^2/2\sigma_u^2} \tag{2.50}$$

.

In alcuni casi semplici è possibile trovare una soluzione analitica di MLE convertendo la funzione obiettivo nella seguente forma:

$$J_{GraphSLAM} = \sum_t \left(\frac{z_t - \overline{z}_t}{\sigma_m}\right)^2 + \sum_t \left(\frac{x_t - \overline{x}_t}{\sigma_u}\right)^2 \tag{2.51}$$

cercando di minimizzare la somma di tutti i vincoli. In scenari realistici più complessi, sono necessarie soluzioni numeriche approssimate, per esempio applicando tecniche di discesa del gradiente. Nel mondo reale, la maggior parte dei sistemi sono multidimensionali e, per affrontare tali scenari, è necessario utilizzare matrici e covarianze. Lo stato e la misura sono dati da x_t e z_t. I vincoli sono dati da:

$$v_t := z_t - h(x_t, m_t) \tag{2.52}$$

$$w_t := x_t - g(x_{t-1}, u_t) \tag{2.53}$$

dove $h(\cdot)$ e $g(\cdot)$ rappresentano le funzioni di misura e di movimento e Q_t e R_t sono le covarianze del rumore di misura e di movimento. La formula multidimensionale per la somma di tutti i vincoli è data da:

$$J_{GraphSLAM} = x_0^T \Omega x_0 + \sum_t (w_t^T * R_t^{(-1)} * w_t + v_t^T * Q_t^{(-1)} * v_t) \qquad (2.54)$$

L'obiettivo del graph SLAM è quello di creare un grafo di tutte le pose del robot e delle caratteristiche incontrate nell'ambiente e il percorso più probabile del robot e la mappa dell'ambiente. Questo compito può essere suddiviso in due parti: il *front-end* e il *back-end*. Il front-end del grafo SLAM riguarda la costruzione del grafo, usando le misure odometriche e sensoriali raccolte dal robot. Questo include l'interpretazione dei dati sensoriali, la creazione del grafo, e la continuazione dell'aggiunta di nodi e bordi ad esso mentre il robot attraversa l'ambiente. Naturalmente il front-end può differire notevolmente da un'applicazione all'altra a seconda dell'obiettivo desiderato, compresa la precisione, il sensore usato e altri fattori, ad esempio il front-end di un robot mobile che applica lo SLAM in ufficio usando un laser range finder sarebbe molto diverso dal front-end di un veicolo che opera in un grande ambiente esterno e usa una telecamera stereo. Il front-end del grafo SLAM ha anche la sfida di risolvere il problema dell'associazione dei dati. In termini più semplici, questo significa identificare accuratamente se le caratteristiche nell'ambiente sono state viste in precedenza.

Il back-end del graph SLAM è la parte cruciale dell'algoritmo. L'input per il back-end è il grafo completato con tutti i vincoli e l'output è la configurazione più probabile delle pose del robot e delle caratteristiche della mappa. Il back-end è un processo di ottimizzazione che prende tutti i vincoli e trova la configurazione del sistema che produce il minor errore. Il back-end è molto più coerente tra le varie applicazioni. Il front-end e il back-end possono essere completati in successione o possono essere eseguiti iterativamente, con il back-end che fornisce un grafo aggiornato al front-end per un'ulteriore elaborazione.

RTAB-Map

Tra il 2009 e il 2013 *Labbé e Michaud* distribuirono una libreria open source, RTAB-Map [58] che implementa una loop closure detection con gestione della memoria. Da allora è stata estesa implementando un approccio SLAM completo basato su grafi (lavoro di *Stachniss et al.* [60]) per essere utilizzato in varie configurazioni e applicazioni, come proposto nei lavori di

Laniel et al. [61], *Foresti et al.* [62], e *Chen et al.* [63]. Di conseguenza, RTAB-Map si è evoluto in una libreria standalone C++ multipiattaforma e in un pacchetto ROS2, guidato da esigenze pratiche quali:

- Elaborazione online: l'output del modulo SLAM dovrebbe essere limitato a un ritardo massimo dopo la ricezione dei dati del sensore. Per lo SLAM basato sul grafo in particolare, man mano che la mappa cresce, è necessario più tempo di elaborazione per rilevare le chiusure dei cicli, per ottimizzare il grafo e per assemblare la mappa. Inoltre, l'integrazione con altri moduli di elaborazione per il controllo, la navigazione, per evitare degli ostacoli, l'interazione con l'utente, il riconoscimento degli oggetti, ecc. può anche limitare il tempo di CPU disponibile per SLAM. Avere la possibilità di limitare il carico di calcolo è quindi vantaggioso per evitare problemi di ritardo con altri moduli, e può anche essere necessario per prevenire situazioni non sicure.

- Odometria robusta e a bassa deriva: mentre il rilevamento della chiusura del loop può correggere la maggior parte della deriva dell'odometria, negli scenari del mondo reale il robot spesso non può localizzarsi correttamente sulla mappa, o perché sta esplorando nuove aree o perché mancano caratteristiche discriminanti nell'ambiente. Durante questo periodo, la deriva dell'odometria dovrebbe essere minimizzata in modo che la navigazione autonoma accurata sia ancora possibile fino a quando la localizzazione può avvenire, per evitare di sovrascrivere erroneamente le aree mappate (ad esempio, aggiungendo erroneamente ostacoli all'ingresso di una stanza, rendendola un'area chiusa per esempio). Stimare l'odometria con sensori esterocettivi come telecamere e lidar può essere molto accurato quando ci sono abbastanza caratteristiche nell'ambiente, ma usare solo una modalità di rilevamento può essere problematico e incline a fallire la localizzazione se le caratteristiche tracciate nell'ambiente non sono più visibili. L'utilizzo di un mix di sensori propriocettivi (ad esempio, encoder delle ruote, unità di misura inerziali (IMU)) ed esteriocettivi aumenterebbe la robustezza della stima dell'odometria.

- Localizzazione robusta: l'approccio SLAM deve essere in grado di riconoscere quando sta rivisitando le posizioni passate (per il rilevamento della chiusura del loop) per correggere la mappa. Ambienti dinamici, cambiamenti di illuminazione, cambiamenti di geometria o anche ambienti ripetitivi possono portare a una localizzazione errata o alla mancata localizzazione, e quindi l'approccio deve essere robusto ai falsi positivi.

- Generazione e sfruttamento pratico della mappa: la maggior parte degli approcci di navigazione popolari sono basati sulla griglia di occupazione, e quindi è vantaggioso sviluppare approcci SLAM che possano fornire una griglia di occupazione 3D o 2D per una facile integrazione. Inoltre, quando l'ambiente è per lo più statico, è più pratico fare una sessione di mappatura e poi passare alla localizzazione, impostando l'uso della memoria e risparmiando tempo di gestione della mappa.

- Mappatura multisessione (in letteratura chiamato anche problema del robot rapito o problema dello stato iniziale): quando è attivata, un robot non conosce la sua posizione relativa ad una mappa precedentemente creata, rendendo impossibile pianificare un percorso verso una posizione precedentemente visitata. Per evitare che il robot riavvii il processo di mappatura per azzerarsi o localizzarsi in una mappa precedentemente costruita prima di iniziare la mappatura, la mappatura multisessione permette all'approccio SLAM di inizializzare una nuova mappa con il proprio referenziale all'avvio, e quando una posizione precedentemente visitata viene incontrata, una trasformazione tra le due mappe può essere calcolata. Questo porta i vantaggi di evitare di rimappare l'intero ambiente quando solo una piccola parte deve essere rimappata o una nuova area deve essere aggiunta.

Essendo RTAB-Map un approccio di chiusura dei cicli con la gestione della memoria come suo nucleo, è indipendente dall'approccio odometrico utilizzato, il che significa che può essere alimentato con odometria visiva, odometria lidar o anche solo odometria delle ruote. Il suo nodo ROS principale è chiamato 'rtabmap'. L'odometria è un input esterno a RTAB-Map, il che significa che SLAM può anche essere fatto utilizzando qualsiasi tipo di odometria per utilizzare ciò che è appropriato per una data applicazione e robot. La struttura della mappa è un grafo con nodi e collegamenti. Dopo la sincronizzazione dei sensori, il modulo Short-Term Memory (STM) crea un nodo che memorizza la posa dell'odometria, i dati grezzi del sensore e le informazioni aggiuntive utili per i moduli successivi (ad esempio,termini visivi per Loop Closure e Proximity Detection, e la griglia di occupazione locale per Global Map Assembling). I nodi sono creati ad un tasso fisso "Rtabmap/DetectionRate" impostato in millisecondi secondo quanto i dati creati dai nodi dovrebbero sovrapporsi l'un l'altro. Per esempio, se il robot si muove velocemente e la portata del sensore è piccola, il tasso di rilevamento dovrebbe essere aumentato per assicurarsi che i dati dei nodi successivi si sovrappongano, ma impostarlo troppo alto aumenterebbe inutilmente l'uso della memoria e il tempo di calcolo. Un link contiene una trasformazione

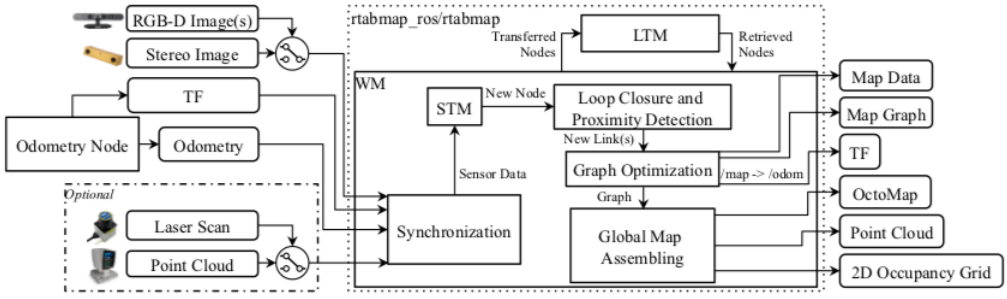


Figura 2.8: Schema a blocchi del nodo ROS rtabmap. Gli input richiesti sono: TF per definire la posizione dei sensori in relazione alla base del robot; Odometria da qualsiasi fonte (che può essere 3DoF o 6DoF); uno degli ingressi della telecamera (una o più immagini RGB-D, o un'immagine stereo) con messaggi di calibrazione corrispondenti. Gli ingressi opzionali sono una scansione laser da un lidar 2D o una nuvola di punti da un lidar 3D. Tutti i messaggi da questi ingressi sono poi sincronizzati e passati all'algoritmo graph-SLAM. Le uscite sono: Map Data contenente l'ultimo nodo aggiunto con i dati dei sensori compressi e il grafo; Map Graph senza alcun dato; correzione dell'odometria pubblicata su TF; una OctoMap opzionale (griglia di occupazione 3D); una Point Cloud densa opzionale; una griglia di occupazione 2D opzionale. Immagine da [58]

rigida tra due nodi. Ci sono tre tipi di collegamenti: Neighbor, Loop Closure e Proximity links. I Neighbor links sono aggiunti nel STM tra nodi consecutivi con trasformazione odometrica. I collegamenti Loop Closure e Proximity sono aggiunti attraverso il rilevamento della chiusura del loop o il rilevamento della prossimità, rispettivamente. Tutti i collegamenti sono usati come vincoli per l'ottimizzazione del grafo. Quando c'è una nuova chiusura di loop o un collegamento di prossimità aggiunto al grafo, l'ottimizzazione del grafo propaga l'errore calcolato all'intero grafo, per diminuire la deriva dell'odometria. Con il grafo ottimizzato, le uscite OctoMap, Point Cloud e 2D Occupancy Grid possono essere assemblate e pubblicate su moduli esterni.

L'approccio di gestione della memoria "RTAB-Map" funziona su moduli di gestione del grafo. Viene utilizzato per limitare la dimensione del grafo in modo che lo SLAM online a lungo termine possa essere realizzato in ambienti di grandi dimensioni. Senza la gestione della memoria, man mano che il grafo cresce, il tempo di elaborazione per moduli come Loop Closure e Proximity Detection, Graph Optimization e Global Map Assembling può alla fine superare i vincoli del tempo reale, cioè il tempo di elaborazione può

diventare maggiore del tempo del ciclo di acquisizione del nodo. Fondamentalmente, la memoria di RTAB-Map è divisa in una Working Memory (WM) e una Long-Term Memory (LTM). Quando un nodo viene trasferito nella LTM, non è più disponibile per i moduli all'interno della WM. Quando il tempo di aggiornamento della RTAB-Map supera la soglia temporale fissa "Rtabmap/TimeThr", alcuni nodi nella WM vengono trasferiti nella LTM per limitare la dimensione della WM e diminuire il tempo di aggiornamento.

Analogamente alla soglia di tempo fissa, esiste anche una soglia di memoria "Rtabmap/MemoryThr" che può essere utilizzata per impostare il numero massimo di nodi che WM può contenere. Per determinare quali nodi trasferire in LTM, un meccanismo di ponderazione identifica le posizioni che sono più importanti di altre, utilizzando euristiche come il fatto che più a lungo una posizione è stata osservata, più è importante e quindi dovrebbe essere lasciata nel WM. Per fare ciò, quando si crea un nuovo nodo, STM inizializza il peso del nodo a 0 e lo confronta visivamente (ricavando una percentuale di termini visivi corrispondenti) con l'ultimo nodo del grafo. Se sono simili (con la percentuale di termini visivi corrispondenti superiore alla soglia di somiglianza "Mem/RehearsalSimilarity"), il peso del nuovo nodo viene aumentato di uno più il peso dell'ultimo nodo. Il peso dell'ultimo nodo viene riportato a 0, e l'ultimo nodo viene scartato se il robot non si sta muovendo per evitare di aumentare inutilmente la dimensione del grafo. Quando si raggiungono le soglie di tempo o di memoria, i nodi più vecchi e con il peso minore vengono trasferiti per primi in LTM. Quando si verifica una chiusura del loop con una posizione nel WM, i nodi vicini di questa posizione possono essere riportati dal LTM al WM per ulteriori chiusure di loop e rilevamenti di prossimità. Quando il robot si muove in un'area precedentemente visitata, può quindi ricordare le posizioni passate in modo incrementale per estendere la mappa corrente assemblata e localizzare usando le posizioni passate globale.

