UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA

Dipartimento di Informatica Sistemistica e Comunicazione

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Specialistica in Informatica

6DoF Monte Carlo Localization in a 3D world with Laser Range Finders

Relatore: Prof. Domenico G. Sorrenti
Correlatore: Dr. Axel Fúrlan

Candidato: *Augusto Luis Ballardini*
Matricola: 057431

Anno Accademico 2010-2011

Indice

Capitolo 1

Introduzione

Lo scopo del presente lavoro è creare un sistema di autolocalizzazione per robot mobili che lavori in un ambiente 3D e che determini i 6 parametri che caratterizzano il posizionamento di un corpo rigido nello spazio. Questo ha richiesto la definizione di un modello di moto probabilistico per la predizione del movimento. A sua volta ciò ha richiesto sia lo sviluppo di un sistema di simulazione nel quale effettuare i test sia una estesa attività sperimentale. Il lavoro rientra nell'ambito del progetto *Urban Shuttles Autonomously Driven* (USAD) sviluppato dal laboratorio IRA che si pone l'obiettivo di permettere ad un veicolo autonomo di effettuare la navigazione in ambito outdoor urbano, ad esempio all'interno del campus universitario.

La robotica mobile, ed in particolare i sistemi di guida autonoma, sono un settore di ricerca relativamente giovane ed in continuo sviluppo. In contrasto con il segmento della robotica industriale, prevalentemente legato a sviluppi relativi a bracci meccanici e nei quali la posizione nello spazio è facilmente calcolabile, i veicoli autonomi sono dotati di un sistema di movimentazione basato su ruote. La presenza delle ruote rende intrattabile il problema della localizzazione mediante il solo utilizzo dei dati odometrici ovvero i dati riguardanti lo spostamento del veicolo calcolati a partire dalle letture di opportuni sensori posizionati sulle ruote. Slittamenti delle singole ruote rispetto al suolo, dovuti alla tipologia del piano stradale, a particolari condizioni meteorologiche, a variazioni del diametro delle ruote, ad esempio dovuti a carichi del veicolo differenti, così come presenza di avvallamenti,

dossi e sconnessioni della pavimentazione, rendono essenziale l'utilizzo di algoritmi che determinino la posizione del veicolo usando sensori non propriocettivi. Va sottolineato che in ambito urbano il sistema GPS apparentemente una soluzione immediatamente disponibile, ha una affidabilità del tutto non adeguata allo svolgimento della navigazione. Sebbene lo stato dell'arte presenti diverse soluzioni per i problemi di localizzazione, tali soluzioni sono prevalentemente concepite per ambienti di robotica indoor, dove l'analisi del moto nella tridimensionalità dello spazio può essere semplificata a favore di una stima dello spostamento del robot limitatta al sul piano (pavimento) bidimensionale di moto; questo comporta un modello di moto probabilistico a 3DoF (i gradi di libertà di un corpo rigido), in due dimensioni. L'inadeguatezza di questa semplificazione in una situazione urban outdoor, verificata sia teoricamente che sul campo, ha richiesto lo sviluppo di un nuovo modello di moto probabilistico, basato sulla modellizzazione di un generico movimento nello spazio. Da questa fase di analisi si è quindi passati alla realizzazione di un prototipo funzionante in linguaggio C++ integrato con un framework robotic-oriented per la gestione delle diverse componenti hardware e software del robot utilizzato.

Nella robotica mobile la capacità di autolocalizzazione è alla base di ogni compito che possa essere assegnato ad un robot. La precisione con cui un robot è in grado di autolocalizzarsi all'interno di una mappa nota assume un ruolo cruciale nell'evitare collisioni con gli elementi strutturali dell'ambiente circostante e con gli oggetti statici e dinamici presenti nella scena; ciò è ancor più vero nel campo dell'autonomous driving, dove una corretta e precisa localizzazione risulta di fondamentale importanza in quanto viene messa in gioco la sicurezza degli individui. Il rischio di urti sia con con la parte statica dell'ambiente come marciapiedi, muri o guardrail, sia con la parte dinamica, ovvero altri veicoli, pedoni o animali, deve essere portato al minimo prevedendo una serie di procedure di emergenza da attuare nel caso di imprevisti. Al fine di ottenere una corretta localizzazione si fa uso di diversi sensori, capaci di percepire l'ambiente circostante, come laser range finder, telecamere o sonar. La scelta di utilizzo di una determinata tipologia di sensore viene fatta tenendo conto della quantità e della qualità di informazione che può essere estratta dai dati acquisiti, oltre che dal costo, peso e consumo del sensore.

Se un sistema basato su telecamere permette non solo di ricostruire lo spazio tridimensionale, pur in modo sparso, ma anche di riconoscere categorie di oggetti, le immagini catturate richiedono innanzitutto un certo numero di elaborazioni di basso livello atte a filtrare e/o segmentare la notevole mole di informazioni raccolte, prima di poter essere utilizzate da un sistema di alto livello. Queste operazioni si traducono nella pratica in trasformazioni negli spazi di colore, applicazione di operatori locali o globali e di filtraggio. Uno svantaggio di questi sistemi risiede nella variabilità della precisione con la quale si riescono ad estrarre informazioni dalle immagini acquisite, non costante rispetto alla profondità. Questo risulta essere un elemento particolarmente penalizzante negli ambienti outdoor, dove le distanze tra il robot e gli oggetti di interesse nel mondo sono solitamente elevate. I sonar, che producono letture molto rumorose e sono caratterizzati da un range di lavoro decisamente modesto, sono di fatto inutilizzabili per compiti complessi come la localizzazione outdoor e sono relegati al più semplice ruolo di sensori di prossimità.
Infine i sensori laser, o *Light Detection and Ranging* (LIDAR), rappresentano ad oggi lo stato dell'arte per quanto concerne l'affidabilità e la precisione delle misure prodotte. Nel lavoro di questa tesi è stata utilizzata quest'ultima tipologia di sensore. In particolare, in aggiunta a due LIDAR ad un piano di scansione (ad oggi un dispositivo standard) è stato possibile utilizzare un innovativo sensore che permette di effettuare contemporaneamente misure su quattro piani di scansione differenti; grazie a questa peculiarità è stato possibile, posizionando il sensore con un opportuno orientamento, percepire le inclinazioni del piano stradale. Anche per i LIDAR non si può ovviamente escludere la presenza di errori, dovuta sia a condizioni sfavorevoli dell'ambiente, come la presenza di elementi riflettenti o elementi inaspettati tipici di un ambiente dinamico, sia a misure casuali, inesatte a causa della risoluzione del sensore oppure errate a causa dell'apparente raggiungimento della massima portata.

Nell'ambito della robotica probabilistica le informazioni ottenute dai sensori vengono pre-elaborate attraverso delle procedure specifiche per il tipo di sensore utilizzato, volte a modellare la naturale presenza di questi errori. Nel caso di sensori di tipo range-sensor come sonar e LIDAR i modelli sono

definiti attraverso una mistura di distribuzioni di probabilità, condizionate dalla posizione del robot nell'ambiente in un certo istante di tempo. In questo lavoro abbiamo preso spunto dal modello proposto da Fox *et al.* [1] che utilizza come tipologia di errore una mistura tra errori derivanti da: piccolo rumore di misura, errori dovuti ad oggetti imprevisti nella scena, errori dovuti a mancate rilevazioni di ostacoli ed infine un rumore casuale di fondo. L'approccio probabilistico viene applicato non solo ai modelli sensoriali, ma anche al modo in cui la posizione del robot viene determinata. Nel lavoro di tesi è stata considerata una specifica categoria di algoritmi che permettono di affrontare in modo efficace e robusto i problemi di localizzazione sia locale che globale, si veda [2]. Questi algoritmi, che vanno sotto il nome di *Monte Carlo Localization* (MCL), nonostante la loro recente introduzione, sono diventati la tecnica di localizzazione più utilizzata in robotica grazie alle loro capacità di adattamento alle più diverse forme di distribuzione e quindi di generazione di ipotesi accettabili; l'adattamento viene effettuato attraverso l'introduzione di campioni che rappresentano in modo non parametrico una generica distribuzione. Gli algoritmi MCL richiedono tipicamente un gran numero di risorse, viste in termini di numero di campioni necessari, affinché possano effettuare correttamente una localizzazione globale, mentre le localizzazioni locali sono effettuate in modo efficace anche con un utilizzo di risorse molto minore. In questo lavoro abbiamo preso spunto dal metodo proposto da Dieter Fox [3] [4] che permette un adattamento dinamico del numero di campioni in base al tipo di distribuzione ad ogni istante di tempo.

La difficoltà affrontata dagli algoritmi di localizzazione globale può essere intuita se pensiamo al problema di individuare la nostra posizione lungo un corridoio con porte equidistanti ed indistinguibili tra di loro e senza punti di riferimento aggiuntivi. Tale sfida, che creerebbe difficoltà anche ad una persona, viene resa ancora più ardua dall'impossibilità di avere a disposizione una rilevazione completa del mondo circostante, dovendoci invece basare su viste dell'ambiente estremamente limitate. Possiamo rivedere questo esempio in una forma più vicina a noi. Supponiamo di dover localizzare una macchina all'interno di un parcheggio multipiano dove solitamente ogni piano è uguale agli altri. Se non conoscessimo il piano di partenza dal quale iniziare la ricerca ovvero senza informazioni aggiuntive che ci permettano di capire la

nostra posizione iniziale, la corretta individuazione della posizione del veicolo risulterebbe estremamente onerosa.