Loop Closure Detection

L'approccio 'Real-time appearance-based mapping' (Rtab-Map) valuta il numero di volte che una posizione è stata abbinata o vista consecutivamente (indicato come *Rehearsal*) per impostare il suo peso, rendendo possibile identificare le posizioni viste più frequentemente di altre e che hanno più probabilità di causare i rilevamenti di chiusura. Quando avviene il trasferimento, viene selezionata la posizione con il peso più basso, mentre se ci sono posizioni con lo stesso peso, quella più vecchia viene selezionata per essere trasferita. Le posizioni nella memoria a breve termine (STM) non sono utilizzate per il rilevamento della chiusura del ciclo, per evitare il rilevamento della chiusura del ciclo su posizioni che sono state appena visitate (l'ultima

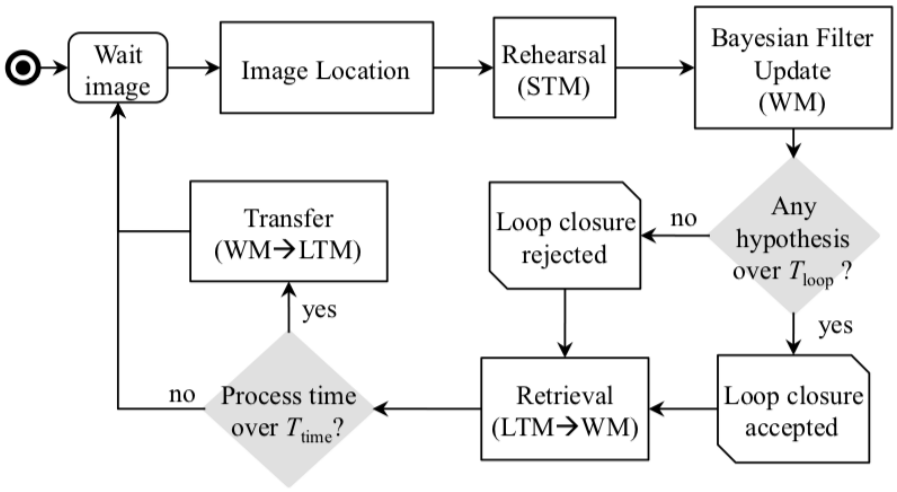


Figura 2.9: Diagramma di flusso del ciclo di elaborazione del rilevamento della chiusura del ciclo di gestione della memoria. Immagine da [64]

posizione è sempre simile a quelle più recenti). La dimensione STM è fissata in base alla velocità del robot e alla velocità di acquisizione delle posizioni: quando il numero di posizioni raggiunge il limite della dimensione STM, la posizione più vecchia in STM viene spostata in WM. Come mostrato in Figura (2.9).

Posizione dell'immagine L'approccio 'bag-of-words' è usato per creare una firma z_t di un'immagine acquisita al tempo t. Una firma dell'immagine è rappresentata da un insieme di termini visivi contenuti in un dizionario visivo costruito online in modo incrementale usando una foresta randomizzata di alberi k-dimensionale. Le Speeded-Up Robust Features (SURF) sono estratte dall'immagine come termini visivi. Una posizione L_t viene quindi creata con la firma z_t, un peso inizializzato a 0 e un collegamento bidirezionale nel grafo con L_{t-1}. Si noti che il dizionario contiene solo parole di località in WM e STM.

Rehearsal Per aggiornare il peso della posizione acquisita, L_t viene confrontata con quelle in STM dalle recenti alle più recenti, e la similarità s viene valutata usando:

$$s(z_t, z_c) = \begin{cases} N_{pair}/N_{z_t}, if N_{z_t} \geq N_{z_c} \\ N_{pair}/N_{z_c}/N_{z_c} if N_{z_t} < N_{z_c} \end{cases} \tag{2.55}$$

dove N_{pair} è il numero delle coppie di parole corrispondenti tra le firme di località confrontate, e N_{z_t} e N_{z_c} sono il numero totale di parole della firma z_t e la firma confrontata z_c rispettivamente. Se $s(z_t, z_c)$ è maggiore di una

soglia fissa di similarità $T_{rehearsal}$, L_c viene fusa in L_t e Rehearsal si ferma. In Rehearsal, solo le parole di z_c sono mantenute nella firma fusa: z_t è cancellato e z_c è copiato in z_t. Per completare il processo di fusione, il peso di L_t viene aumentato di quello di L_c più uno, i link vicini di L_c vengono aggiunti a L_t e L_c viene cancellato da STM.

Aggiornamento del filtro di Bayes Il ruolo del filtro bayesiano discreto è quello di tenere traccia delle ipotesi di chiusura del ciclo, stimando la probabilità che la posizione corrente L_t corrisponda a una posizione già visitata memorizzata nel WM. Sia S_t una variabile casuale che rappresenta gli stati di tutte le ipotesi di chiusura del loop al tempo t. $S_t = i$ è la probabilità che L_t chiuda un loop con una posizione L_i passata, rilevando così che L_t e L_i rappresentano la stessa posizione. $S_t = -1$ è la probabilità che L_t sia una nuova posizione. Il filtro stima l'intera probabilità a posteriori $p(S_t|L_t)$ per tutti gli $i = -1, ..., t_n$, dove t_n è l'indice temporale associato alla posizione più recente nella WM, espressa come segue:

$$p(S_t|l^t) = \underbrace{\eta P(L_t|S_t)}_{Osservazioni} \underbrace{\sum_{i=-1} t_n \underbrace{(S_t|S_{t-1} = i)}_{Transizione} p(S_{t-1} = i|L^{t-1})}_{Pos.presuntaBelief} \tag{2.56}$$

dove η è la costante di normalizzazione e $L^t = L_{-1}, .., L_t$. Si noti che la sequenza di luoghi L_t include solo i luoghi contenuti nel WM e STM, quindi cambia nel tempo quando vengono creati nuovi luoghi e quando alcuni luoghi vengono recuperati dal LTM o trasferiti al LTM, deviando dal classico filtraggio bayesiano dove tali sequenze sono fisse. Il modello di osservazione $p(L_t|S_t)$ è valutato usando una funzione di verosimiglianza $L(S_t|L_t)$: la posizione corrente L_t è confrontata utilizzando la (2.55) con le posizioni corrispondenti ad ogni stato di chiusura del ciclo $S_t = j$ dove $j = 0, .., t_n$, dando un punteggio $s_j = s(z_t, z_j)$. Ogni punteggio viene poi normalizzato utilizzando la differenza tra il punteggio s_j e la deviazione standard σ, normalizzata dalla media μ di tutti i punteggi non nulli, come in:

$$p(L_t|S_t = -1) = L(S_t = -1|L_t)0\frac{\mu}{\sigma} + 1 \tag{2.57}$$

dove il punteggio è relativo al rapporto μ su σ. Se $L(S_t = -1|L_t)$ è altro, significa che L_t non è simile a un particolare in WM (quando $\mu < \sigma$), allora L_t è più probabile che sia una nuova posizione. Il modello di transizione $p(S_t|S_{t-1} = i)$ è usato per predire la distribuzione di S_t, dato ogni stato della distribuzione S_{t-1} secondo il movimento del robot tra t e $(t-1)$. Combinato con $p(S_{t-1} = i|L^{t-1})$ (cioè la parte ricorsiva del filtro) questo costituisce la probabilità della prossima chiusura del ciclo.

Il modello di transizione è espresso:

- $p(S_t = -1|S_{t-1} = -1) = 0,9$, la probabilità di un nuovo evento di localizzazione al tempo t dato che nessuna chiusura del ciclo è avvenuta al tempo $t-1$.

- $p(S_t = i|S_{t-1} = -1) = 0.1/N_{WM}$ con $i \in [0; t_n]$ la probabilità di un evento di chiusura del ciclo al tempo t, dato che nessuna chiusura del ciclo è avvenuta a $t-1$. NWM è il numero di posizioni in WM dell'iterazione corrente.

- $p(S_t = -1|S_{t-1} = j) = 0.1$ con $j \in [0; t_n]$ la probabilità di un nuovo evento di localizzazione avvenuta al tempo t dato che si è verificata una chiusura del ciclo al tempo $t-1$.

- $p(S_t = i|S_{t-1} = j)$ con $i, j \in [0; t_n]$ la probabilità di un evento di chiusura del ciclo al tempo t dato che una chiusura del ciclo si è verificata al tempo $t-1$ in una posizione vicina. La probabilità è definita come una curva gaussiana discretizzata centrata su j e dove i valori non sono nulli per esattamente otto vicini (da $i = j-4, ..., j+4$) e la loro somma è 0.9.

Selezione dell'ipotesi di chiusura del ciclo Quando $p(S_t|L^t)$ è stato aggiornato e normalizzato, viene selezionata l'ipotesi più alta di $p(S_t|L^t)$ maggiore della soglia di chiusura del loop T_{loop}. Poiché un'ipotesi di chiusura del loop è generalmente diffusa ai suoi vicini (le località vicine condividono alcun termini simili tra loro), per ogni probabilità $p(S_t = i|L_t)$ dove $i \geq 0$, le probabilità dei suoi vicini nell'intervallo definito dal modello di transizione $p(S_t = i|S_{t-1} = j)$ vengono sommate. Si noti che perché avvenga una selezione, un minimo di località T_{minHyp} deve essere in WM: quando il grafo è inizializzato, le prime località aggiunte a WM hanno automaticamente un'alta probabilità di chiusura del loop (dopo la normalizzazione con (2.56), quindi chiusure di loop errate potrebbero essere accettate se le ipotesi sono oltre T_{loop}. Se c'è un'ipotesi di chiusura del loop $p(S_t = i|L^t)$ dove $i \geq 0$ è sopra T_{loop}, l'ipotesi $S_t = i$ viene accettata, L_t viene fusa con la vecchia locazione L_i: il peso di L_t viene aumentato di quello di L_i più uno e i link vicini di L_i vengono aggiunti a L_t. Dopo aver aggiornato L_t, a differenza di Rehearsal, L_i non viene immediatamente cancellata dalla memoria. Per la costanza del filtro bayesiano con le prossime immagini acquisite, l'ipotesi di chiusura del loop associata S_i deve essere valutata fino a quando L_i non è più nel vicino dell'ipotesi di chiusura del loop più alta, dopo di che viene cancellata.

Recupero I vicini della locazione L_i in LTM sono trasferiti di nuovo in WM se $p(S_t = i|L^t)$ è la probabilità più alta. Poiché questo passo richiede

tempo, un massimo di due località vengono recuperate ad ogni iterazione. Il dizionario visivo viene aggiornato con i termini associati alle firme delle località recuperate. Le parole comuni delle firme recuperate esistono ancora nel dizionario, quindi viene aggiunto un semplice riferimento tra queste parole e le firme corrispondenti. Per i termini che non sono più presenti (perché sono stati rimossi dal dizionario quando le località sono state trasferite), vengono abbinate a quelle nel dizionario per trovare se parole più recenti rappresentano le stesse caratteristiche SURF. Questo passo è particolarmente importante perché le nuove parole aggiunte dalla nuova firma z_t possono essere identiche alle parole precedentemente trasferite. Per le parole corrispondenti, le vecchie parole sono sostituite da quelle nuove nelle firme recuperate. Tutti i riferimenti nel LTM vengono cambiati e le vecchie parole vengono definitivamente rimosse. Se alcune parole non sono ancora abbinate, vengono semplicemente aggiunte al dizionario.

Trasferimento Quando il tempo di elaborazione diventa maggiore di T_{time}, la posizione più vecchia con il peso minore viene trasferita da WM a LTM. T_{time} (T_{time} corrisponde quindi al tempo medio di elaborazione di un'immagine dell'algoritmo) deve essere impostato per permettere al robot di elaborare le immagini percepite in tempo reale. Un T_{time} più alto significa che più località (e implicitamente più termini) possono essere tenute nella WM, e più ipotesi di chiusura di loop possono essere tenute per rappresentare meglio l'ambiente complessivo. Il T_{time} deve quindi essere impostato in base alle capacità della CPU del robot, al carico di calcolo e all'ambiente operativo. Se T_{time} è impostato su un valore superiore al tempo di acquisizione dell'immagine, l'algoritmo utilizza intrinsecamente un tasso di immagine corrispondente a T_{time}, con un utilizzo della CPU del 100%. Come regola generale, T_{time} può essere impostato a circa 200-400 ms più piccolo del tasso di acquisizione dell'immagine a 1 Hz, per garantire che tutte le immagini siano elaborate sotto il tasso di acquisizione dell'immagine, anche se il tempo di elaborazione va oltre T_{time} (T_{time} corrisponde quindi al tempo medio di elaborazione di un'immagine dell'algoritmo). Così, per un tasso di acquisizione delle immagini di 1 Hz, T_{time} potrebbe essere impostato tra 600 ms e 800 ms. Poiché il passo più costoso del nostro algoritmo è quello di ricostruire il dizionario visivo, il tempo di elaborazione può essere regolato cambiando la dimensione del dizionario, che influenza indirettamente la dimensione del WM. Una firma di una località trasferita in LTM rimuove i riferimenti alle sue parole dal dizionario visivo. Se una parola non ha più riferimenti a una firma, viene trasferita in LTM. Se il numero di parole trasferite dal dizionario è inferiore al numero di parole aggiunte dalle località nuove o recuperate, vengono trasferite più località. Alla fine di questo processo, la dimensione del dizionario è più piccola di quella precedente all'aggiunta delle nuove pa-

role dalle località nuove e recuperate, riducendo così il tempo necessario per aggiornare il dizionario per la prossima immagine acquisita. Il salvataggio delle località trasferite nel database è fatto in modo asincrono usando un thread in background, portando ad un minimo sovraccarico di tempo. Si noti inoltre che per essere in grado di valutare adeguatamente le ipotesi di chiusura del ciclo utilizzando il filtro bayesiano discreto, le posizioni recuperate dell'iterazione corrente non possono essere trasferite.

Capitolo 3

Caso di studio: Pepper Robot

Pepper è un robot da compagnia/sociale progettato per consentire l'interazione cognitiva e fisica con gli esseri umani. I robot sociali mirano a migliorare le condizioni di vita delle persone che interagiscono con loro. Pepper è in grado di riconoscere i volti e le emozioni umane di base come gioia, tristezza, rabbia o sorpresa. Queste abilità lo rendono ideale per impegnarsi con le persone attraverso la conversazione. Il robot Pepper, creato dalla SoftBank Robotics, è in grado di esibire il linguaggio del corpo, percepire e interagire con l'ambiente circostante e muoversi. Può analizzare le espressioni e i toni della voce delle persone, utilizzando i più recenti progressi e algoritmi nel riconoscimento vocale ed emotivo per stimolare le interazioni. Il robot è dotato di funzionalità e interfacce di alto livello per la comunicazione multimodale con gli esseri umani che lo circondano [65].

Negli ultimi anni è stato proposto l'utilizzo dei robot nell'ambito educativo. L'uso dei robot nell'ambito educativo ha generalmente seguito due direzioni principali: robot come materiale educativo e robot come agenti educativi. Il termine "agente educativo" si riferisce sia ai robot insegnanti, che sono progettati per fornire istruzioni agli studenti, sia ai robot che sono progettati per studiare accanto agli studenti e sostenere il loro apprendimento [66].

Un'altra interessante applicazione, riguarda l'uso di Pepper come guida turistica. Sfruttando gli strumenti software messi a disposizione, è stata progettata un'applicazione di assistenza che permette a un Pepper di comportarsi come una guida turistica e culturale. L'applicazione, chiamata CUMA (Cultural hUManoid Assistant), permette a Pepper di svolgere alcuni compiti di guida, come seguire un visitatore nel suo percorso in un museo, dare una descrizione adeguata dei vari oggetti del museo, fornire informazioni utili sulle altre risorse turistiche della città, rispondere alle domande, raccogliere feedback, ecc. Questo lavoro fa parte di Archeo-

matica (*www.archeomatica.unict.it*), un progetto sulla conservazione e lo sfruttamento del patrimonio culturale attraverso l'informatica [67].

Nell'ambito della interazione uomo-robot (HRI), sono state sviluppate diverse applicazioni. I robot hanno assunto diversi ruoli e svolto compiti come conversare e portare i cestini della spesa in un negozio di alimentari; relazionarsi con le persone, fare pubblicità in un centro commerciale e aiutare negli acquisti, come assistente allo shopping, come nel lavoro dell'UE MuMMER (MultiModal Mall Entertainment Robot [68]). In questo progetto , l'obiettivo è quello di sviluppare un'interazione socialmente appropriata, divertente e coinvolgente per il robot all'interno di un centro commerciale [69].

Viene anche presentato un esempio di utilizzo del robot umanoide Pepper al lavoro della reception. Le sue possibilità di interazione con gli esseri umani sono state estese con capacità aggiuntive includendo dispositivi e moduli esterni. Viene presentato il concetto di un sistema di receptionist automatico costruito sul robot Pepper presentato nel lavoro [70].

Non c'è dubbio che i robot stiano diventando di uso comune: macchine robotiche di tutte le forme e dimensioni stanno entrando nella nostra vita quotidiana. Anche se Pepper è stato inizialmente progettato per una particolare applicazione di usi business-to-business nei negozi SoftBank, il robot è diventato una piattaforma di interesse in tutto il mondo per varie altre applicazioni, anche nelle aree business-to-consumer, business-to-accademics, e business-to-developers e in una varietà di casi d'uso. A livello di software, Pepper è dotato di moduli per realizzare una navigazione di base, ed è anche in grado di evitare gli ostacoli. Inoltre, è possibile utilizzare moduli basati su ROS per la percezione e la pianificazione orientata alla navigazione [71], ed è questa la strada percorsa nel presente lavoro di tesi. Nelle prossime sezioni verranno illustrate la parte hardware e software del robot Pepper.

3.1 Hardware

Il robot Pepper è un robot umanoide alto 120 cm, ha 20 gradi di libertà per il movimento di tutto il corpo (17 articolazioni) e navigazione omnidirezionale (tre ruote). I gradi di libertà sono distribuiti due nella testa, due in ciascuna spalla (sinistra e destra), due in ciascun gomito (sinistro e destro), uno in ciascun polso (sinistro e destro), uno in ciascuna mano (mano destra e sinistra a cinque dita), due sui fianchi, uno sul ginocchio e tre alla base. Le ruote omnidirezionali consentono al robot di salire un gradino di 1.5 cm e fino ad una pendenza di 5°. Gli attuatori sono motori a corrente continua nella parte superiore (negli arti) e motori brushless a corrente continua nell'arto inferiore. Il sensore di posizione del giunto è basato su encoder rotativo

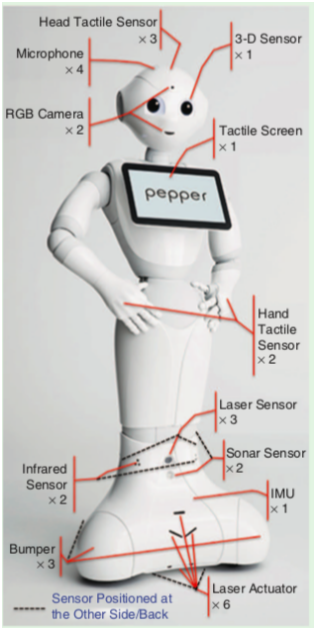


Figura 3.1: Sensori del robot Pepper (immagine fornita da Softbank Robotics).

magnetico e su ciascun motore è presente un sensore di posizione per gli arti superiori. In quasi tutti gli attuatori (spalla, gomito, collo e gamba) vengono utilizzate boccole in plastica per garantire una buona guida, essendo più leggeri, più piccoli e più economici dei cuscinetti a sfera, mentre i cuscinetti a sfera vengono utilizzati solo per gli attuatori delle ruote. Inotre è dotato di supporto alla connettività di rete: include Ethernet (e Wi-Fi) . In termini di protocollo di comunicazione tra i componenti, viene utilizzato RS-485 tra la scheda motore/sensore e il computer interno e il modello di controllo Ethernet di classe del dispositivo di comunicazione viene utilizzato su un cavo bus seriale universale tra il tablet e la CPU.

Sensori per l'interazione umana

Il robot Pepper ha una gamma di sensori che gli consentono di percepire gli oggetti ed esseri umani nei dintorni:

- Microfoni ed altoparlanti: Pepper ha quattro microfoni posizionati nella testa per fornire la localizzazione del suono. Questi hanno una sensibilità di 250 mV / Pa (± 3 dB a 1 kHz) e una gamma di frequenza da 100 Hz a 10 kHz (-10 dB rispetto a 1 kHz). Gli altoparlanti sono posti lateralmente sui lati sinistro e destro della testa.

- Telecamere 2D: il robot ha due telecamere red-green-blue (RGB) posizionate rispettivamente nella fronte e nella bocca. La risoluzione è 640 × 480 a 30 fotogrammi / s.

- Camera ASUS Xtion 3D: la telecamera 3D, sita dietro gli occhi, ha una risoluzione 320 x 120 a 20 fotogrammi / s.

- Sensori sensibili al tatto posizionati rispettivamente tre sulla testa e due sul dorso di entrambe le mani.

- Sensori bumper posizionati rispettivamente su ogni ruota della base mobile ed infine un tablet sul petto.

Sensori di ambiente

- Sensore unità di misura inerziale a sei assi (IMU): il robot è dotato di una IMU composta da un giroscopio a tre assi con un fondo scala di circa 500 °/s e un accelerometro a tre assi per misurare accelerazione fino a circa 2 g. I dati di output consentono una stima della velocità e dell'assetto di base (imbardata, beccheggio e rollio). All'interno della scheda inerziale, viene implementato un algoritmo per calcolare l'angolo di base dall'accelerometro e dal giroscopio.

- Sensori laser: è dotato di un sensore laser per ogni lato della base mobile, tre sono posizionati frontalmente alla base e uno nella testa. I sensori utilizzati coprono un range dai 20 cm ai 2,8 m, con una lunghezza d'onda di 808 nm. Quest'ultima li classifica nella classe 1M, ovvero emettono una radiazione laser innocua nelle normali condizioni di uso fin quando non vengono impiegati strumenti ottici come lenti di ingrandimento o binocoli che possono concentrare l'energia sulla cornea.

- Sonar: sono presenti due sensori sonar rispettivamente anteriormente e posteriormente alla base, coprono un range fino a 5 m con distanza minima di rilevamento pari a 30 cm.

- Sensori ad infrarossi posizionati su entrambi i lati dell'arto inferiore, utilizzati per il rilevamento di oggetti e di movimento.

Dati forniti e consultabili a [72].

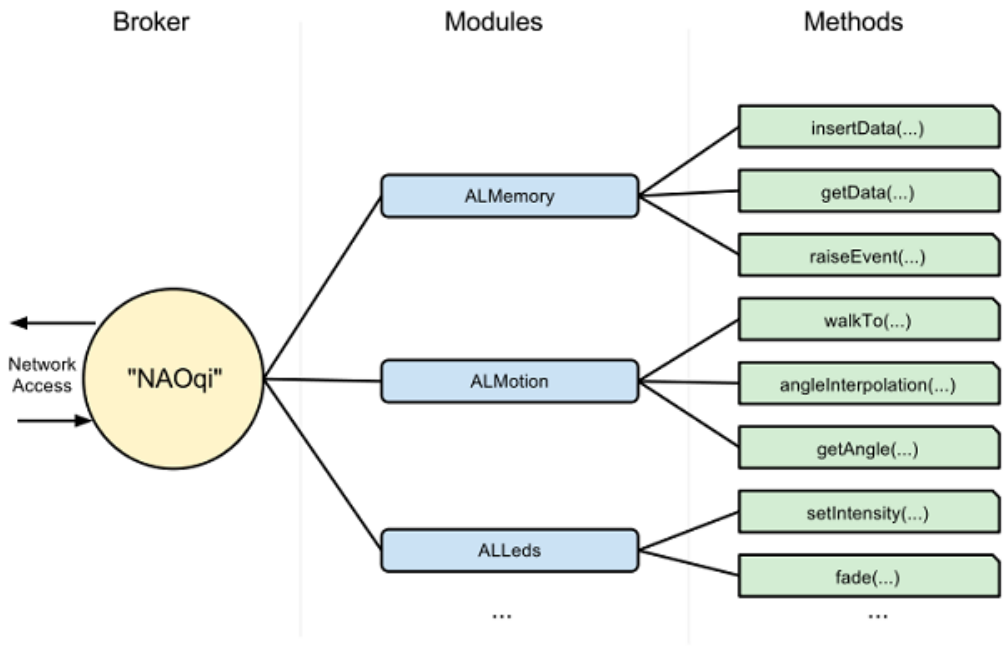


Figura 3.2: Framework NAOqi (immagine fornita da Softbank Robotics).

3.2 Software

Il *NAOqi Framework* è il framework di programmazione utilizzato per programmare NAO e fornisce anche la possibilità di programmazione per sviluppare applicazioni sul robot Pepper.La struttura del framework NAOqi è mostrata in Figura (3.2). Risponde alle esigenze di robotica comuni, tra cui: parallelismo, risorse, sincronizzazione ed eventi. Questo framework consente una comunicazione omogenea tra diversi moduli (movimento, audio, video), una programmazione e condivisione omogenea delle informazioni. Sono forniti diversi kit di sviluppo software per controllare Pepper e svilupparlo: Python, C ++, Java, JavaScript e l'interfaccia Robot Operating System (ROS). L'eseguibile NAOqi che gira sul robot è un broker. All'avvio, carica un file delle preferenze che definisce quali librerie caricare. Ogni libreria contiene uno o più moduli che utilizzano il broker per pubblicare i propri metodi e fornisce servizi di ricerca in modo che qualsiasi modulo nell'albero o nella rete possa trovare qualsiasi metodo pubblicato. Un broker è un oggetto che fornisce due ruoli principali: servizi di directory che consentono di trovare moduli e metodi, e, l' accesso alla rete che consente di chiamare i metodi dei moduli collegati dall'esterno del processo. Per quanto riguarda i moduli,

essi sono tipicamente delle classi all'interno di una libreria che contengono i vari metodi. Al caricamento della libreria, essa istanzia automaticamente la classe del modulo. Un modulo può essere remoto o locale, a seconda se esso può essere utilizzato esclusivamente dal robot o anche dall'esterno. Tra i vari software compatibili è stato utilizzato il Robot Operating System che verrà illustrato nella seguente sottosezione.

3.2.1 Robot Operating System

Il *Robot Operating System* (ROS) è una piattaforma di sviluppo di applicazioni per robot che fornisce varie funzionalità come trasmissione di messaggi, elaborazione distribuita, riutilizzo del codice e così via. ROS è un framework middleware open source con librerie e strumenti per lo sviluppo di software per robot. Include algoritmi all'avanguardia, un framework di comunicazione inter-processo e strumenti di visualizzazione. È usato su molti robot e gruppi di ricerca grazie alla sua astrazione dell'hardware e alla gestione dei pacchetti, con molti algoritmi di percezione, pianificazione e localizzazione. In ROS le unità di elaborazione sono chiamate nodi, che comunicano tramite argomenti o servizi. Gli argomenti seguono il paradigma publisher/subscriber, mentre i servizi lavorano in un modello client/server. C'è sempre un nodo di coordinamento principale, ma altri nodi possono essere distribuiti, permettendo il funzionamento distribuito su più macchine [73].

I motivi per la sua scelta sono molteplici:

- Funzionalità di alto livello: ROS viene fornito con funzionalità pronte per l'uso, come ad esempio, i pacchetti SLAM (Simultaneous Localization and Mapping) e AMCL (Adaptive Monte Carlo Localization) che possono essere utilizzati per eseguire la navigazione autonoma nei robot mobili.

- Varietà di strumenti: ROS fornisce numerosi strumenti tra cui visualizzazione e esecuzione di simulazioni. Ad esempio Rviz e Gazebo sono alcuni dei potenti strumenti open source per la visualizzazione e simulazione.

- Supporto ad alto livello di sensori e attuatori: fornisce una serie di pacchetti per permettere l'interfacciamento tra sensori e attuatori senza problemi.

- Interoperabilità della piattaforma: attraverso il suo middleware è possibile mettere in comunicazione nodi differenti, inoltre, questi nodi pos-

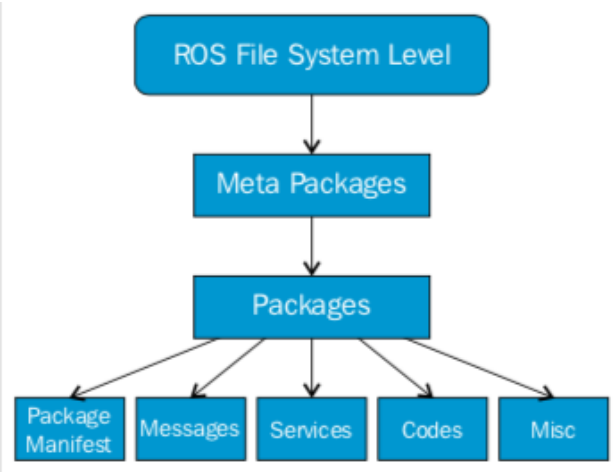


Figura 3.3: File System ROS. Immagine da [74]

sono essere programmati in linguaggi diversi (purché abbiano le librerie ROS), come ad esempio C, C++, Python e Java.

- Modularità: questa caratteristica assicura il funzionamento dell'applicazione anche se sono presenti dei nodi malfunzionanti. Infatti in ROS è possibile scrivere un nodo per ogni processo, in modo da permettere al processo di continuare a lavorare in caso di presenza di nodi non funzionanti. Inoltre fornisce anche metodi affidabili per riprendere operazioni in caso di guasto a sensori o attuatori.