Il problema viene affrontato creando diverse ipotesi sull'effettivo posizionamento del robot che vengono man mano scartate sulla base di acquisizioni sensoriali successive, integrate nel tempo. Affinché gli algoritmi di localizzazione possano aggiornare in modo corretto l'ipotesi, o le ipotesi, effettuate vengono introdotti, sempre in termini probabilistici, dei modelli di moto che hanno il compito di regolare l'evoluzione delle ipotesi, sulla base delle informazioni odometriche. Questo procedimento, che si diversifica a seconda della cinematica del robot utilizzato, assume un ruolo cruciale in ogni algoritmo di localizzazione in quanto è il principale responsabile dell'accrescimento dell'incertezza sulle coordinate che rappresentano lo stato del robot. In letteratura sono presenti diversi approcci per la gestione del modello di moto e si distinguono sia per la tipologia di sensori propriocettivi utilizzati sia, ed in particolar modo, per il tipo di moto modellato. Il moto nello spazio richiede una trattazione diversa rispetto al moto nel piano in quanto lo stato, che identifica il posizionamento del robot all'interno di una mappa, viene ampliato considerando sia la coordinata di quota sia l'assetto in termini di rollio e beccheggio.
In questa tesi proponiamo un modello di moto probabilistico a sei gradi di libertà ispirato dal modello a tre gradi di libertà proposto da Thrun, Fox e Burgard [5], in grado di generare stime di posizione ed orientamento integrando dati proveniente da sensori eterogenei ed introducendo una parametrizzazione atta a prevedere la mancanza delle informazioni non fornite dai sensori odometrici basati su ruote. Il lavoro è stato svolto nelle seguenti fasi:

1. La fase iniziale del progetto è stata dedicata alla predisposizione del veicolo utilizzato in questa tesi. Sono stati sviluppati i driver necessari per la comunicazione con i sensori LIDAR ed è stata fatta una recensione dei framework robotici orientati alla comunicazione tra le diverse componenti del veicolo. E' stata inoltre necessaria la creazione di opportuni supporti metallici atti ad ancorare i LIDAR al veicolo in modo stabile, evitando l'introduzione di oscillazioni dovute all'elasticità del materiale utilizzato. Abbiamo scelto il framework denominato *ROS* -

Robotic Operating System Quigley *et al.* [6] per via dell'ottimo supporto per una serie di robot già presenti sul mercato così come per la disponibilità di pacchetti software per la comunicazione con alcuni sensori e la grande partecipazione della community mondiale in questo progetto open source. Inoltre, sempre nell'ambiente ROS, è disponibile un ambiente grafico di visualizzazione 3D che ha facilitato considerevolmente il debug del codice e degli algoritmi proposti in questa tesi.

2. Una volta preparati e resi operativi veicolo, sensoristica ed hardware, siamo passati ad una prima fase di *demo*, svolta inizialmente nel garage dell'edificio U5 e nel parcheggio esterno dell'edificio U5 per poi passare in Piazza della Scienza tra gli edifici U1 ed U2. I test, effettuati con successo, ci hanno però persuasi dell'assoluta necessità di una localizzazione 6DoF in un ambiente 3D. Le problematiche riscontrate sono legate principalmente alla portata elevata dei LIDAR. Variazioni di assetto nell'ordine di pochi gradi ed avvallamenti dell'ambiente hanno fatto sì che i LIDAR fossero in realtà inclinati diversamente da quanto prevedibile con un semplice moto piano, ad esempio generando una misura di distanza dal piano stradale anziché dagli ostacoli presenti su di esso. L'adozione di un ambiente 3D a 6DoF ci permette inoltre il trattamento di situazioni estreme come la salita da un parcheggio sotterraneo effettuata per mezzo di una rampa, situazione che ha messo in crisi il modello 3DoF di Thrun *et al.* [5] in quanto non è stato possibile individuare una parametrizzazione capace di gestire notevoli variazioni nei livelli di beccheggio.

3. Sono state analizzate diverse tipologie di modelli di moto e di modelli sensoriali pensati per ambienti outdoor, sia bidimensionali che tridimensionali (si vedano Sakai*et al.* [7], Kümmerle *et al.* [8], Petrovskaya *et al.* [9], Fox *et al.* [10]) riscontrando che i modelli di moto proposti non trattano in modo completo o esplicito l'incertezza di tutte le componenti della pose 6DoF. Abbiamo deciso di iniziare il lavoro utilizzando i modelli bidimensionali proposti da Thrun *et al.* [5] e disponibili, seppure con diverse modifiche non del tutto documentate, nell'ambiente

ROS. Abbiamo quindi svolto una fase di studio di queste modifiche proponendo un nuovo approccio su diverse sottoparti dell'algoritmo.

4. Nella fase successiva abbiamo affrontato la parte centrale della tesi ovvero la definizione di un nuovo modello di moto a partire dal modello bidimensionale proposto da Thrun *et al.* [5]. La trattazione dell'argomento da parte degli autori non fornisce però una spiegazione dettagliata sulle motivazioni delle scelte da loro effettuate e questo ha fatto sì che la generazione del modello proposto sia stata corredata da diversi tentativi di modellizzazione e di implementazione falliti. Abbiamo quindi affrontato il problema dal punto di vista teorico, sviluppando una estensione geometricamente e probabilisticamente corretta. La creazione del nuovo modello ha richiesto inoltre un notevole sforzo a causa dei requisiti di geometria tridimensionale.

5. Successivamente, per testare il modello generato, abbiamo creato un ambiente di simulazione nel quale è stato possibile identificare e valutare l'impatto dei diversi parametri che regolano il comportamento del modello.

6. Affinché fosse possibile integrare il nostro modello nel sistema ROS è stato necessario l'adattamento delle strutture software al nuovo tipo di ambiente tridimensionale. L'integrazione delle procedure matematiche atte ad eseguire le trasformazioni di rotazione e traslazione è stata effettuata attraverso l'utilizzo della libreria open source BulletPhysics. Per quanto concerne il trattamento delle mappe, le strutture sono state adeguate per la gestione della componente di quota andando a definire una mappa formata da *Voxel* tridimensionali.

7. Una volta preparato il software per il supporto di un mondo tridimensionale con i rispettivi assetti (vale a dire il posizionamento sia nelle componenti spaziali x, y e z sia nelle componenti di rollio, beccheggio ed imbardata) abbiamo creato una procedura atta alla ricostruzione di mappe tridimensionali a partire da disegni CAD. Questa procedura sarà in futuro sostituita da algoritmi di mapping 3D che facciano uso di sensori LIDAR, con il fine di poter ottenere ricostruzioni ad alta fedeltà

anche di parti del mondo per le quali non è possibile pensare di ottenere informazioni da un disegno CAD (come ad esempio la reale pendenza della pavimentazione oppure aree per le quali non esiste un disegno). I software open source utilizzati per l'interpretazione e la trasformazione dei disegni CAD in mappe 3D fatte di Voxel appartengono alla suite BinVox. E' stato inoltre creato un ambiente di simulazione nel quale tutte le componenti del progetto sono state messe alla prova.

8. Infine abbiamo effettuato i test direttamente sul veicolo nei pressi dell'edificio U5, dimostrando la correttezza del modello odometrico sviluppato e gli obiettivi di autolocalizzazione che ci eravamo prefissati all'inizio del progetto.

Il lavoro svolto è stato efficacemente impiegato in diverse situazioni pubbliche come la fiera *Electrical Intelligent Vehicles 2010* e la demo del programma televisivo di Rai3 *Buongiorno Regione* effettuata nel piazzale tra gli edifici U1 ed U2 (Piazza della Scienza). Dal punto di visto della ricerca scientifica il lavoro è di massima attualità e, al meglio delle nostre conoscenze, è quanto di più avanzato sia stato sviluppato nel suo genere, tanto per correttezza scientifica che per efficacia. E' in fase di stesura un articolo sul lavoro svolto da sottomettere alla principale conferenza del settore.

Presenteremo il lavoro secondo il seguente ordine: nel Capitolo 2 ci occuperemo dell'analisi dello stato dell'arte nell'ambito della localizzazione della robotica mobile; nel Capitolo 3 presenteremo il modello di moto ideato nel lavoro di questa tesi ed utilizzato dall'algoritmo di localizzazione sviluppato; nel Capitolo 4 presenteremo il vero e proprio algoritmo di localizzazione, gli accorgimenti necessari per trattare mappe tridimensionali ed esporremo le scelte implementative delle varie componenti scritte in linguaggio C++; nel Capitolo 4.5 descriveremo la configurazione del veicolo utilizzato in questa tesi; presenteremo gli ambienti di simulazione utilizzati ed i risultati sperimentali; nel Capitolo 5 trarremo le conclusioni del progetto; infine nelle appendici verranno trattate in dettaglio le procedure tecniche descritte in questa tesi.

Capitolo 2

Stato dell'arte

In questo capitolo analizzeremo lo stato dell'arte fornendo un quadro sulle tecniche di robotica probabilistica. I robot outoor, ideati per essere a contatto con il mondo reale, ovvero ambienti non progettati ad-hoc quali possono essere le catene di montaggio, hanno un grande svantaggio rispetto ai robot progettati per lavorare in ambienti indoor, ovvero l'intrinseca imprevedibilità dell'ambiente in cui i compiti di questi robot vengono eseguiti. La dinamicità di un ambiente casalingo oppure di un ambiente outdoor rendono necessario gestire l'enorme grado di incertezza esistente in questo tipo di ambienti. Le fonti di errore riguardano inoltre sia la parte sensoristica, ovvero i sensori che vengono utilizzati per percepire l'ambiente stesso, sia la parte di attuazione del movimento. Ulteriori incertezze sono introdotte anche a livello di software, dovute ad approssimazioni ed astrazioni della fisica attraverso modelli semplificati atti a consentire le operazioni in un sistema real-time, solitamente limitato nella quantità di risorse di computazione. In riferimento al lavoro di tesi verrà fornita una panoramica sulle tecniche di localizzazione e di stima del moto in ambito probabilistico.