- Gestione simultanea delle risorse: è possibile accedere ai dispositivi utilizzando i ROS topic dai ROS driver, qualsiasi numero di nodi può iscriversi ai topic ed ognuno di essi può svolgere funzionalità diverse.

File system

Simile a un sistema operativo, anche i file ROS sono organizzati su disco in modo particolare, vedi Figura (3.3):

- Packages: i pacchetti ROS sono l'unità di base del software. Contengono il processo a runtime ROS (nodi), librerie, file di configurazione e così via, e sono organizzati come una singola unità.

- Package manifest: il file .xml del package manifest è contenuto all'interno di un pacchetto che contiene informazioni riguardanti il pacchetto stesso, autore, licenza, dipendenze ed altro.

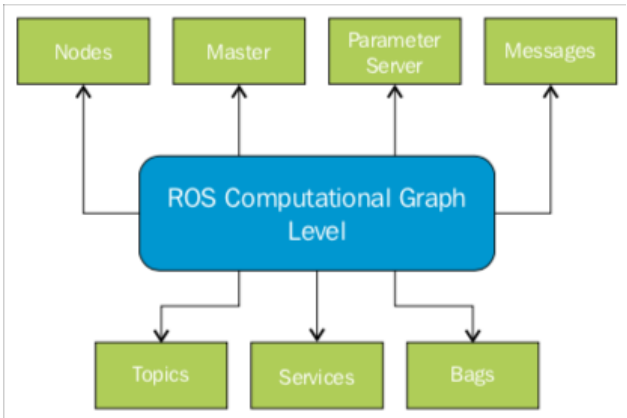


Figura 3.4: Computation graph level ROS. Immagine da [74]

- Meta package: questo termine è usato per un gruppo di pacchetti che possiedono particolari funzioni come ad esempio lo stack di navigazione ROS.
- Meta package manifest: è simile al package manifest, le differenze sono che potrebbe includere pacchetti al suo interno come dipendenze a runtime e dichiarazioni di tag di esportazione.
- Messaggi: i messaggi ROS sono delle informaizioni che vengono inviate tra processi ed è possibile definire dei messaggi personalizzati.
- Servizi: sono una sorta di interazione di tipo richiesta / risposta tra processi ed è possibile definire il tipo di dato all'interno del pacchetto.
- Archivio: la maggior parte dei pacchetti ROS viene mantenuta utilizzando un sistema di controllo della versione (VCS) come Git, subversion (svn), mercurial (hg) e così via. La raccolta di pacchetti che condividono un VCS comune può essere chiamata repository.

Computation graph level

Il *Computation Graph*, mostrato in Figura (3.4), è la rete di tipo peer-to-peer dei processi ROS che elaborano dati. I concetti base alla base di questo sono nodi, master, server dei parametri, messaggi, servizi, topic e bag, i quali, forniscono dati ma in modi diversi. I nodi sono i processi che eseguono calcoli. Ogni nodo ROS è scritto usando le librerie client ROS come roscpp e rospy. Utilizzando le API delle librerie, si possono implementare diversi tipi di metodi di comunicazione nei nodi ROS. In un robot, ad esempio, ci saranno molti nodi per eseguire diversi tipi di compiti. Utilizzando i metodi

di comunicazione ROS, possono comunicare tra loro e scambiare dati. Uno degli obiettivi dei nodi ROS è quello di costruire processi semplici piuttosto che un grande processo con tutte le funzionalità, ad esempio un nodo può controllare i motori delle ruote ed un altro esegue la localizzazione e così via. L'uso dei nodi può rendere il sistema tollerante ai guasti, ad esempio se un nodo si blocca, un intero sistema robotico può ancora funzionare, ed inoltre riducono la complessità e aumentano la possibilità di debug rispetto ai codici monolitici in quanto ogni nodo gestisce una singola funzione.

Il *ROS Master* fornisce la registrazione del nome e la ricerca al resto dei nodi. I nodi non saranno in grado di trovarsi a vicenda, scambiare messaggi o invocare servizi senza un ROS Master, quando qualsiasi nodo si avvia nel sistema ROS, inizierà a cercare il master e a registrarsi, così il master ha i dettagli di tutti i nodi attualmente in esecuzione sul sistema, e, quando uno qualsiasi dei dettagli dei nodi cambia, genererà una call-back e lo aggiornerà. Questo permette quindi ai nodi di connettersi tra loro. Il server dei parametri permette di mantenere i dati da memorizzare in una posizione centrale, in cui tutti i nodi possono accedervi ed apportare modifiche. Esso è una parte del Master.

I messaggi vengono utilizzati tra i vari nodi per la comunicazione, sono semplicemente una struttura dati contenente un campo tipizzato che può contenere un insieme di dati. I tipi standard (int, bool, float e così via) sono supportati dai messaggi ROS. Ogni messaggio è trasportato su un bus di comunicazione chiamato topic. Quando un nodo invia un messaggio attraverso un topic esso è di tipo 'publish' ovvero sta pubblicando sul topic, mentre, se un nodo riceve messaggi, esso è un nodo di tipo 'subscribe' ovvero sta sottoscrivendo il topic. Entrambe le tipologie di nodi non sono a conoscenza l'una dell'altra, questo ci permette di sottoscrivere dei topic che non hanno alcun publisher, in altre parole, la produzione e il consumo di dati sono disaccoppiati. Ogni topic ha un nome unico e qualsiasi nodo può accedervi e inviare messaggi attraverso di esso, purché abbia il giusto tipo di messaggio. In alcune applicazioni robotiche, il modello publish/subscribe non è sufficiente se si ha bisogno di interazioni di tipo richiesta/risposta, come ad esempio i sistemi distribuiti, perché è una sorta di sistema di trasporto unidirezionale che non è sufficiente, ed è in questi casi che vengono utilizzati i *servizi* ROS. Vengono definiti in due parti, una è per le richieste e l'altra è per le risposte; usandoli, è possibile quindi definire nodi client e nodi server.

Per quanto riguarda i ROS bag, essi sono un formato per salvare e riprodurre dati dei messaggi, sono un meccanismo per la memorizzazione dei dati, come quelli sensoriali ad esempio, che possono risultare difficili da raccogliere ma necessari per lo sviluppo e test di algoritmi.

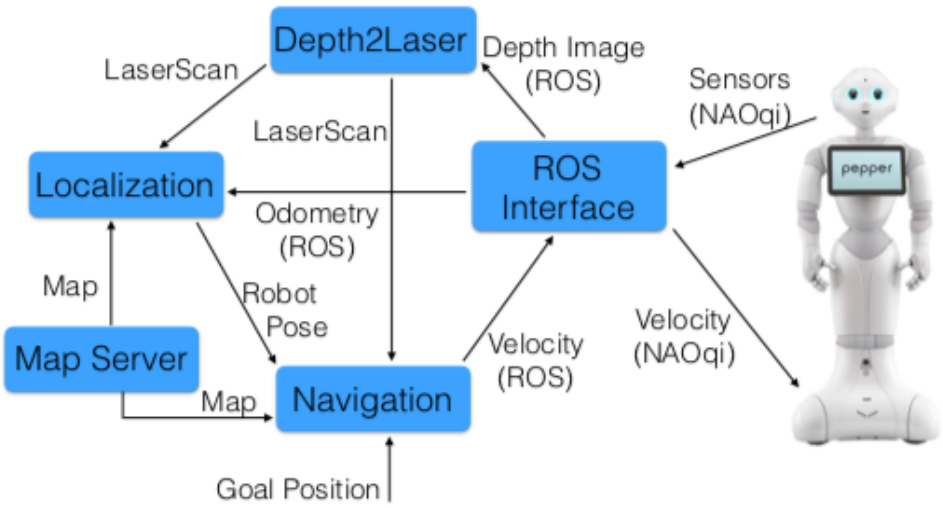


Figura 3.5: Architettura software per localizzazione e navigazione con ROS su Pepper Robot, con i nodi ROS mostrati in blu e le frecce che rappresentano la comunicazione sugli argomenti. Immagine da [73]

Interfaccia ROS per navigazione e localizzazione

La comunità ROS supporta un nodo ROS Interface per collegare ROS e il sistema NAOqi [73]. Funziona sia come nodo ROS che come modulo NAOqi, e traduce le chiamate NAOqi in servizi ROS e argomenti con tipi standard. ROS è utilizzato come strumento per consentire una facile integrazione di Pepper con tecniche di navigazione innovative e strumenti di visualizzazione di base. Inoltre, una volta che l'interfaccia ROS è in esecuzione, gli sviluppatori possono creare software per Pepper con tipi e comunicazioni standard, astraendo dal suo hardware specifico. Questo rende anche facile il porting del codice da e verso altri robot e simulazioni.

L'architettura software per eseguire la localizzazione e la navigazione con ROS su Pepper, presentata nella Figura (3.5), ha un elemento centrale, la ROS Interface. La ROS Interface si registra come modulo nel NAOqi, e poi fa delle chiamate con le API del NAOqi per leggere i sensori di Pepper e inviare comandi di velocità alla sua base. In questo specifico esempio in Figura (3.5) , il ciclo è chiuso dai restanti nodi ROS. Dopo aver letto i dati del sensore da Pepper, l'interfaccia ROS pubblica l'immagine di profondità e l'odometria usando i tipi standard di ROS. Un altro nodo sottoscrive l'immagine di profondità e la converte in una scansione laser.

Localizzazione e navigazione Con la scansione laser e l'odometria pubblicate, e una mappa fornita dal map server, il robot è in grado di localizzarsi nel sistema di riferimento della mappa usando AMCL, l'Adaptive Monte Carlo Localization. Infine, usando la mappa, la stima della posa del robot dalla localizzazione, e il Laser Scan per evitare gli ostacoli, il robot può navigare autonomamente verso una posizione obiettivo. Questa posizione può essere data da uno strumento di visualizzazione come RViz, o anche indirettamente attraverso l'interazione vocale come richiesta dell'utente. Il nodo di navigazione trova il percorso ottimale per andare dalla posizione attuale alla posizione di destinazione, evitando gli ostacoli nella mappa e anche visti per la prima volta durante la navigazione. Infine, il nodo di navigazione trova anche i comandi di velocità che devono essere inviati al robot in modo che segua il percorso pianificato. L'interfaccia ROS chiude poi il ciclo traducendo i comandi di velocità con tipo ROS standard in una chiamata di funzione al NAOqi per inviare il comando.

Capitolo 4

Test e risultati ottenuti

In questo capitolo si mostreranno le varie prove eseguite e i risultati ottenuti dall'applicazione dei vari algoritmi di localizzazione precedentemente illustrati. In particolare, l'algoritmo SLAM è stato utilizzato per la creazione della mappa dell'ambiente (dipartimento dell'ingegneria dell'informazione DII dell'università Politecnica delle Marche) tele-operando il robot da tastiera, mentre, l'algoritmo Monte Carlo è stato utilizzato per localizzare il robot all'interno delle mappe create, valutando la qualità delle mappe ottenute. Per questa valutazione, su ogni mappa creata, sono stati eseguiti due test. Per il primo test è stato definito un percorso rettangolare di dimensione 7 e 1.45 m, rispettivamente, considerando le coordinate (x,y) dei quattro vertici noti. Partendo dal punto iniziale è stato percorso 10 volte (per ottenere 10 misure su ogni punto) e su ogni vertice sono stati poi calcolati valor medio dell'errore e deviazione standard. Per il secondo test è stata definita una traiettoria che copre l'intero ambiente precedentemente mappato.

4.1 Creazione mappe

Il pacchetto utilizzato per la creazione delle mappe è *Gmapping* [75], che contiene un wrapper ROS per Gmapping di OpenSlam. Il pacchetto *gmapping* fornisce SLAM, come un nodo ROS chiamato *slam_gmapping*. Utilizzando *slam_gmapping*, è possibile creare una mappa a griglia di occupazione 2D (come la pianta di un edificio) esclusivamente dai dati laser e di posa (odometria) raccolti dal robot mobile Pepper. Una mappa è vista come un campo casuale di variabili che sono disposte in una griglia. Ogni variabile è binaria e indica se la posizione corrispondente è occupata o meno.

Questo pacchetto, come appena detto, utilizza dati in ingresso di tipo *LaserScan* (struttura dati che contiene una singola acquisizione dei dati la-

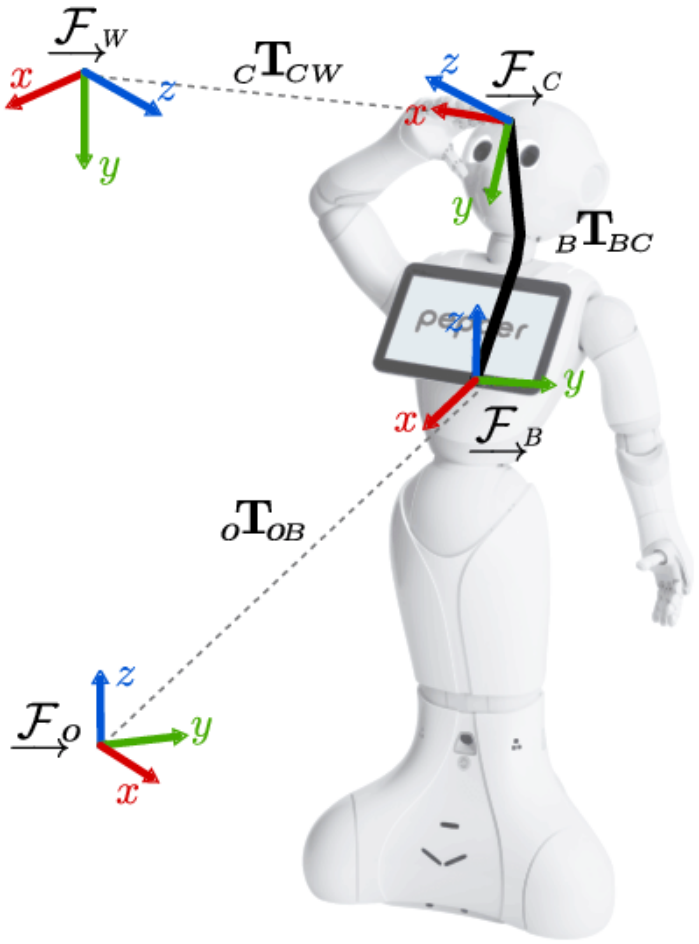


Figura 4.1: Esempio di trasformazioni tra sistemi di riferimento. (Immagine da [76])

ser) e le trasformazioni (TF) necessarie per mettere in relazione i vari frame, cioè i vari sistemi di riferimento del laser, della base del robot e dell'odometria. Nella Figura (4.1) è mostrato un esempio di vari sistemi di riferimento che sono in relazione e quindi collegati attraverso trasformazioni. Le varie trasformazioni di coordinate sono raccolte nel pacchetto *tf* che permette all'utente di tenere traccia di più frame di coordinate. Il pacchetto *tf* mantiene la relazione tra i frame di coordinate in una struttura ad albero e permette all'utente di trasformare punti, vettori, ecc. tra qualsiasi due frame di coordinate collegati tra loro in qualsiasi momento, come mostrato nella Figura (4.2).

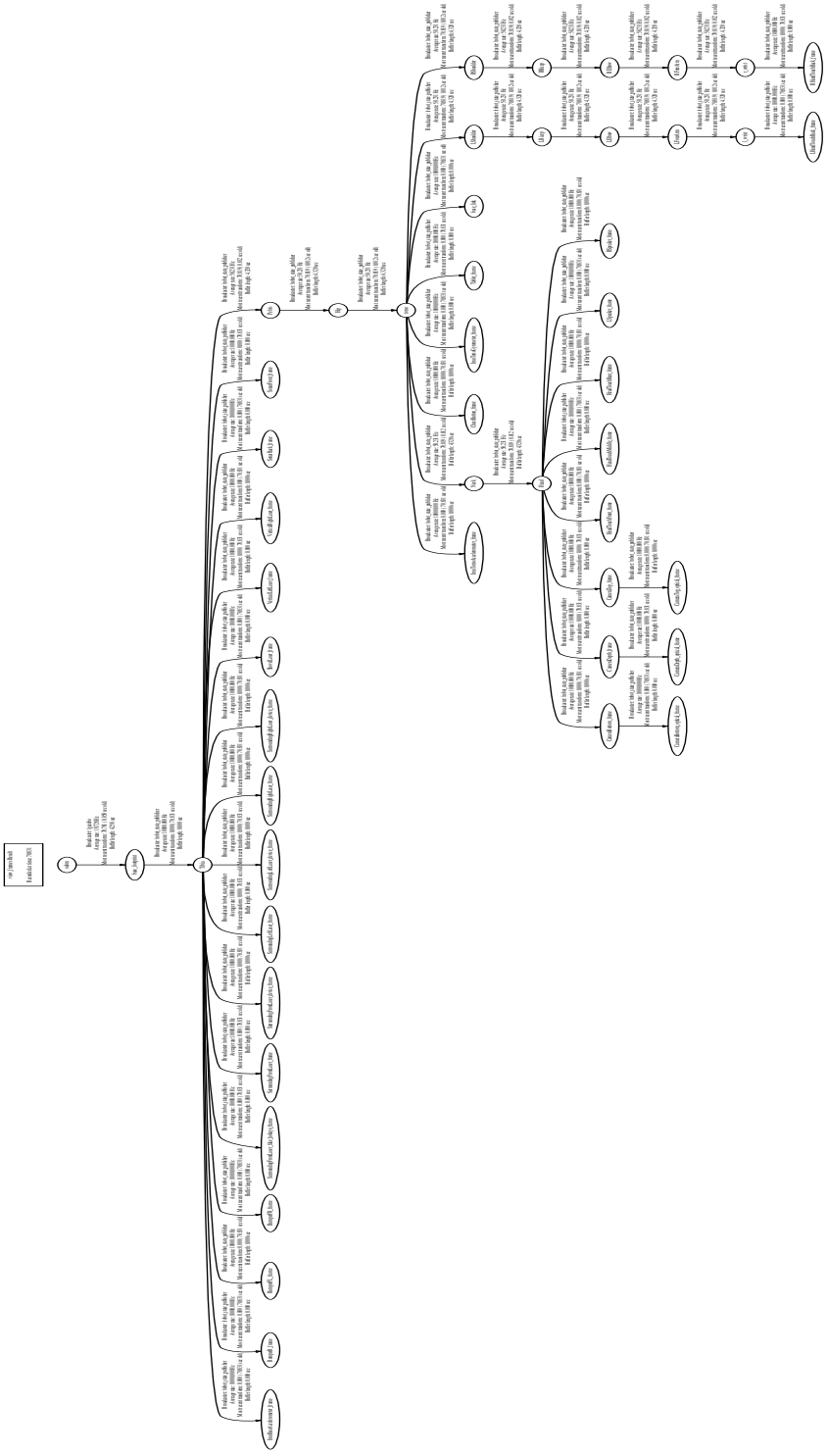


Figura 4.2: Albero delle trasformazioni tra i vari frame di coordinate

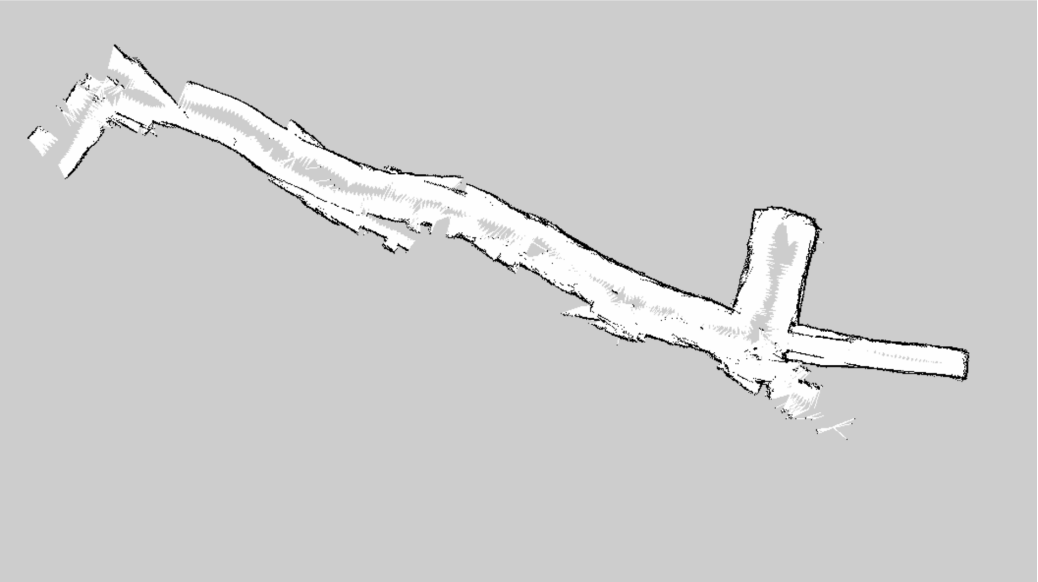


Figura 4.3: Mappa 2D ottenuta con tecnica 1

La trasformazione richiesta dal pacchetto *gmapping* è quella dal frame 'base-link' al frame 'odom', che viene già data dal sistema odometrico, mentre, la trasformazione che viene pubblicata dal pacchetto è quella da 'map' a 'odom' che indica lo stato stimato della posa del robot al frame mappa, in altre parole la trasformazione pubblicata indica che il robot oltre a creare la mappa sta contemporaneamente localizzandosi all'interno di essa. Sono state utilizzate tre differenti tecniche, elencate di seguito, per creare le mappe di occupazione 2D che verranno poi usate in seguito nei test:

- *Tecnica 1*: in questo caso è stato adottato l'approccio generalmente utilizzato in letteratura in fase di creazione di mappe 2D; in particolare è stato utilizzato l'algoritmo SLAM (pacchetto *slam_gmapping*) al quale sono stati forniti i dati dai 3 sensori laser del robot Pepper, posizionati sulla base.

  In questo caso è stato necessario un ulteriore step in quanto SLAM necessita che tutti i dati sensoriali siano in un unico topic, per cui è stato creato un nodo ad hoc che prende le misure sensoriali dai tre topic presenti e le aggrega in uno unico. La mappa ottenuta è mostrata in Figura (4.3).

- *Tecnica 2*: in questo caso è stato adotatto lo stesso approccio ma utilizzando il sensore camera 3D CameraDepth. Per utilizzare questo sensore, è stato necessario eseguire delle conversioni in quanto il pacchetto

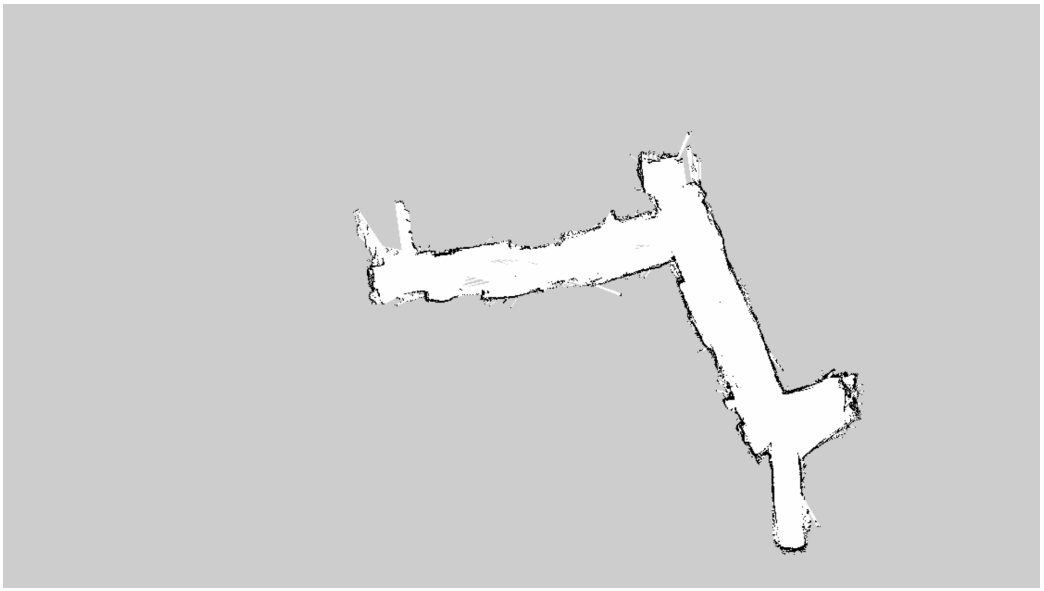


Figura 4.4: Mappa 2D ottenuta con tecnica 2

gmapping prevede esclusivamente in ingresso dati di tipo *LaserScan*: prima è stata eseguita la conversione in Point Cloud (punti x,y e z) e successivamente la conversione in dati compatibili con quelli richiesti dal pacchetto LaserScan.

In particolare, partendo dall'immagine di profondità generata dal sensore 3D del Pepper, tramite una trasformazione si ottiene la corrispondente scansione laser. La comunità ROS fornisce un semplice pacchetto, *depthimage to laser scan*, che può eseguire questa trasformazione. Tuttavia, questo pacchetto è adatto per le immagini acquisite da sensori 3D fissi e montati orizzontalmente, ad una bassa altezza che permette loro di rilevare facilmente gli ostacoli. Tuttavia, Pepper ha il suo sensore 3D nella testa, che può quindi muoversi e cambiare orientamento. Pertanto, sono stati utilizzati invece altri due pacchetti: il primo, *depth image proc*, converte le immagini di profondità in dati di tipo PointCloud; il secondo, *pointcloud to laserscan*, crea la scansione laser 2D dalla nuvola di punti 3D. L'ultimo nodo può convertire i dati 3D dal suo frame di riferimento ad altri frame di destinazione, essendo flessibile sull'altezza minima e massima dei punti da considerare per la conversione e permette anche di fare la conversione quando la testa del robot è inclinata verso il basso. La mappa così ottenuta è mostrata in Figura (4.4).

Figura 4.5: Mappa 2D ottenuta con tecnica 3

- *Tecnica 3*: in questo caso non viene utilizzato l'algoritmo SLAM del pacchetto *slam_ gmapping* come nei casi precedenti, bensì, la mappa è stata creata attraverso il pacchetto *Rtab-Map* [77]: questo pacchetto è un wrapper ROS di RTAB-Map (Real-Time Appearance-Based Mapping), un approccio RGB-D SLAM basato su un rilevatore di chiusura globale del loop (loop closure detection) con vincoli in tempo reale. Questo pacchetto può essere utilizzato per generare un point cloud 3D dell'ambiente e per creare una mappa a griglia di occupazione 2D per la navigazione. Questo algoritmo differisce dai precedenti in quanto richiede in ingresso dati sensoriali diversi: immagini RGB, immagini depth, scansioni laser e point cloud generato dal sensore laser (questo messaggio contiene una collezione di punti N-dimensionali, che possono contenere informazioni aggiuntive). La mappa ottenuta è mostrata nella Figura (4.5).

Inoltre viene considerata un ulteriore mappa per eseguire i test, che è stata creata con un sensore Hokuyo non appartenente al robot Pepper e utilizzando l'algoritmo *slam_ gmapping*. La mappa è mostrata in Figura (4.6). È evidente come la mappa creata con il sensore 3D (tecnica 2) differisca dalle altre e non verrà utilizzata per ulteriori test. Questo può essere il risultato di ambiguità di misura ovvero due punti di riferimento sono abbastanza vicini che l'osservazione potrebbe plausibilmente provenire da uno dei due, o, da

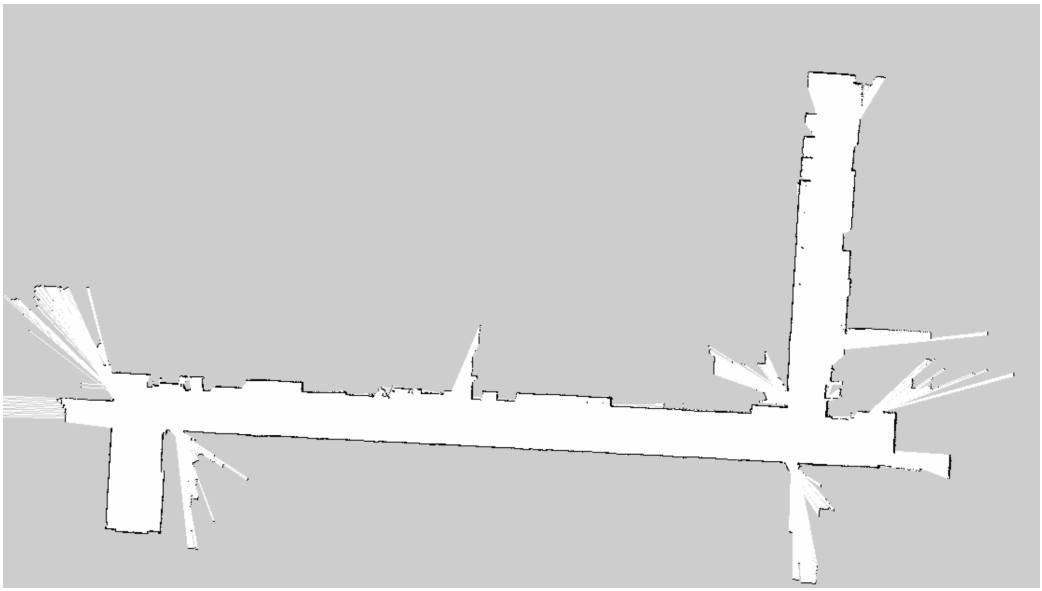


Figura 4.6: Mappa 2D ottenuta con sensore laser Hokuyo

ambiguità di movimento, osservazioni potrebbero essere associate a landmark completamente differenti se l'orientamento del robot cambia.