2.1 I filtri

Introduciamo innanzitutto il concetto di filtro [1]. Nella robotica probabilistica i dati sensoriali del robot, il suo stato e l'ambiente circostante vengono rappresentati mediante variabili aleatorie in spazi continui; ad ogni variabile è possibile assegnare una distribuzione di probabilità (probability density function, PDF). Alcuni esempi di PDF sono la distribuzione uniforme

$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{se } a \leqslant x \leqslant b \\ 0, & \text{altrimenti} \end{cases}$$

e la distribuzione normale (gaussiana) con media μ e varianza σ^2

$$p(x) = \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}$$

Quest'ultima gioca un ruolo di estrema importanza per tutta la famiglia dei filtri gaussiani sia nella forma mono-dimensionale appena vista che in quella n-dimensionale

$$p(x) = \det\left(2\pi\Sigma\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$$

Con le distribuzioni di probabilità possiamo rappresentare i dati sensoriali, lo stato del robot e l'ambiente circostante come delle 'nuvole' di incertezza in uno spazio n-dimensionale.

Supponiamo ora di avere a disposizione questo insieme di 'nuvole' per ogni tempo $t \in \{T_0, T\}$. Possiamo chiederci, per esempio, dato lo stato del robot ed i suoi dati sensoriali (ad esempio i dati odometrici) al tempo $t = T$, in quale stato esso potrebbe trovarsi al tempo successivo. Facciamo quindi una **predizione** sul possibile stato del robot utilizzando un modello di moto, ovvero un modello fisico che esprime le caratteristiche cinetiche di un oggetto. Una volta giunti al tempo $t + 1$, possiamo utilizzare i nuovi dati dei sensori del robot per **aggiornare** la nostra predizione attraverso un modello delle misure ed il modello di moto.

Abbiamo appena definito il ruolo di un **filtro**, di cui una possibile formalizzazione è data dal **filtro di Bayes**: il filtro di Bayes è un filtro ricorsivo,

[1] Questa sezione è interamente ripresa da [5]

cioè la distribuzione di probabilità al tempo t viene stimata da quella al tempo $t-1$, pertanto richiedendo la markovianità delle variabili di stato. Fatto ciò, il filtro aggiorna lo stato integrando le misure dell'ambiente circostante.

Sistemi markoviani

Un processo stocastico markoviano o processo di Markov è un processo nel quale la probabilità della transizione, che determina il passaggio ad uno stato del sistema, dipende unicamente dallo stato del sistema immediatamente precedente e non dall'insieme degli stati passati.

In robotica, un sistema è detto markoviano se in esso vige l'assunzione di markovianità. Tale assunzione postula appunto che dati e stati passati e futuri sono completamente inutili se si è a conoscenza dei dati e dello stato attuale.

2.1.1 Filtri gaussiani

Storicamente, i filtri gaussiani rappresentano le primissime implementazioni del generico filtro di Bayes per i domini continui e sono di gran lunga la famiglia di filtri più popolare in robotica mobile. Essi devono il loro nome al fatto che modellano tutte le probabilità con distribuzioni normali multivariate.

Questa scelta comporta alcune grosse limitazioni. Innanzitutto le distribuzioni normali sono **unimodali**, cioè possiedono un unico massimo. Tale assunzione porta a buoni risultati quando il tracking si concentra su un singolo oggetto e la probabilità viene concentrata in un punto con un (piccolo) margine di incertezza.

Vediamo ora più in dettaglio i due principali membri della famiglia dei filtri gaussiani, il **filtro di Kalman** ed il **filtro di Kalman esteso**.

Il filtro di Kalman

Il filtro di Kalman è probabilmente la tecnica più studiata e più diffusa per implementare un filtro di Bayes [5]. Esso fu inventato negli anni '50 da **Rudolph Emil Kalman** quale tecnica di filtraggio e predizione nei sistemi lineari. Il filtro di Kalman implementa il calcolo delle probabilità per stati

continui, non è quindi applicabile a spazi discreti o ibridi. Le probabilità vengono rappresentate sotto forma di momenti: al tempo t la probabilità è rappresentata dalla media μ_t e dalla covarianza Σ_t. Le probabilità a posteriori possono essere anch'esse rappresentate come gaussiane se valgono, in aggiunta all'assunzione di markovianità del filtro di Bayes, le seguenti tre proprietà:

1. La probabilità dello stato al tempo successivo deve essere una funzione lineare dei suoi argomenti, con l'aggiunta di rumore gaussiano. Quanto detto si esprime con la seguente formula

   $$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

   dove x_t ed x_{t-1} sono vettori di stato di dimensione n ed u_t è il vettore di controllo di dimensione m. A_t e B_t sono matrici di dimensioni $n \times n$ ed $n \times m$ rispettivamente. Moltiplicando i due vettori per le matrici A e B rispettivamente, la funzione di transizione di stato diviene lineare nei suoi argomenti. Da qui deriva la linearità del filtro di Kalman. Infine, ε è un vettore gaussiano che modella la casualità nella transizione di stato. Esso ha le stesse dimensioni del vettore di stato, ha media nulla ed una matrice di covarianza R_t che verrà utilizzata per calcolare la probabilità dello stato $p(x_t|u_t, x_{t-1})$ a posteriori dell'azione di controllo con una distribuzione normale multivariata (vedi sopra).

2. Anche la probabilità delle misure $p(z_t|x_t)$ deve essere lineare nei suoi argomenti con l'aggiunta di rumore gaussiano

   $$z_t = C_t x_t + \delta_t$$

   dove C è una matrice $k \times n$ essendo k la dimensione del vettore delle misure. Come sopra, δ_t è un vettore casuale gaussiano con media nulla e matrice di covarianza Q_t utilizzata per calcolare la probabilità delle misure $p(z_t|x_t)$ con una distribuzione normale multivariata.

3. Infine, la distribuzione iniziale deve anche essa essere una distribuzione normale multivariata con media μ_0 e matrice di covarianza Σ_0.

1 **Input**: $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2 $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3 $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4 $K_t = \bar{\Sigma}_t C_t^T \left(C_t \bar{\Sigma}_t C_t^T + Q_t \right)^{-1}$
5 $\mu_t = \bar{\mu}_t + K_t \left(z_t - C_t \bar{\mu}_t \right)$
6 $\Sigma_t = \left(I - K_t C_t \right) \bar{\Sigma}_t$
7 **restituisci** μ_t, Σ_t

Tabella 2.1: *Formulazione algoritmica del filtro di Kalman per funzioni di transizione di stato e misure Gaussiane.*

Queste tre assunzioni sono sufficienti a garantire che la distribuzione di probabilità a posteriori rimanga sempre una distribuzione normale multivariata, a qualsiasi istante di tempo t.

Nella Tabella 2.1 è presentata la formulazione algoritmica del filtro di Kalman, mentre in Figura 2.1 possiamo vedere un esempio di evoluzione nel tempo della probabilità dello stato di un robot con l'utilizzo di un filtro di Kalman.

Il filtro di Kalman esteso

Nel mondo reale si verifica molto raramente che siano soddisfatte le assunzioni che stanno alla base del filtro di Kalman, cioè la linearità con aggiunta di rumore gaussiano delle funzioni di transizione di stato e delle misure. Per fare un esempio semplice, basti pensare ad un robot che si muova con velocità traslazionale e rotazionale costanti. Tale moto descrive una traiettoria circolare che non può essere descritta da una funzione di transizione di stato lineare. Alla luce di questa osservazione e dell'assunzione di distribuzioni di probabilità unimodali, possiamo dire che il filtro di Kalman semplice, per come descritto al paragrafo precedente risulta inapplicabile in quasi tutti gli ambienti reali salvo pochi casi specifici.

Il filtro di Kalman esteso (EKF) supera una di queste assunzioni: l'assunzione di linearità. In questo caso l'assunzione è che le funzioni di transizione di stato e delle misure siano governate da funzioni non lineari g ed h,

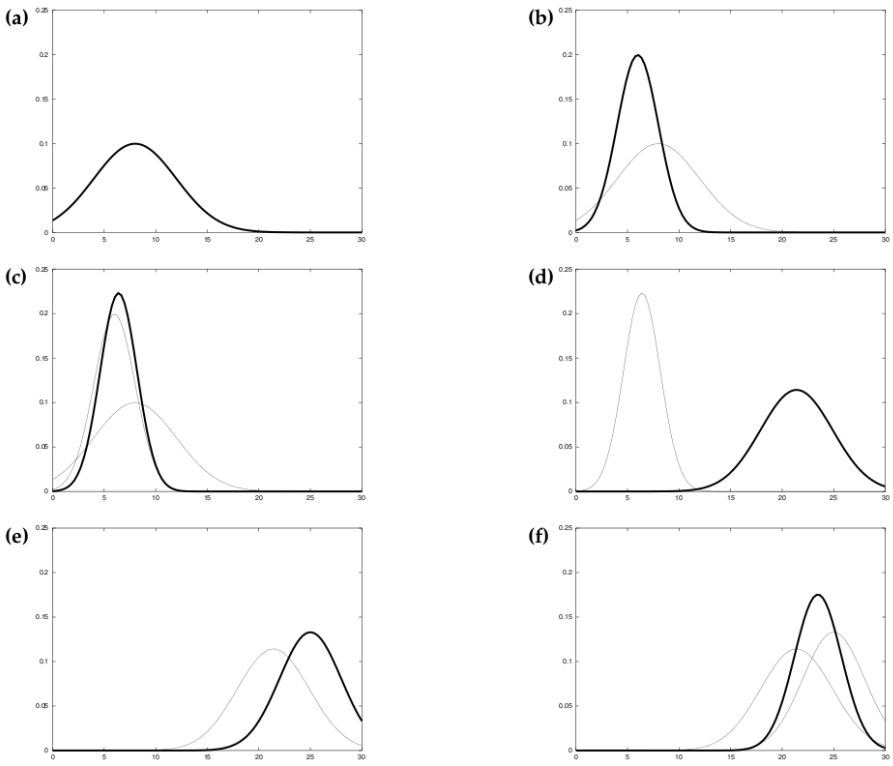


Figura 2.1: *Evoluzione nel tempo di un filtro di Kalman: (a) distribuzione iniziale di probabilità, (b) una misura (in grassetto) con l'incertezza ad essa associata, (c) probabilità dopo aver integrato la misura nella probabilità iniziale utilizzando l'algoritmo del filtro di Kalman, (d) probabilità dopo uno spostamento a destra che introduce un certo grado di incertezza, (e) una nuova misura con la relativa incertezza, ed (f) la probabilità risultante. Questa figura è stata presa da [5].*

rispettivamente:

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \qquad\qquad z_t = h(x_t) + \delta_t$$

Questo modello generalizza il modello gaussiano lineare alla base dei filtri di Kalman. La funzione g sostituisce le matrici A_t e B_t ed h sostituisce la C_t. Sfortunatamente, con g ed h arbitrarie, la distribuzione di probabilità non è più una gaussiana. Di fatto, è praticamente impossibile eseguire il passo di aggiornamento della probabilità per funzioni g e h non lineari, nel senso che il filtro di Bayes non possiede una soluzione in forma chiusa.

Il filtro di Kalman esteso calcola un'approssimazione della distribuzione reale e rappresenta tale approssimazione tramite una gaussiana. In particolare, la probabilità al tempo t viene rappresentata da una media μ_t e da una matrice di covarianza Σ_t. Quindi l'EKF eredita dal filtro di Kalman semplice la rappresentazione della probabilità, ma, a differenza del primo, questa probabilità è solamente un'approssimazione della reale distribuzione.

L'intuizione alla base del filtro di Kalman esteso è detta linearizzazione. Supponiamo di avere una funzione di transizione di stato g non lineare. Una gaussiana, proiettata attraverso questa funzione, tipicamente non rimane tale. Ciò è dovuto al fatto che la non linearità di g distorce la distribuzione di probabilità storpiandone la graziosa forma gaussiana. La linearizzazione approssima la g con una funzione lineare tangente a g nel punto medio della gaussiana. Proiettando ora la gaussiana attraverso questa approssimazione lineare, la probabilità a posteriori ritorna ad essere una gaussiana. Una volta in possesso di questa approssimazione, il meccanismo di propagazione della probabilità è equivalente a quello del filtro di Kalman semplice. Per quel che riguarda la funzione h il discorso è identico: l'EKF approssima la h con una funzione lineare tangente ad h, riacquisendo la gaussianità del risultato della proiezione.

Nella Tabella 2.2 possiamo vedere la formulazione algoritmica del filtro di Kalman esteso.

Il filtro di Kalman esteso è divenuto il più popolare strumento per la stima dello stato in robotica [11] [12] [13] [14]. La sua forza risiede nella sua semplicità e nella sua efficienza computazionale. Infatti, ogni passo di

1 **Input**: $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2 $\bar{\mu}_t = g(u_t, \mu_{t-1})$
3 $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4 $K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t\right)^{-1}$
5 $\mu_t = \bar{\mu}_t + K_t \left(z_t - h(\bar{\mu}_t)\right)$
6 $\Sigma_t = \left(I - K_t H_t\right) \bar{\Sigma}_t$
7 **restituisci** μ_t, Σ_t

Tabella 2.2: *Formulazione algoritmica del filtro di Kalman esteso.*

aggiornamento richiede un tempo $O(k^{2.8} + n^2)$, dove k è la dimensione del vettore delle misure ed n è la dimensione del vettore di stato x_t.

2.1.2 Filtri non parametrici

Come abbiamo detto nella sezione precedente, i filtri gaussiani soffrono dell'unimodalità delle distribuzioni normali. Ad esempio, in ambiti nei quali l'ipotesi di posizionamento di un oggetto sono molteplici, ognuna dei quali forma la propria moda nella distribuzione di probabilità a posteriori, questi filtri non sono in grado di rappresentare la corretta distribuzione di probabilità.

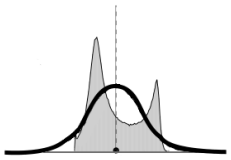


Figura 2.2

Un'alternativa molto diffusa per ovviare a questo problema sono i **filtri non parametrici**. I filtri di questa famiglia non si basano su una forma funzionale fissa per rappresentare la distribuzione di probabilità a posteriori, ma cercano di rappresentare al meglio tale distribuzione con un numero finito di valori. Il numero di tali valori è variabile e, idealmente, per tale numero che tende ad infinito, le tecniche non parametriche tendono a convergere alla corretta distribuzione di probabilità a posteriori.

Le tecniche non parametriche non impongono restrittive assunzioni sulle PDF a posteriori e quindi ben si prestano a rappresentare complesse probabilità multimodali. Queste qualità vanno però a pesare sulla complessità computazionale, rendendo necessario adattare la cardinalità dell'insieme di valori in base a delle stime sulla complessità della PDF a posteriori. Vediamo nel dettaglio il più noto esponente di questa famiglia, il **filtro a particelle**.

Il filtro a particelle

Il filtro a particelle è un'implementazione alternativa, non parametrica, del filtro di Bayes. L'idea fondamentale del filtro a particelle è di rappresentare la distribuzione di probabilità a posteriori con un numero finito di campioni casuali dello stato, campionati dalla probabilità stessa. Invece di rappresentare una distribuzione in forma parametrica (ad esempio la funzione esponenziale che definisce la PDF di una distribuzione normale), il filtro a particelle la rappresenta con un insieme di campioni, quindi con un'approssimazione. Tale approssimazione è però non parametrica, il che permette di rappresentare un spazio di funzioni molto più ampio di quanto possa fare una distribuzione normale. Per questo motivo il filtro a particelle viene ampiamente usato nei sistemi di localizzazione e di object tracking [15] [16] [17] [18] [19] [20].

Nei filtri a particelle i campioni di una distribuzione a posteriori vengono detti, appunto, particelle e si denotano

$$X_t := x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}$$

Ogni particella $x_t^{[m]}$ (con $1 \leq m \leq M$) è un'istanza dello stato al tempo t, cioè un'ipotesi di come il mondo reale potrebbe essere al tempo t. Qui M denota il numero di particelle presenti nell'insieme X_t ed è tipicamente un numero abbastanza grande, p.e. $M = 1000$. In alcune implementazioni M è funzione del tempo t o di altre quantità correlate alla distribuzione di probabilità. L'intuizione che sta alla base di un filtro a particelle è di approssimare la distribuzione con un insieme di particelle X_t. Idealmente, la probabilità di un'ipotesi di stato x_t che deve essere inclusa nell'insieme X_t dovrebbe essere proporzionale alla probabilità a posteriori del filtro di Bayes:

$$x_t^{[m]} \sim p(x_t | z_{1:t}, u_{1:t})$$

Di conseguenza, tanto più densamente è popolata da particelle una sottoregione dello spazio degli stati, tanto più è probabile che lo stato reale del mondo cada in questa regione. Poichè la tecnica delle particelle si basa su un'approssimazione della distribuzione di probabilità, la proprietà di proporzionalità vista poc'anzi verrebbe soddisfatta solo asintoticamente per $M \to \infty$ per l'algoritmo standard del filtro a particelle. Per M numero finito, le particelle vengono campionate da una distribuzione leggermente differente da quella originale; in pratica, però, tale differenza risulta ininfluente finchè M rimane un numero sufficientemente grande (p.e. $M > 100$) rispetto alla complessità della PDF che si vuole approssimare.

Esattamente come tutti gli altri filtri di Bayes discussi finora, anche l'algoritmo del filtro a particelle costruisce la distribuzione di probabilità dello stato $p(x_t)$ in modo ricorsivo dalla distribuzione $p(x_{t-1})$ al tempo precedente. Dal momento che le distribuzioni di probabilità qui vengono rappresentate da insiemi di particelle, quanto detto si traduce nel fatto che i filtri a particelle costruiscono l'insieme di particelle X_t ricorsivamente dall'insieme X_{t-1}. Nella tabella 2.3 possiamo vedere la formulazione algoritmica di un filtro a particelle standard.

Uno dei problemi di questa categoria di filtri risiede nella natura discreta di questo algoritmo. Accade talvolta, a causa della natura stessa di questi filtri, che le particelle non vengano posizionate in una area nelle vicinanze dello stato corretto. Questo fenomeno è chiamato *particle deprivation* e si verifica usualmente quando il numero di particelle non è in grado di coprire tutte le regioni con alta verosimiglianza. Il fenomeno è dovuto ad una scelta non ottimale delle particelle fatta dall'algoritmo di campionamento che potrebbe, a seguito ad una sfortunata serie di estrazioni, cancellare completamente le particelle nelle vicinanze dello stato $p(x_t)$ corretto. Una parziale soluzione a questo problema consiste nella generazione di particelle casuali. Altre soluzioni più raffinate riguardano la modifica delle modalità di campionamento [21].

Input: $(\mathcal{X}_{t-1}, u_t, z_t)$
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ *to* M **do**
    campiona $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = p(z_t|x_t^{[m]})$
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
for $m = 1$ *to* M **do**
    campiona i con probabilità $\propto w_t^{[i]}$
    aggiungi $x_t^{[i]}$ a $\mathcal{X}_t$
restituisci $\mathcal{X}_t$

Tabella 2.3: *Formulazione algoritmica del filtro a particelle.*

2.2 Robot Localization

La localizzazione è alla base di ogni compito che possa essere affidato ad un robot mobile e rappresenta il problema della determinazione della posizione di un robot relativamente ad una mappa nota di un ambiente. Il problema può essere visto come un lavoro di trasformazioni di coordinate. Le mappe sono descritte in termini di sistemi di riferimento globali, indipendenti dalla posizionamento del robot. La localizzazione consiste nella procedura di determinazione della corrispondenza tra le coordinate della mappa e lo coordinate locali del robot. Tali corrispondenze permettono al robot di individuare la posizione di oggetti all'interno della mappa in riferimento al proprio sistema di coordinate, operazione fondamentale per la navigazione. E' grazie a queste corrispondenze che un robot mobile può calcolare una traiettoria da seguire per muoversi evitando ostacoli noti nella mappa ed aggiornando correttamente la stessa con i dati di nuove rilevazioni. Dato che la conoscenza della posizione del robot non è solitamente osservabile in modo diretto (in quanto se cosi fosse non ci porremmo il problema della localizzazinoe) essa deve essere dedotta dalle osservazioni sensoriali, informazioni raccolte da sensori di loro natura soggetti ad errori di rilevazione ed approssimazioni. Inoltre, una singola acquisizione di dati non è solitamente sufficiente per la determinazione

Input: $(bel(x_{t-1}, u_t, z_t, m))$
for *all* x_t **do**
  $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m)bel(x_{t-1})dx_{t-1}$
  $bel(x_t) = \eta p(z_t|x_t, m)bel(x_t)$
restituisci $bel(x_t)$

Tabella 2.4: *Localizzazione Markoviana.*

della posizione che deve essere integrata nel tempo mediante una sequenza di atti di moto e di osservazioni sensoriali. Gli algoritmi di localizzazione sono suddivisi in due macro categorie denominate localizzazione locale e localizzazione globale. Nell'ambito della localizzazione locale viene data per conosciuta una posizione iniziale del robot; questo ci riconduce al problema noto in letteratura come *position tracking*. La conoscenza di una posizione iniziale permette di poter localizzare il robot nel tempo mediante l'integrazione dell'incertezza sul moto effettuato. Siccome tale incertezza è considerata comunemente piccola, vengono spesso utilizzate distribuzioni di probabilità unimodali. Quando una posizione iniziale non viene data il problema della localizzazione viene chiamato localizzazione globale, da intendersi come all'interno della mappa fornita. La mancanza di una area delimitata all'interno della quale effettuare una stima porta all'uso di distribuzioni multimodali, generalmente rappresentate da campioni o discretizzazioni. Gli algoritmi di localizzazione globale permettono inoltre di poter affrontare il problema del *kidnapped robot*. Sebbene sia inusuale che un robot outdoor come un veicolo autonomo venga trasportato da una zona ad un'altra come invece può accadere per un piccolo robot indoor (ad esempio una aspirapolvere robotizzata), questo problema rappresenta una sfida per la valutazione dell'efficienza dell'algoritmo: è il caso di recupero da situazioni critiche, come una sfortuita concatenazione di errori dovuta ad uno slittamento eccessivo delle ruote, che porterebbe ad una totale perdita della localizzazione.

2.2.1 Localizzazione markoviana

Gli algoritmi di localizzazione sono una variante dei filtri Bayesiani visti nel Paragrafo 2.1; la diretta applicazione del filtraggio ai problemi di localizzazione è chiamata *localizzazione markoviana*. La Tabella 2.4 descrive il generico algoritmo markoviano: l'algoritmo riceve in input lo stato credenza precedente (bel_{t-1}), l'azione di controllo (u_t), la misura sensoriale (z_t) e la mappa nella quale effettuare la localizzazione (m). La mappa ha un ruolo fondamentale nell'incorporazione della misura sensoriale (linea 4) mentre non è sempre utilizzata nella fase di predizione dello stato $\overline{bel}(x_t)$. Per descrivere il processo di localizzazione markoviana utilizziamo l'ambiente descritto nella Figura 2.3 e descriviamo i singoli passaggi che permettono ad un robot di localizzarsi in un ambiente monodimensionale. L'intero processo è descritto visivamente nella Figura 2.5.

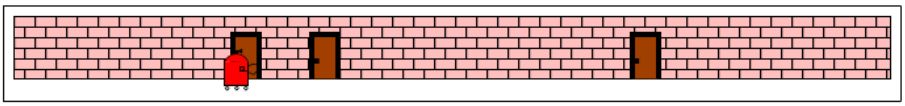


Figura 2.3: *Esempio di ambiente monodimensionale: un corridoio con tre porte indistinguibili. Inizialmente il robot non ha informazioni circa la sua attuale posizione. Si vuole individuare la corretta posizione del robot. Questa figura è stata presa da [5].*

1. Inizialmente non abbiamo informazioni circa la posizione attuale del robot; sappiamo che si trova all'interno di una mappa m rappresentata in Figura 2.3; introduciamo la funzione *bel* che rappresenta la distribuzione di probabilita circa la posizione del robot (stato credenza); tale funzione è inizialmente uniforme ed assegna un peso equiprobabile ad ogni posizione nella mappa (si noti che l'integrale di questa funzione rappresenta l'intera distribuzione di probabilità associata alle posizioni ammissibili e quindi somma ad 1); questo corrisponde allo stato di massima confusione; l'ambiente di esempio in cui vogliamo localizzarci ha le seguente caratteristiche: possiede tre punti di riferimento, identificati da tre porte indistinguibili tra di loro ed è facile riconoscere la porta dal muro del corridoio.

2. Ipotizziamo che il robot sia posizionato vicino ad una porta e che i sensori ci comunichino questa informazione; la funzione che identifica lo stato credenza viene quindi aggiornata incorporando l'informazione sensoriale e questo rappresenta il passo fondamentale per la localizzazione; l'uniformità della probabilità che avevamo nel passo precedente viene modificata: in corrispondenza delle tre porte avremo un aumento della probabilità, mentre in corrispondenza delle altre zone si avrà una diminuzione (si ricorda che l'integrale della funzione deve sommare sempre ad 1); il fallimento dell'individuazione della porta è rappresentato dall'attenuazione, ma non annullamento, della probabilità nelle zone in corrispondenza dei muri. La funzione che incorpora le informazioni sensoriali viene chiamata *distribuzione a posteriori*. A questo punto non abbiamo ancora una corretta posizione del robot, ma come anticipato nel Paragrafo 2.2 la localizzazione avviene mediante integrazione di misure successive.

3. Supponiamo a questo punto di effettuare un movimento verso destra. Mediante una convoluzione della funzione *bel* spostiamo i tre picchi di probabilità in base al movimento effettuato; introduciamo incertezza nella funzione *bel* facendo diminuire le intensità dei tre picchi dello stato precedente. La convoluzione riguarda la funzione *bel* allo stato precedente, nel rispetto dell'ipotesi markoviana.

4. Una volta effettuato il movimento il robot effettua una nuova misura e supponiamo che la sensoristica del robot ci dica che siamo nuovamente di fronte ad una porta; moltiplichiamo quindi la funzione *bel*, che rappresenta in questo momento la nostra *distribuzione a priori*, per la nuova misura sensoriale e notiamo che tale funzione concentra ora il suo maggior contributo in termini di peso in corrispondenza della seconda. Il risultato contiene una serie di picchi minori ma l'unico grande picco è posizionato in corrispondenza della seconda porta.

2.2.2 Bayes Rule

In questo paragrafo dimostreremo come l'applicazione dell'algoritmo di localizzazione corrisponde all'applicazione della Regola di Bayes (2.1), ovvero una delle regole fondamentali della statistica inferenziale. In questo esempio supporremo che la mappa sia discretizzata in celle di uguale dimensione come in Figura 2.4.

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{2.1}$$

Siano dati:

- X: la distribuzione di probabilità del posizionamento del robot sulla mappa.
- Z: la misura fatta dal sensore

Come abbiamo visto poc'anzi, il passo di localizzazione della posizione consiste nell'introduzione della misura Z sulla distribuzione a priori $P(X)$, aggiornando di fatto lo stato della distribuzione; tradotto in termini statistici ed applicando esattamente la regola di Bayes (2.1) avremo:
$P(X) \rightarrow$ Distribuzione a priori sull'intera mappa
$P(Z|X) \rightarrow$ Probabilità di aver effettuato una misurazione; ad esempio, nel caso del corridoio corrisponde alla probabilità di aver osservato una porta
$P(Z) \rightarrow$ Gioca il ruolo di normalizzatore

in termini di mappa discreta:

$P(X_i) \rightarrow$ Probabilità che il robot sia nella cella i-esima
$P(Z|X_i) \rightarrow$ Probabilità di aver rilevato una porta essendo nella i-esima cella
$\alpha = \sum_i P(Z|X_i) \cdot P(X_i)$ costante di normalizzazione
$\overline{p}(X_i|Z) = P(Z|X_i) \cdot P(X_i)$ Probabilità a posteriori non normalizzata, ovvero dopo l'introduzione della misura attraverso la moltiplicazione della probabilità che il robot sia nella i-esima cella per la probabilità di aver effettuato la misura nell'i-esima cella.
$P(X_i|Z) = \frac{1}{\alpha} \cdot \overline{p}(X_i|Z)$ Probabilità a posteriori, normalizzata.