4.2 Validazione delle perfomance in localizzazione

Le mappe sono state poi utilizzate come ingresso al pacchetto *amcl*, che implementa l'algoritmo di localizzazione Monte Carlo, il quale produce stime della posa richiedendo come ingresso una mappa 2D e messaggi di trasformazione. *Amcl* trasforma le scansioni laser in entrata nel frame dell'odometria, quindi deve esistere un percorso attraverso l'albero delle trasformazioni dal frame in cui le scansioni laser sono pubblicate al frame dell'odometria. Durante il funzionamento *amcl* stima la trasformazione del sistema di riferimento della base (*base frame*) rispetto al sistema di riferimento globale (*global frame*), ma pubblica solo la trasformazione tra il sistema di riferimento globale e dell' odometria (*odom frame*). Essenzialmente, questa trasformazione tiene conto della deriva che si verifica utilizzando il Dead Reckoning. Per capire meglio, viene mostrata la differenza tra le trasformazioni della localizzazione tramite odometria e della localizzazione tramite *amcl* mostrato in Figura (4.7), e, proprio per questo che il pacchetto *amcl* influisce sui dati forniti dall'odometria e quindi sulle rispettive letture.

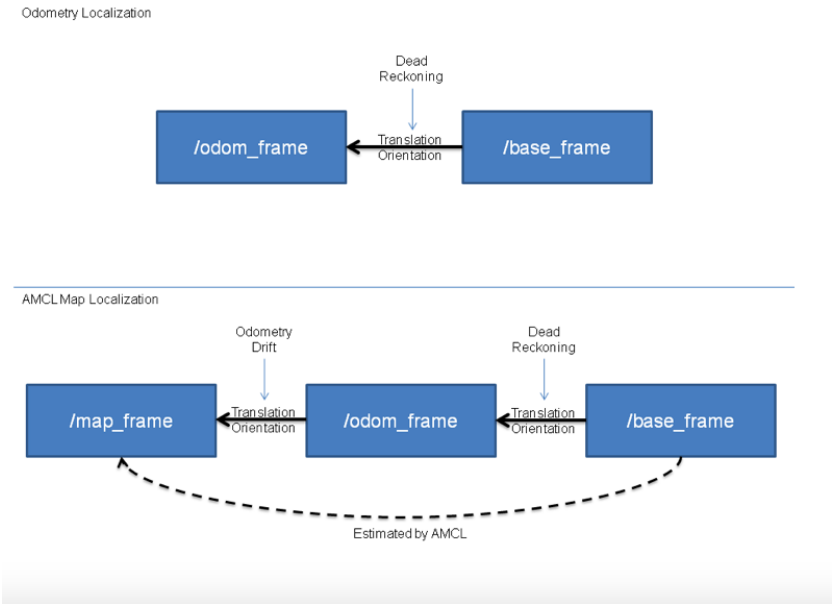


Figura 4.7: Differenza tra trasformazioni di coordinate tra localizzazione odometrica e localizzazione Monte Carlo. Immagine disponibile su *wiki.ros.org*.

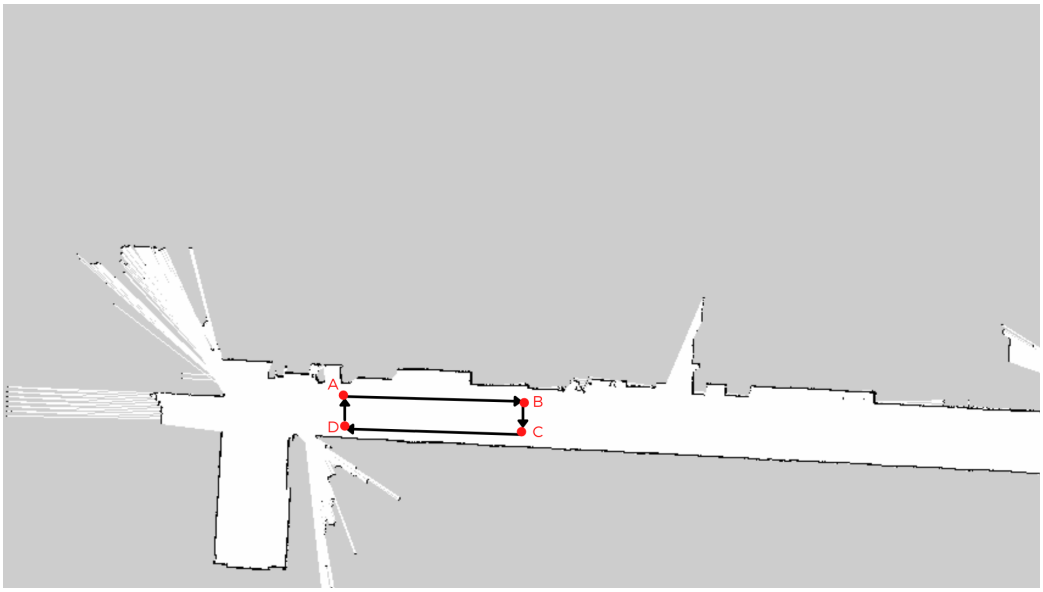


Figura 4.8: Percorso eseguito per il primo test

Per valutare la qualità delle mappe sono stati effetuati due test. Per il primo test è stato definito un percorso chiuso all' interno della mappa, vedi

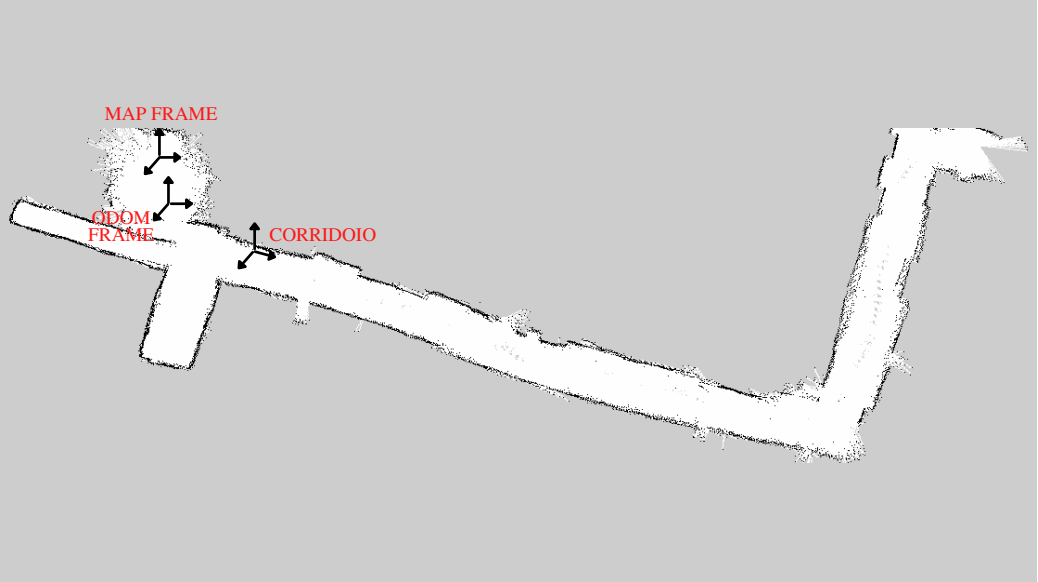


Figura 4.9: Sistemi di riferimento odom, map e corridoio.

Figura (4.8). Partendo dal punto iniziale, sono stati compiuti 10 giri consecutivi e sono stati raccolti i dati odometrici relativi ai quattro vertici A,B,C, e D. I dati odometrici vengono intrinsecamente influenzati dalla stima della posa ottenuta da Monte Carlo, che risulta essere tanto migliore quanto migliore è la qualità della mappa utilizzata. Per la valutazione vengono considerate le coordinate dei quattro vertici del percorso, che vengono lette attraverso i dati odometrici stimati (come appena spiegato) dall'algoritmo Monte Carlo (stime di posa). Su ogni vertice del percorso rettangolare vengono poi calcolati valor medio dell'errore e deviazione standard di tutte le varie misurazioni eseguite. Questi valori vengono confrontati con le coordinate reali del percorso, cioè: consideriamo un sistema di riferimento che coincida con il vertice 'A' del percorso (formato quindi dai vertici A, B, C e D), ed orientato con l'asse X lungo il lato più lungo del rettangolo. In questo modo immaginiamo il sistema di riferimento reale, chiamato 'corridoio'. I dati odometrici forniti dal robot, invece, sono espressi rispetto il sistema di riferimento 'odom'. La posa del robot nel frame odom può andare alla deriva nel tempo, senza alcun limite. Questa deriva rende questo frame inutile come riferimento globale a lungo termine, anche se la posa è garantita come continua, il che significa che evolve sempre in modo regolare, senza salti discreti. In una tipica configurazione l"odom frame' si riferisce a una fonte odometrica, come l'odometria delle ruote, l'odometria visiva o un'unità di misura inerziale. L'odom frame è utile come riferimento locale accurato e a breve termine, ma la deriva lo

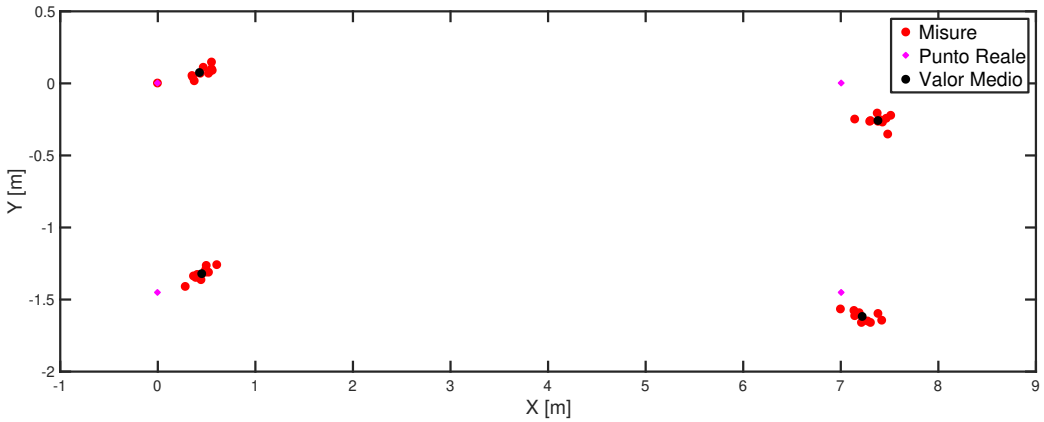


Figura 4.10: Posizione stimata sulla mappa ottenuta tramite tecnica 1

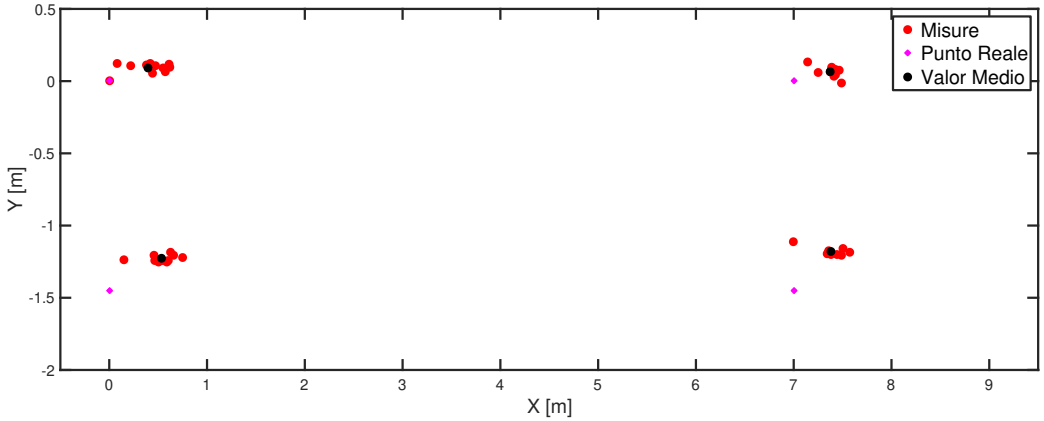


Figura 4.11: Posizione stimata sulla mappa ottenuta tramite tecnica 3

rende un frame non adatto per il riferimento a lungo termine. Il frame 'map' è un frame fisso al sistema mondo, con il suo asse Z che punta verso l'alto. La posa del robot, relativa al sistema di riferimento della mappa, non deve avere una deriva significativa nel tempo, ovvero un componente di localizzazione ricalcola costantemente la posizione del robot nel frame della mappa in base alle osservazioni dei sensori, eliminando così la deriva, ma causando salti discreti quando arrivano nuove informazioni dai sensori. In Figura (4.9) vengono mostrati i sistemi di riferimento. Il frame 'map' indica l'origine della mappa, cioè il punto dal quale il robot Pepper ha iniziato a creare la mappa. Il frame 'odom' indica il punto dove il robot viene inizializzato. Il frame 'corridoio' coincide con il punto A del percorso.

Per poter effettuare quindi un confronto considerando le coordinate (x,y) reali del sistema 'corridoio', occorre in primo luogo eseguire una trasforma-

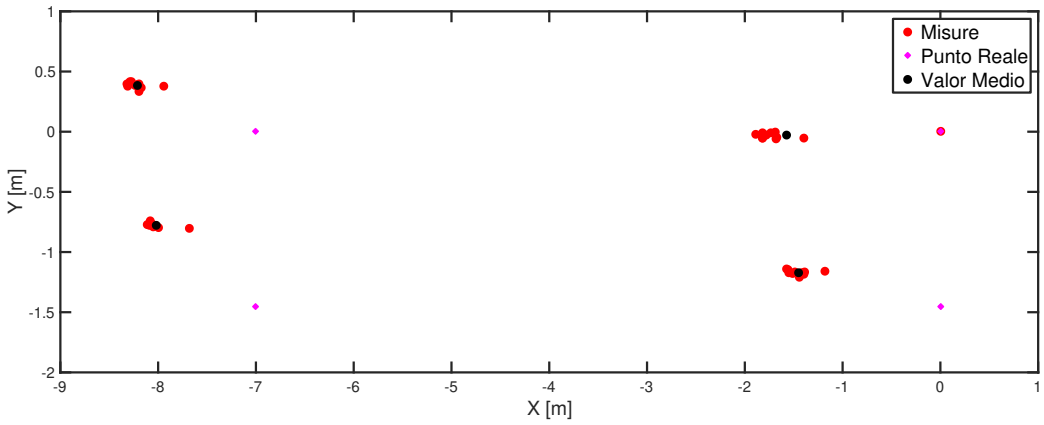


Figura 4.12: Posizione stimata sulla mappa ottenuta tramite sensore Hokuyo

zione di coordinate dal frame 'odom' al frame 'map' (che corrisponde al punto iniziale da cui il robot è partito precedentemente in fase di creazione della mappa), trasformazione possibile essendo l'albero delle trasformazioni collegato ad entrambi i frame, e, infine traslando il sistema 'map' su quello reale 'corridoio'. In questo modo si possono considerare le coordinate dei vertici reali, cioè, note perché misurate direttamente sul luogo nel quale è stato effettuato il test. Nella Figura (4.8) viene mostrato il percorso eseguito.