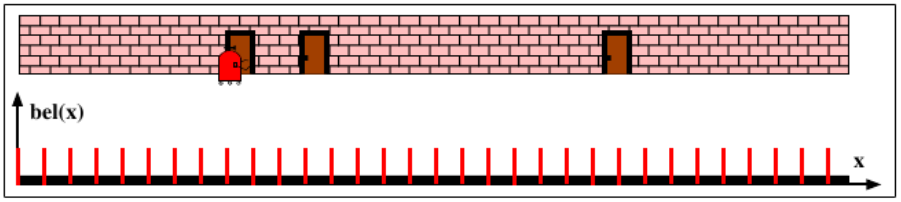


Figura 2.4: *Esempio di ambiente monodimensionale discretizzato. Questa figura è ispirata da [5].*

2.3 Robot Motion

In questa sezione verranno coperti gli aspetti probabilistici legati ai modelli di moto, responsabili della convoluzione sulla funzione di probabilità $p(x_t|u_t, x_{t-1})$ associata alla localizzazione. Questi procedimenti, che si diversificano a seconda della cinematica del robot utilizzato, assumono un ruolo cruciale in ogni algoritmo di localizzazione in quanto sono i principali responsabili dell'accrescimento lordo dell'incertezza sulle coordinate che rappresentano lo stato del robot. Il risultato di una azione di controllo verrà quindi descritto sottoforma di distribuzione di probabilità a posteriori. Questa sezione prende largamente spunto da [5].

2.3.1 Configurazione cinematica

Lo studio della cinematica rappresenta la descrizione delle possibilità di moto di un corpo rigido; la *configurazione* di un robot mobile è comunemente indicata da sei variabili, tre coordinate cartesiane in aggiunta a tre misure di angoli denominati angoli di *rollio, beccheggio* ed *imbardata* che rappresentano le rispettive rotazioni attorno ai tre assi del piano cartesiano. Le sei variabili esprimono una diversità di configurazione rispetto ad un sistema di coordinate esterno; tale configurazione rappresenta lo stato cinematico, viene indicata con la scrittura x_t e chiamata *pose 6DoF*. La pose può essere vista come una composizione di due differenti tipi di informazione: *Location* o *Position* per quanto concerne la parte cartesiana e *Bearing* o *Orientation* per la parte di rotazione. La pose associata ad un robot operante su un mondo bidimensionale, raffigurata in Figura 2.6, può essere descritta attraverso una sottoparte della pose 6DoF, eliminando le variabili non rappresentate in questo mondo

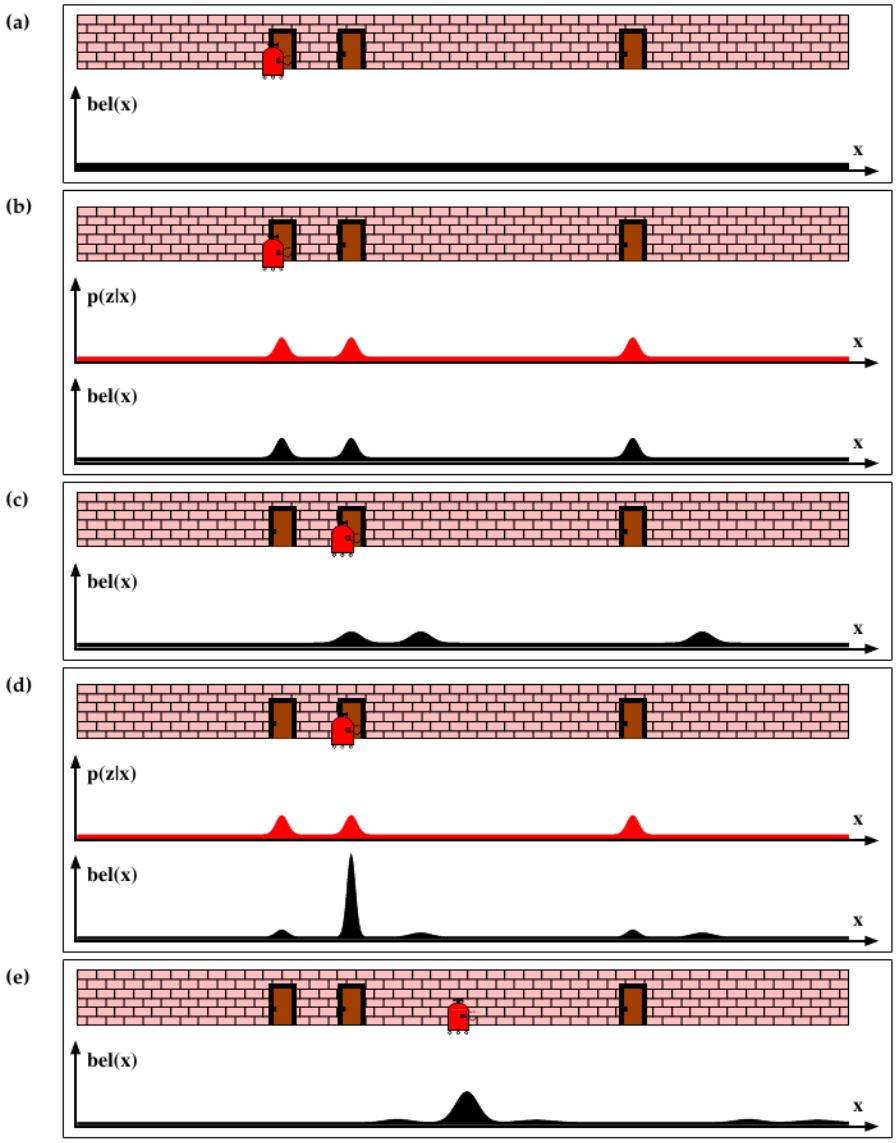


Figura 2.5: *Nella figura è descritto il funzionamento di un generico algoritmo di localizzazione markoviano in un ambiente monodimensionale. Ogni figura descrive la posizione del robot in un corridoio con il rispettivo stato credenze belx_t. In (a) il robot allo stato iniziale, si noti la distribuzione uniforme sullo stato credenza; in (b) viene recepita l'informazione sensoriale ed attraverso il modello sensoriale $p(z_t|x_t)$ viene descritta la probabilità di osservare una porta nelle diverse posizioni del corridoio, operazione che aggiorna lo stato credenza; in (c) ed (e) viene effettuato un movimento, si noti come la distribuzione viene spostata ma altresì sparsa. Questa figura è stata presa da [5].*

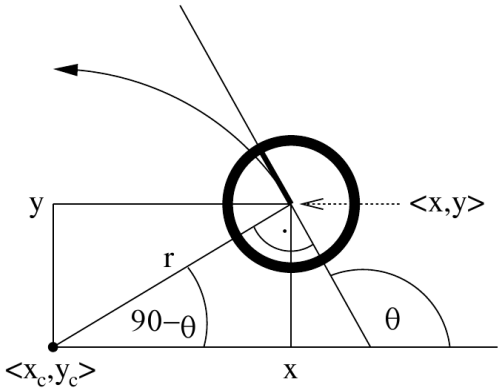


Figura 2.6: *La pose 3DoF di un robot in rispetto al sistema di riferimento globale:* (x, y, θ)*. Questa figura è presa da [5].*

2D.

I modelli di moto probabilistico agiscono sulle configurazioni cinematiche introducendo incertezza a fronte di una transizione di stato comandata da un generico controllo u_t, ed il modo in cui viene introdotta l'incertezza è descritto in Equazione 2.2.

$$p(x_t|u_t, x_{t-1}) \tag{2.2}$$

Il modello di moto descrive la distribuzione a posteriori attraverso il modello cinematico u_t applicato all'istante x_{t-1}. Il risultato dell'applicazione del modello consiste nella determinazione di un'area plausibile entro la quale il robot potrebbe trovarsi a causa di errori del movimento. La distribuzione $p(x_t|u_t, x_{t-1})$ è rappresentata dall'area ombreggiata in Figura 2.7: più scura è la zona ombreggiata, più è probabile la *pose* in quell'area. La definizione di questa area corrisponde alla definizione del modello di moto ed esistono numerosi approcci in letteratura [5]. La precisione con cui l'area viene definita gioca un ruolo determinante nella efficienza dell'algoritmo di localizzazione. Meno preciso sarà il modello, più alti dovranno essere i gradi di incertezza introdotti dal modello per sopperire ad una semplificazione eccessiva utilizzata in fase di modellazione.