Di seguito vengono riportati i risultati ottenuti da questo primo test. In particolare, nelle Tabelle (4.1), (4.2), e (4.3) sono riportati i valori medi e le deviazioni standard di ogni punto, per le mappe generate con tecnica 1, 3, e tramite Hokuyo (quella di riferimento per la valutazione delle performance), rispettivamente. Le Figure (4.10), (4.11), e (4.12) rappresentano graficamente i dati odometrici misurati (in rosso) e la posizione vera dei punti A,B,C, e D (in viola), per le stesse mappe.

I risultati ottenuti da questo primo test mostrano su entrambe le mappe create tramite i sensori del robot, una deviazione standard, cioè, una dispersione dei dati intorno al valore medio, bassa che non supera i 0.25 m lungo l'asse X e i 0.05 m lungo l'asse Y (risultato coerente essendo il percorso lungo Y (1.45 m) più piccolo rispetto a quello lungo X (7 m)). Questi risultati mostrano che la mappa ottenuta dal laser Hokuyo ha un errore maggiore e una deviazione standard minore rispetto alle altre mappe e questo può essere spiegato considerando un offset sulle misure.

Come secondo test, viene eseguito un percorso lungo una traiettoria scelta, che copra l'intero corridoio, mostrata in Figura (4.13). La traiettoria è rappresentata graficamente in Figura (4.13) dove la croce rossa rappresenta il punto di partenza e di arrivo, in quanto il percorso viene eseguito in avanti e a ritroso. Vengono graficate le coordinate (x,y) della traiettoria ottenute dai

	Valore medio dell'errore [m]	Deviazione standard [m]
A	(0.4258,0.073)	(0.1623,0.0425)
B	(0.3789,0.2578)	(0.1092,0.0390)
C	(0.2202,0.1658)	(0.1255,0.0354)
D	(0.4478,0.1282)	(0.0916,0.0462)

Tabella 4.1: Valor medio dell'errore e rispettiva deviazione standard riferiti alla mappa ottenuta con tecnica 1, a seguito di 10 ripetizioni del percorso, Figura (4.10)

	Valore medio dell'errore[m]	Deviazione standard [m]
A	(0.3975,0.0904)	(0.2114,0.0369)
B	(0.3748,0.0672)	(0.1033,0,0400)
C	(0.3804,0.2698)	(0.1555,0.0271)
D	(0.5336,0.2216)	(0.1631,0.0232)

Tabella 4.2: Valor medio dell'errore e rispettiva deviazione standard riferiti alla mappa ottenuta con tecnica 3, a seguito di 10 ripetizioni del percorso, Figura (4.11)

	Valore medio dell'errore [m]	Deviazione standard [m]
A	(-0.1203,-1.6624)	(0.0463,0.5678)
B	(0.0268,-1.5736)	(0.0279,0.1122)
C	(-0.0804,-1.6098)	(0.0191,0.1285)
D	(-1.4917,-0.3236)	(0.0224,0.1215)

Tabella 4.3: Valor medio dell'errore e rispettiva deviazione standard riferiti alla mappa otteuta con sensore Hokuyo, a seguito di 10 ripetizioni del percorso, Figura (4.12)

dati odometrici. Anche in questo caso, come nei test precedenti, le coordinate del percorso vengono trasformate dal frame 'odom' al frame 'map' e infine per una maggiore comprensione viene poi riportato sul sistema di riferimento 'corridoio'. In tutti i test eseguiti, dobbiamo anche tener conto che i percorsi eseguiti sono influenzati da fattori esterni come lo slittamento delle ruote che viene causato dal pavimento essendo una superficie non perfettamente liscia. Questi fattori dovranno essere considerati nel confronto tra il percorso desiderato e quello ottenuto.

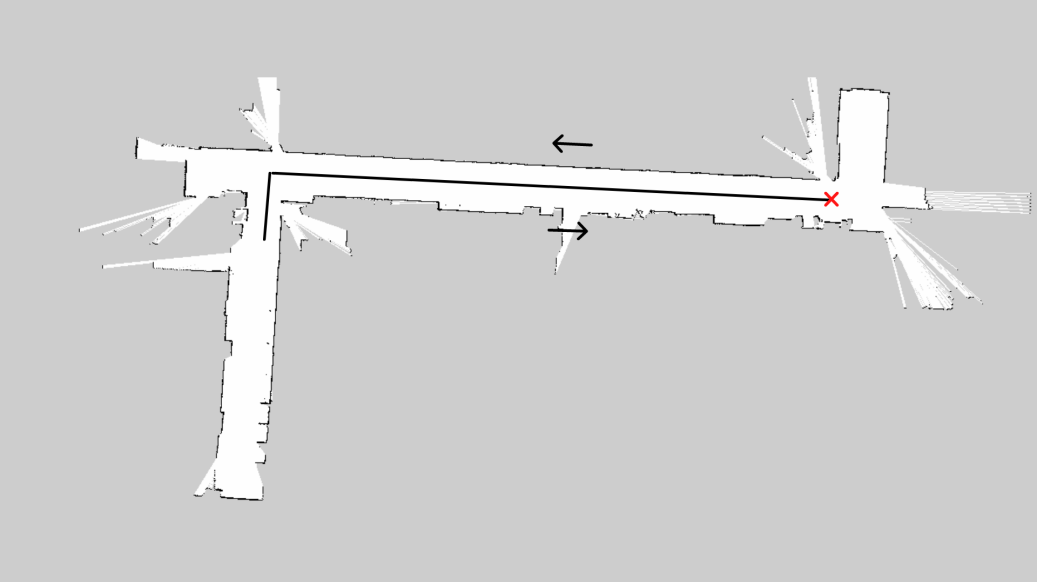


Figura 4.13: Percorso eseguito per il secondo test

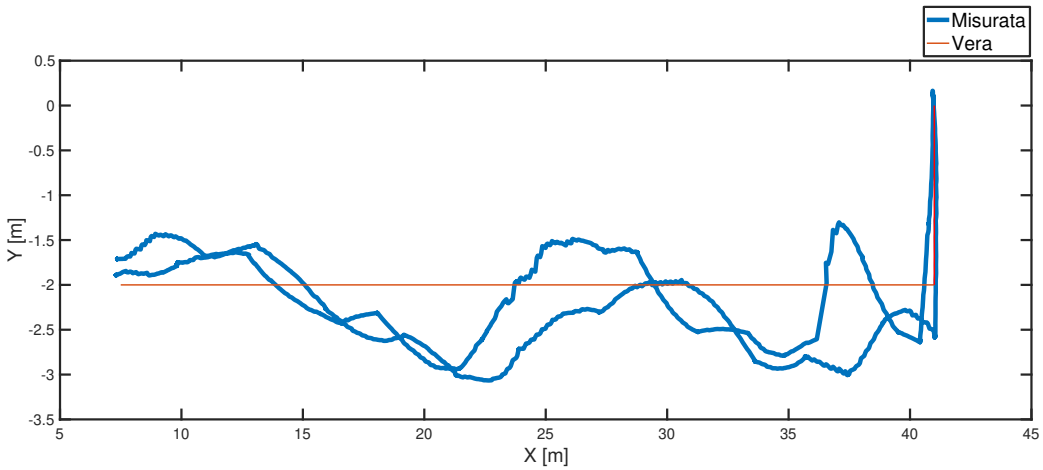


Figura 4.14: Percorso eseguito su mappa ottenuta con tecnica 1

Le Figure (4.14), (4.15), e (4.16) mostrano i percorsi eseguiti cioè le coordinate (x-y) ottenute dalle letture odometriche di ogni mappa. Il percorso eseguito è lungo circa 73 metri. Vengono mostrati gli errori medi e massimi commessi lungo il percorso nella Tabella (4.4).

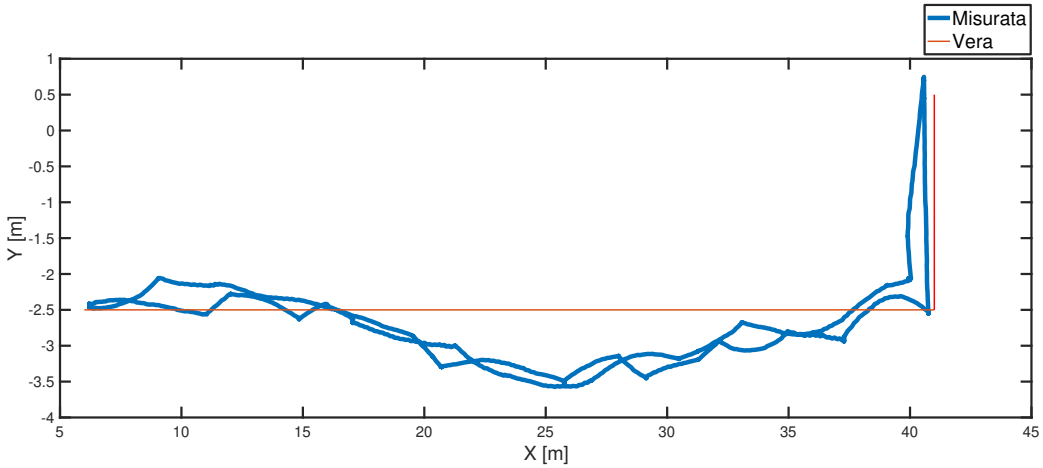


Figura 4.15: Percorso eseguito su mappa ottenuta con tecnica 3

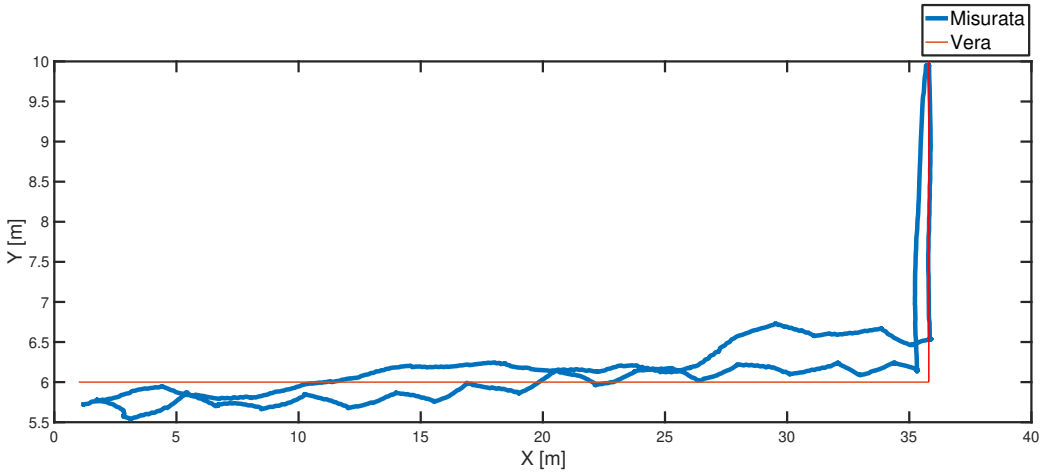


Figura 4.16: Percorso eseguito su mappa ottenuta da sensore Hokuyo

	Errore medio [m]	Errore max [m]
Mappa Tecnica 1	(0.5452, 0.2679)	(1.4955, 1.0657)
Mappa Tecnica 3	(0.5788, 0.4000)	(1.07,1.3276)
Mappa Hokuyo	(0.22, 0.18)	(0.4832, 0.7800)

Tabella 4.4: Errore medio ed errore massimo commessi lungo la traiettoria

Capitolo 5

Conclusioni e sviluppi futuri

Questa tesi ha presentato uno studio dei principali algoritmi di localizzazione presenti in letteratura, applicandoli poi al caso di studio del Pepper Robot. Pepper è un robot da compagnia/sociale progettato per consentire l'interazione cognitiva e fisica con gli esseri umani, e il cui software NAOqi può essere interfacciato con il sistema ROS.

In primo luogo l'algoritmo SLAM è stato utilizzato in fase di creazione delle mappe, utilizzando 3 tecniche che differiscono per l'uso dei sensori del robot. La prima tecnica utilizza tutti i 3 sensori laser del robot che sono posti sulla sua base. Le informazioni vengono raccolte in un unico topic dato in ingresso al pacchetto utilizzato. La seconda utilizza il sensore di profondità 3D (trasformando i dati immagini depth, con opportune tecniche, in dati di tipo laser), mentre la terza tecnica prevede l'uso della camera RGB, il sensore di profondità e il sensore laser che permettono di creare sia la mappa 3D dell'ambiente che quella 2D. Create le mappe, è emerso che l'utilizzo della seconda tecnica non ha generato una mappa utilizzabile per i successivi test ed è stata scartata. Viene poi presa in considerazione un'altra mappa, creata utilizzando un sensore esterno (laser Hokuyo), per confrontare la qualità delle mappe ottenute. Successivamente l'algoritmo Monte Carlo viene utilizzato per eseguire la localizzazione sulle mappe precedentemente create, eseguendo due test. Il primo test consiste nell'eseguire un percorso rettangolare con quattro punti intermedi, e si vanno a valutare il valor medio dell'errore e la deviazione standard sulle coordinate di questi punti. Il secondo test consiste nell'eseguire un percorso di andata e ritorno, e si valuta l'errore medio e massimo sull'intera traiettoria.

Dal primo test, confrontando i risultati ottenuti sulle varie mappe, si evince nella mappa creata dal laser Hokuyo un valor medio dell'errore maggiore e una deviazione standard minore rispetto alle altre mappe. Questo viene giustificato dalla presenza di un offset sulle misurazioni. Nel secondo test invece

si mostra una maggiore precisione nella mappa creata dal sensore Hokuyo, tuttavia, i risultati ottenuti dalle mappe create con il Pepper, presentano un errore non superiore a 1 m rispetto gli errori commessi usando la mappa creata con Hokuyo. I risultati ottenuti sono coerenti con quelli attesi, in quanto, la qualità della localizzazione complessiva è inferiore rispetto a quella ottenuta con sensori di fascia più alta, come in questo caso il sensore Hokuyo, ma comunque sufficiente per un utilizzo del robot in applicazioni dove viene richiesta autonomia per quanto riguarda localizzazione e navigazione. Infatti il robot Pepper rientra nella categoria dei robot sociali da compagnia e quindi non viene utilizzato in applicazioni industriali dove è richiesta una maggiore precisione.

Come detto in precedenza, creare la mappa all'interno di un ambiente in cui il robot si trova, è lo step precedente a quello della navigazione autonoma in quanto per la navigazione è necessario che il robot abbia alcune informazioni rilevanti sull'ambiente in cui si muove, che è a priori sconosciuto nella maggior parte delle applicazioni. La navigazione permette al robot di navigare e operare autonomamente e in sicurezza nell' ambiente di lavoro ed è la funzione principale di robot mobili autonomi: si costruisce un modello dell'ambiente attraverso l'osservazione dello stesso, e, con tale acquisizione, si pianifica di muoversi da un punto di partenza ad uno di arrivo. Anche la localizzazione è una delle competenze fondamentali richieste da un robot autonomo poiché la conoscenza della posizione del robot è un precursore essenziale per prendere decisioni sulle azioni future. Pertanto i problemi di mappatura, localizzazione e navigazione sono strettamente correlati tra loro. Essi, hanno ricevuto molta attenzione in letteratura e, attualmente, continuano ad essere aree di ricerca molto attive.

Bibliografia

[1] Kazuhiko Kawamura e Moenes Iskarous. «Trends in service robots for the disabled and the elderly». In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*. Vol. 3. IEEE. 1994, pp. 1647–1654.

[2] Bruno Siciliano e Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[3] Haixia Wang et al. «A practical service robot system for greeting guests». In: *IEEE Proceedings of the 31st Chinese Control Conference*. 2012, pp. 4997–5001.

[4] Maki K Habib e Yvan Baudoin. «Robot-assisted risky intervention, search, rescue and environmental surveillance». In: *International Journal of Advanced Robotic Systems,* 7.1 (2010), p. 10.

[5] EunKyoung Lee et al. «Elementary and middle school teachers', students' and parents' perception of robot-aided education in Korea». In: *EdMedia+ Innovate Learning*. Association for the Advancement of Computing in Education AACE. 2008, pp. 175–183.

[6] Rolf Dieter Schraft e Gernot Schmierer. *Service robots*. CRC Press, 2000.

[7] Takayuki Kanda e Hiroshi Ishiguro. *Human-robot interaction in social robotics*. CRC Press, 2017.

[8] Selma Musić e Sandra Hirche. «Classification of human-robot team interaction paradigms». In: *IFAC-PapersOnLine* 49.32 (2016), pp. 42–47.

[9] Sebastian Thrun, Wolfram Burgard e Dieter Fox. «A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping». In: *IEEE International Conference on Robotics and Automation. Symposia Proceedings*. Vol. 1. 2000, pp. 321–328.

[10] Maxim A Batalin, Gaurav S Sukhatme e Myron Hattig. «Mobile robot navigation using a sensor network». In: *IEEE International Conference on Robotics and Automation*. Vol. 1. IEEE. 2004, pp. 636–641.

[11] Luis Paya, Arturo Gil e Oscar Reinoso. «A state-of-the-art review on mapping and localization of mobile robots using omnidirectional vision sensors». In: *Journal of Sensors* 2017 (2017).

[12] Ingemar J Cox. «Blanche-an experiment in guidance and navigation of an autonomous robot vehicle». In: *IEEE Transactions on robotics and automation* 7.2 (1991), pp. 193–204.

[13] B Everett e L Feng. «Navigating mobile robots: Systems and techniques». In: *AK Peters, Ltd. Natick, MA, USA* (1996).

[14] Sean Philip Engelson. «Passive map learning and visual place recognition». Tesi di dott. Yale University, 1994.

[15] Hermann Endres, Wendelin Feiten e Gisbert Lawitzky. «Field test of a navigation system: Autonomous cleaning in supermarkets». In: *Proceedings. IEEE International Conference on Robotics and Automation*. Vol. 2. 1998, pp. 1779–1781.

[16] Wolfram Burgard et al. «The interactive museum tour-guide robot». In: *Aaai/iaai*. 1998, pp. 11–18.

[17] Reid Simmons e Sven Koenig. «Probabilistic robot navigation in partially observable environments». In: *IJCAI*. Vol. 95. 1995, pp. 1080–1087.

[18] Anthony R Cassandra, Leslie Pack Kaelbling e James A Kurien. «Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation». In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 2. 1996, pp. 963–972.

[19] Randall Smith, Matthew Self e Peter Cheeseman. «Estimating uncertain spatial relationships in robotics». In: *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[20] Wolfram Burgard et al. «Estimating the absolute position of a mobile robot using position probability grids». In: *Proceedings of the national conference on artificial intelligence*. 1996, pp. 896–901.

[21] Dieter Fox et al. «Position estimation for mobile robots in dynamic environments». In: *AAAI/IAAI* 1998 (1998), pp. 983–988.

[22] JE Handschin. «Monte Carlo techniques for prediction and filtering of non-linear stochastic processes». In: *Automatica,* 6.4 (1970), pp. 555–563.

[23] Neil J Gordon, David J Salmond e Adrian FM Smith. «Novel approach to nonlinear/non-Gaussian Bayesian state estimation». In: *IEE Proceedings F-radar and signal processing*. Vol. 140. 2. 1993, pp. 107–113.

[24] Genshiro Kitagawa. «Monte Carlo filter and smoother for non-Gaussian nonlinear state space models». In: *Journal of computational and graphical statistics,* 5.1 (1996), pp. 1–25.

[25] Michael Isard e Andrew Blake. «Condensation—conditional density propagation for visual tracking». In: *International journal of computer vision,* 29.1 (1998), pp. 5–28.

[26] Keiji Kanazawa, Daphne Koller e Stuart Russell. «Stochastic simulation algorithms for dynamic probabilistic networks». In: *arXiv preprint arXiv:1302.4965* (2013).

[27] Donald B Rubin. «Using the SIR algorithm to simulate posterior distributions». In: *Bayesian statistics,* 3 (1988), pp. 395–402.

[28] Daphne Koller e Raya Fratkina. «Using Learning for Approximation in Stochastic Processes.» In: *ICML*. 1998, pp. 287–295.

[29] Dieter Fox et al. «Monte carlo localization: Efficient position estimation for mobile robots». In: *AAAI/IAAI* 1999,343-349 (), pp. 2–2.

[30] John J Leonard e Hugh F Durrant-Whyte. *Directed sonar sensing for mobile robot navigation*. Vol. 175. Springer Science & Business Media, 2012.

[31] Tim Bailey e Hugh Durrant-Whyte. «Simultaneous Localisation and Mapping (SLAM) Part 2: State of the Art». In: *Robotics and Automation Magazine* (2006).

[32] M.W.M.G. Dissanayake et al. «A solution to the simultaneous localization and map building (SLAM) problem». In: *IEEE Transactions on Robotics and Automation,* 17.3 (2001), pp. 229–241.

[33] Sebastian Thrun, Wolfram Burgard e Dieter Fox. «A probabilistic approach to concurrent mapping and localization for mobile robots». In: *Autonomous Robots* 5.3 (1998), pp. 253–271.

[34] Brian Yamauchi, Alan Schultz e William Adams. «Mobile robot exploration and map-building with continuous localization». In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation*. Vol. 4. IEEE. 1998, pp. 3715–3720.

[35] Randall C Smith e Peter Cheeseman. «On the representation and estimation of spatial uncertainty». In: *The international journal of Robotics Research* 5.4 (1986), pp. 56–68.

[36] Hugh F Durrant-Whyte. «Uncertain geometry in robotics». In: *IEEE Journal on Robotics and Automation* 4.1 (1988), pp. 23–31.

[37] Nicholas Ayache e Olivier Faugeras. «Maintaining representations of the environment of a mobile robot». In: *IEEE transactions on Robotics and Automation* 5.6 (1989), pp. 804–819.

[38] Raja Chatila e Jean-Paul Laumond. «Position referencing and consistent world modeling for mobile robots». In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. IEEE. 1985, pp. 138–145.

[39] Wolfgang D Rencken. «Concurrent localisation and map building for mobile robots using ultrasonic sensors». In: *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*. Vol. 3. IEEE. 1993, pp. 2192–2197.

[40] Jae-Bok Song e Seo-Yeon Hwang. «Past and state-of-the-art SLAM technologies». In: *Journal of Institute of Control, Robotics and Systems* 20.3 (2014), pp. 372–379.

[41] T. Bailey e H. Durrant-Whyte. «Simultaneous localization and mapping (SLAM): part II». In: *IEEE Robotics Automation Magazine,* 13.3 (2006), pp. 108–117.

[42] Shoudong Huang e Gamini Dissanayake. «Robot localization: An introduction». In: *Wiley Encyclopedia of Electrical and Electronics Engineering* (1999), pp. 1–10.

[43] Sebastian Thrun et al. «Monte carlo localization for mobile robots». In: *In Proceedings of the IEEE International Conference on Robotics and Automation*. 1999.

[44] Sebastian Thrun et al. «Robust Monte Carlo localization for mobile robots». In: *Artificial intelligence,* 128.1-2 (2001), pp. 99–141.

[45] Martin A Tanner. *Tools for statistical inference*. Springer, 2012.

[46] Thomas L Dean e Mark S Boddy. «An Analysis of Time-Dependent Planning.» In: *AAAI*. Vol. 88. 1988, pp. 49–54.

[47] Anthony Stentz et al. «FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association». In: *In Proceedings of the AAAI National Conference on Artificial Intelligence*. 2003, pp. 593–598.

[48] M. Montemerlo e S. Thrun. «Simultaneous localization and mapping with unknown data association using FastSLAM». In: *IEEE International Conference on Robotics and Automation,* 2 (2003), 1985–1991 vol.2.

[49] Michael Montemerlo et al. «FastSLAM: A factored solution to the simultaneous localization and mapping problem». In: *Aaai/iaai* 593598 (2002).

[50] G. Dissanayake, H. Durrant-Whyte e T. Bailey. «A computationally efficient solution to the simultaneous localisation and map building (SLAM) problem». In: *IEEE International Conference on Robotics and Automation. Symposia Proceedings.* Vol. 2. 2000, pp. 1009–1014.

[51] Gibson Hu et al. «A robust RGB-D SLAM algorithm». In: *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2012, pp. 1714–1719.

[52] Michal Irani, Benny Rousso e Shmuel Peleg. *Recovery of ego-motion using image stabilization.* Leibniz Center for Research in Computer Science, Department of Computer ..., 1993.

[53] Wilhelm Burger e Bir Bhanu. «Estimating 3D egomotion from perspective image sequence». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.11 (1990), pp. 1040–1058.

[54] D. Nister, O. Naroditsky e J. Bergen. «Visual odometry». In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* Vol. 1. 2004, pp. I–I.

[55] Richard A Newcombe et al. «Kinectfusion: Real-time dense surface mapping and tracking». In: *10th IEEE international symposium on mixed and augmented reality.* 2011, pp. 127–136.

[56] Peter Henry et al. «RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments». In: *The International Journal of Robotics Research,* 31.5 (2012), pp. 647–663.

[57] Cedric Audras et al. «Real-time dense RGB-D localisation and mapping». In: *Australian Conference on Robotics and Automation.* 2011.

[58] Mathieu Labbé e François Michaud. «RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation». In: *Journal of Field Robotics* 36.2 (2019), pp. 416–446.

[59] Sagarnil Das. «Simultaneous Localization and Mapping (SLAM) using RTAB-Map». In: *arXiv preprint arXiv:1809.02989* (2018).

[60] Cyrill Stachniss, John J Leonard e Sebastian Thrun. «Simultaneous localization and mapping». In: *Springer Handbook of Robotics*. Springer, 2016, pp. 1153–1176.

[61] Sébastien Laniel et al. «Adding navigation, artificial audition and vital sign monitoring capabilities to a telepresence mobile robot for remote home care applications». In: *IEEE International Conference on Rehabilitation Robotics (ICORR)*. 2017, pp. 809–811.

[62] Henrique Foresti et al. *Emotive Robotics with I-Zak*. 2016.

[63] Yingfeng Chen et al. «KeJia-LC: a low-cost mobile robot platform—champion of demo challenge on benchmarking service robots at RoboCup 2015». In: *Robot Soccer World Cup*. Springer. 2015, pp. 60–71.

[64] Mathieu Labbé e François Michaud. «Memory management for real-time appearance-based loop closure detection». In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 1271–1276.

[65] João R. Silva et al. «Navigation and obstacle avoidance: a case study using Pepper robot». In: *45th Annual Conference of the IEEE Industrial Electronics Society*. Vol. 1. 2019, pp. 5263–5268.

[66] Fumihide Tanaka et al. «Pepper learns together with children: Development of an educational application». In: *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. 2015, pp. 270–275.

[67] Dario Allegra et al. «Experiences in Using the Pepper Robotic Platform for Museum Assistance Applications». In: *25th IEEE International Conference on Image Processing (ICIP)*. 2018, pp. 1033–1037.

[68] Mary Ellen Foster et al. «The MuMMER project: Engaging human-robot interaction in real-world public spaces». In: *International Conference on Social Robotics*. Springer. 2016, pp. 753–763.

[69] Iina Aaltonen et al. «Hello Pepper, may I tickle you? Children's and adults' responses to an entertainment robot at a shopping mall». In: *Proceedings of the Companion of the ACM/IEEE International Conference on Human-Robot Interaction*. 2017, pp. 53–54.

[70] Arkadiusz Gardecki et al. «The Pepper humanoid robot in front desk application». In: *IEEE Progress in Applied Electrical Engineering (PAEE)*. IEEE. 2018, pp. 1–7.

[71] Amit Kumar Pandey e Rodolphe Gelin. «A mass-produced sociable humanoid robot: Pepper: The first machine of its kind». In: *IEEE Robotics & Automation Magazine,* 25.3 (2018), pp. 40–48.

[72] *Pepper Datasheet 1.8a*. datasheet, ultimo accesso Giugno 2021. SoftBank Robotics. 2017. URL: https://pepper.generationrobots.com/Pepper%20Datasheet%201.8a%2020170116%20EMEA.pdf.

[73] Vittorio Perera et al. «Setting up pepper for autonomous navigation and personalized interaction with users». In: *arXiv preprint arXiv:1704.04797* (2017).

[74] Lentin Joseph e Jonathan Cacace. *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. Packt Publishing Ltd, 2018.

[75] *GMapping*. URL: http://wiki.ros.org/gmapping.

[76] Cristopher Gómez et al. «Visual SLAM-Based Localization and Navigation for Service Robots: The Pepper Case». In: *Robot World Cup XXII*. A cura di Dirk Holz et al. Cham: Springer International Publishing, 2019, pp. 32–44.

[77] *Rtab-Map*. URL: http://wiki.ros.org/rtabmap_ros.