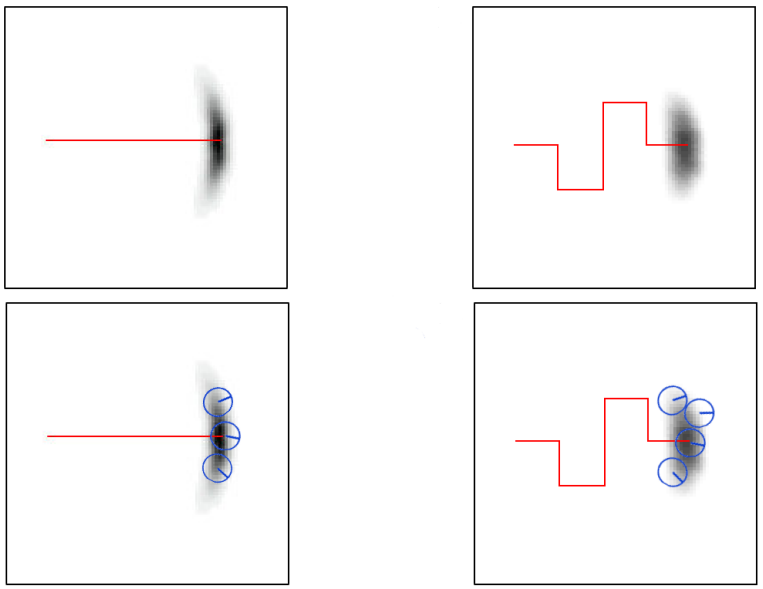


Figura 2.7: *Figure a sinistra: come esempio immaginiamo di avere un robot mobile che deve seguire una linea dritta. Per eseguire questo compito daremo comandi appropriati u_t affinché il robot vada ovviamente diritto. A causa di errori però il robot non segue una linea dritta; l'area di incertezza è rappresentata dall'area ombreggiata. Nella figura in basso a sinistra sono rappresentati in modo discreto ed in rosso i possibili orientamenti. Nelle figure a destra un secondo esempio con un movimento non rettilineo. Questa figura è ispirata da [5].*

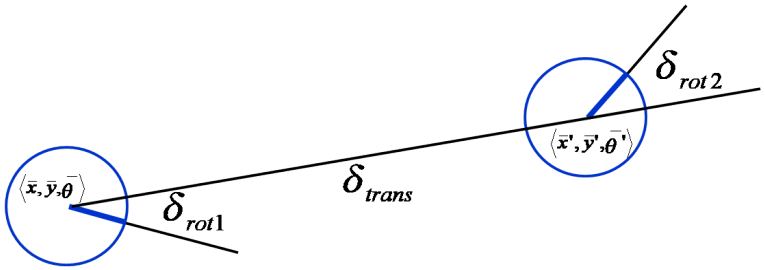


Figura 2.8: *Odometry Model. Questa figura è ispirata da [5].*

Descriveremo ora il modello di moto dal quale ci siamo ispirati per il lavoro di questa tesi. Il modello assume che il robot abbia accesso diretto ad opportuni sensori posizionati sulle ruote, chiamati d'ora in avanti sensori *odometrici*. Le letture odometriche verranno usate come input di controllo u_t, anche se tecnicamente non lo sono in quanto risultato di una azione di controllo. Per trattare un modello basato sui dati odometrici lo stato cinematico dovrebbe includere le informazioni di velocità del robot, ma nella pratica questo non viene effettuato a favore di uno spazio degli stati più piccolo ed un modello più semplice.

2.3.2 Odometry Model 3DoF

Il modello di moto introdotto in [5] prevede l'utilizzo delle misure odometriche, ovvero delle informazioni di moto relativo riferite ad un istante precedente. Nello specifico si intende che il robot, nell'intervallo di tempo $(t-1, t]$, effettua un avanzamento da una a pose x_{t-1} ad una pose x_t. L'odometria riporta l'avanzamento dalla pose $\overline{x}_{t-1} = (\overline{x}\ \overline{y}\ \overline{\theta})^T$ alla pose $\overline{x}_t = (\overline{x}'\ \overline{y}'\ \overline{\theta}')^T$. Le barre sopra le componenti dello stato indicano che esse sono relative a coordinate interne al robot e la trasformazione tra il robot ed il mondo non è conosciuta. L'idea chiave alla base di questo modello consiste nella stima del parametro di controllo u_t, ottenuta mediante una differenza tra il termine $\overline{x}_{t-1}$ ed il termine $\overline{x}_t$. Il controllo u_t viene scomposto in una sequenza di tre passi che possono essere osservati in Figura 2.8: una rotazione seguita da una traslazione ed infine una ultima rotazione. E' facilmente verificabile che per ogni coppia di pose $(\overline{s}, \overline{s}')$ esiste un unico vettore di parametri $(\delta_{rot1}, \delta_{trans}, \delta_{rot2})^T$ e che tale vettore è sufficiente per ricostruire il moto re-

1 **Input**: (x_t, u_t, x_{t-1})
2 δ_{rot1}=atan2($\overline{y}' - \overline{y}$,$\overline{x}' - \overline{x}$)-$\overline{\theta}$
3 δ_{trans}= $\sqrt{(\overline{x}' - \overline{x})^2 + (\overline{y}' - \overline{y})^2}$
4 δ_{rot2}=$\overline{\theta}' - \overline{\theta} - \delta_{rot1}$
5 $\hat{\delta}_{rot1}$=atan2($y' - y$,$x' - x$)-θ
6 $\hat{\delta}_{trans}$=$\sqrt{(x' - x)^2 + (y' - y)^2}$
7 $\hat{\delta}_{rot2}$=$\theta' - \theta - \delta_{rot1}$
8 p1=**prob** $(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1\hat{\delta}^2_{rot1} + \alpha_2\hat{\delta}^2_{trans})$
9 p2=**prob** $(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3\hat{\delta}^2_{trans}, +alpha_4\hat{\delta}^2_{rot1} + \alpha_4\hat{\delta}^2_{rot2}$
10 p3=**prob** $(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1\hat{\delta}^2_{rot2} + \alpha_2\hat{\delta}^2_{trans})$
11 **restituisci** $p1 \cdot p2 \cdot p3$

Tabella 2.5: *L'algoritmo alla base dell'Odometry Model per calcolare* $p(x_t|u_t, x_{t-1})$.

lativo tra le due pose $\overline{s}$ e $\overline{s}'$. Utilizzando questo modello si assume che i tre parametri siano sufficienti ad esprimere tutta l'incertezza introdotta dalla suddetta scomposizione. Possono quindi essere utilizzati per propagare l'incertezza dalla pose precedente, senza il rischio di sottostimarla.
L'algoritmo descritto in Tabella 2.5 implementa il calcolo della $p(x_t|u_t, x_{t-1})$. L'algoritmo prende in ingresso una pose iniziale x_{t-1}, una coppia di pose $u_t = (\overline{x}_{t-1}\ \overline{x}_t)^T$ ottenuta dall'odometria del robot ed ipotizza una pose finale x_t. Le righe da 2 a 4 di Tabella 2.5 calcolano gli spostamenti del robot letti mediante le misure odometriche. Le righe da 5 a 7 identificano il moto relativo tra le due posizioni x_{t-1} e x_t. La funzione **prob**(a, b^2) introduce un errore su a campionando da una distribuzione normale con con media nulla e varianza b^2. I valori da α_1 fino ad α_4 si presentano come parametri specifici per il robot e permettono di adattare il modello a diversi tipi di robot.

2.3.3 Motion Errors

Concludiamo la sezione riguardante i modelli di moto mostrando alcune delle numerose cause che possono portare ad una rilevazione errata del movimento da parte dei sensori, odometrici e non, presenti sul robot.

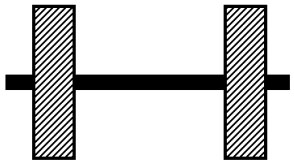


Figura 2.9: *Caso Ideale. Ruote perfettamente parallele e della stessa dimensione. Fondo stradale liscio*

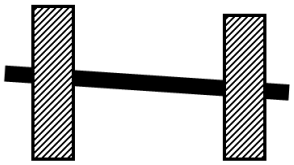


Figura 2.10: *La dimensione delle ruote può cambiare a causa di un consumo non uniforme delle ruote, per una differente pressione di gonfiaggio o per cause meteorologiche.*

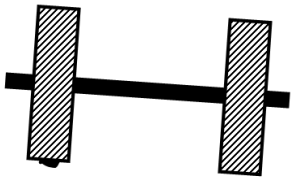


Figura 2.11: *Soprattutto in ambito outdoor è facile trovare dei dossi o delle sconnessioni stradali che coinvolgono solamente un sottoinsieme delle ruote del veicolo*

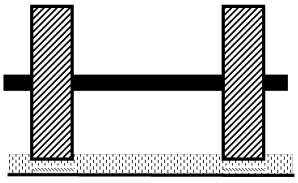


Figura 2.12: *Concludiamo la carrellata con i tipici casi di slittamento delle ruote; nei robot indoor un esempio può essere quello di un tappeto, mentre in ambiente outdoor la presenza di ghiaccio.*

2.4 Robot Perception

Nella seguente sezione verranno trattati gli aspetti probabilistici legati ai modelli sensoriali $p(z_t|x_t, m)$, dove x_t è la pose del robot, z_t è la rilevazione sensoriale al tempo t ed m è la mappa dell'ambiente. Questi modelli descrivono il processo di formazione con il quale le misure sensoriali vengono generate nel mondo fisico, anche se nella pratica risulta estremamente difficile modellare il sensore in modo accurato proprio a causa della complessità dei fenomeni fisici. Le specifiche dei modelli dipendono ovviamente dalla tipologia del sensore utilizzato. Esistono diverse categorie di sensori che vanno dai sensori di prossimità (sonar, radar, LIDAR, infrared) ai sensori fisici (accelerometri, giroscopi, magnetometri) o satellitari (GPS). Ogni modello viene quindi adattato per trattare le misure generate in modo probabilistico, introducendo i diversi tipi di errore che possono influenzare il sensore specifico e creando una distribuzione di probabilità condizionale $p(z_t|x_t)$ al posto di una funzione deterministica $z_t = f(x_t)$. Ci limiteremo alla descrizione dettagliata dei modelli di sensore di tipo *range finder* ed in particolare alla descrizione del modello per i sensori LIDAR descritto in [5].

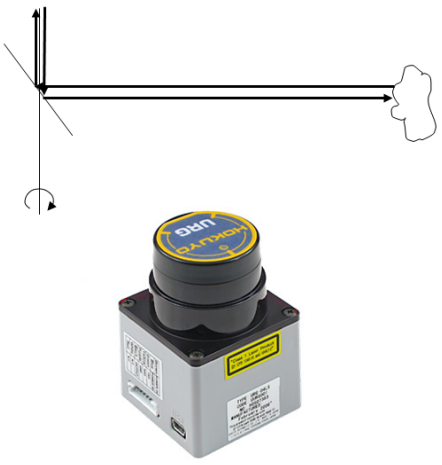


Figura 2.13: *Un tipico sensore LIDAR. Il sensore dispone di un emettitore laser e di uno specchio motorizzato che permette di orientare il fascio laser su un arco di circonferenza che può andare da pochi gradi fino all'intero angolo giro.*

La scansione di un LIDAR consiste in K misure $z = \{z_1, z_2, \cdots, z_k\}$ e la

probabilità totale al tempo t $p(z_t|x_t, m)$ è la distribuzione di probabilità totale ottenuta come prodotto delle verosimiglianze delle singole $z - esime$ misure.

$$p(z_t|x_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m) \tag{2.3}$$

Questo procedimento assume che ci sia indipendenza condizionale tra le scansioni successive, assunzione valida solo in un caso ideale; è comune e facilmente intuibile che ci sia un certo grado di dipendenza tra due misure successive. Il modello proposto in [5] non terrà conto di queste dipendenze.

2.4.1 Beam Model

Il beam model, che rientra nella categoria degli *enviroment measurement models*, ha lo scopo di modellare il rumore delle letture ottenute da laser range finder (o LIDAR) attraverso una distribuzione condizionale $p(z_t|x_t, m)$ dove z_t è il dato restituito dal sensore (nella totalità delle misure), x_t è la pose del robot ed m la mappa dell'ambiente. Per modellare il processo di generazione delle misure è necessario specificare l'ambiente in cui le misure vengono generate: a questo scopo viene utilizzata la mappa voxel creata seguendo le indicazioni del paragrafo 4.3. Il beam model introduce quattro tipologie di fonte di imprecisione che hanno l'origine nelle seguenti considerazioni: piccoli errori di misura, errori dovuti ad oggetti imprevisti nello spazio, errori dovuti al fallimento nell'individuazione di un oggetto ed infine un errore casuale dovuto a cause non precisate. Il modello $p(z_t|x_t, m)$ risultante si presenta come una mistura di quattro distribuzioni di probabilità diverse per ogni tipologia di errore.

In tutte le formule che verranno descritte, z_k rappresenta la distanza dell'ostacolo dalla posizione x_t, valutata dalla k-esima misura del laser. Tale misura viene determinata mediante un procedimento di raycasting sulla voxel map.

- Misura corretta con errore locale (si veda Figura 2.14 (a)): in questa categoria rientrano gli errori di lettura dovuti alla risoluzione dei laser range finder (ad esempio la risoluzione del laser LSM4001 è di 4cm) e

sono modellizzati da una gaussiana molto stretta centrata sulla k-esima misura del dato sensoriale. La varianza della gaussiana è il primo parametro del modello Beam Model e viene chiamata σ_{hit}. La probabilità, p_{hit}, di avere effettuato tale misura è data da:

$$p_{hit}(z_t^k|x_t, m) = \begin{cases} \eta\mathcal{N}(z_t^k - z_t^{k*}, \sigma_{hit}^2) & \text{se } 0 \leqslant z_t^k \leqslant z_{max} \\ 0 & \text{altrimenti} \end{cases} \tag{2.4}$$

- Errori dovuti ad oggetti imprevisti nello spazio (si veda Figura 2.14 (b)): gli errori di questa tipologia sono ad esempio oggetti in movimento, persone che camminano o oggetti spostati e non rappresentati correttamente nelle mappe. Le letture che rientrano in questa categoria di errori sono letture z_k^* sempre più corte di quelle reali (ovviamente non possono essere più lunghe in quanto si prevede un oggetto estraneo alla mappa). Il modo in cui questa situazione viene modellizzata coincide con una distribuzione esponenziale

$$p_{short}(z_t^k|x_t, m) = \begin{cases} \eta\lambda_{short}e^{-\lambda_{short}z_t^k} & \text{se } 0 \leqslant z_t^k \leqslant z_t^{k*} \\ 0 & \text{altrimenti} \end{cases} \tag{2.5}$$

- Fallimento della rilevazione (si veda Figura 2.14 (c)): in questo caso assumiamo che il laser riporti la misura massima (max range). Le cause che possono portare al fallimento di una rilevazione sono molteplici ed includono la presenza di materiali riflettenti (nel garage ad esempio sono presenti pannelli rosso/bianco catarifrangenti per l'indicazione delle colonne) oppure materiali che assorbono completamente il laser beam.

$$p_{max}(z_t^k|x_t, m) = I(z = z_{max}) = \begin{cases} 1 & \text{se } 1 = z_{max} \\ 0 & \text{altrimenti} \end{cases} \tag{2.6}$$

- Misure casuali (si veda Figura 2.14 (d)): l'ultima tipologia di errore consiste nelle misure casuali, non spiegabili in modo deterministico. La possibilità che ci sia una misura di questo tipo è uniformemente distribuita lungo l'intero range di funzionamento del sensore.

$$p_{rand}(z_t^k|x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{se } 0 \leqslant z_t^k \leqslant z_{max} \\ 0 & \text{altrimenti} \end{cases} \tag{2.7}$$

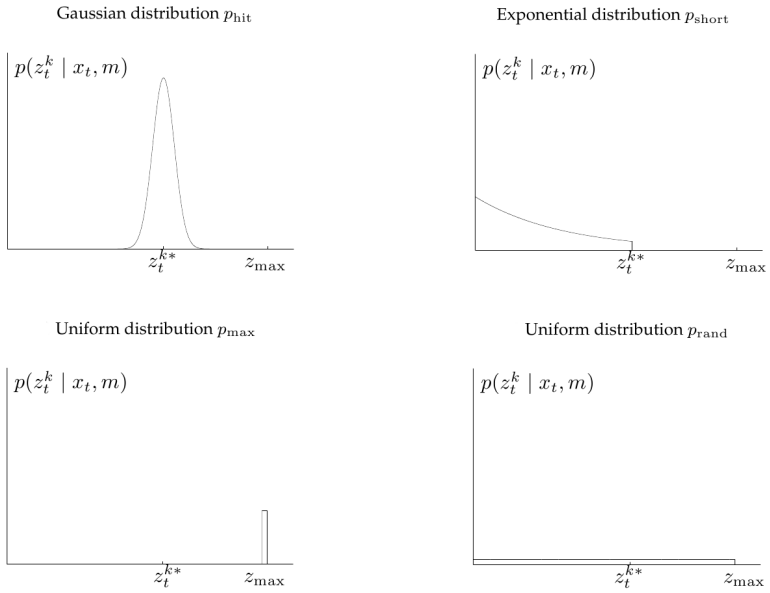


Figura 2.14: *Componenti del modello sensoriale Beam Model. In tutti i grafici l'ascissa corrisponde alle misure z_t^k, l'ordinata alla verosimiglianza. Questa figura è stata presa da [5].*

Le quattro distribuzioni sopra elencate vengono composte utilizzando dei pesi associati a ciascuna tipologia di incertezza e la complessiva probabilità è data da

$$p(z_t^k|x_t, m) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit}(z_t^k|x_t, m) \\ p_{short}(z_t^k|x_t, m) \\ p_{max}(z_t^k|x_t, m) \\ p_{rand}(z_t^k|x_t, m) \end{pmatrix} \tag{2.8}$$

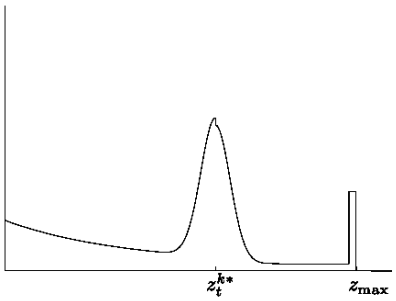


Figura 2.15: *La $p(z_t^k|x_t, m)$ ottenuta dalla composizione delle quattro distribuzioni. Questa figura è stata presa da [5].*

Capitolo 3

Estensione del modello di moto a 6DoF

In questo capitolo presenteremo il problema della localizzazione in ambiente 3D con un modello di moto probabilistico a 6DoF. Verrà inizialmente analizzato in dettaglio il modello proposto da Thrun *et al.* [5] dal quale il lavoro di questa tesi prende spunto, evidenziando le inadeguatezze della semplificazione 3DoF in ambito urban outdoor e verrà quindi proposto un nuovo approccio. Il modello che proponiamo permette di rappresentare in termini probabilistici tutte e sei le componenti del vettore di stato 6DoF, con innovative proposte per quanto concerne il modo in cui l'incertezza viene introdotta. Il sistema che ne deriva, completamente adattabile alle diverse tipologie di robot mediante opportuni parametri, è stato progettato in modo da permettere la modellazione di movimenti 6DoF anche in assenza di un vettore di transizione completo nelle sue sei componenti (ovvero in mancanza di sensori aggiuntivi a quelli visti fino ad ora). A tal fine sono stati introdotti valori di soglia, anche essi parametrizzabili, atti a rappresentare gli errori minimi e massimi tollerati dal modello in presenza/assenza di ulteriori sensori inerziali.
L'interpretazione completa del sistema 3DoF ha richiesto una lunga analisi al fine di coglierne gli aspetti chiave. Nonostante gli sforzi compiuti la comprensione è risultata tutt'altro che semplice ed ha portato più volte alla

scelta ed allo sviluppo di strade sbagliate.

3.1 Limitazioni modello 3DoF

Il modello proposto nel lavoro Thrun *et al.* [5] si basa sul presupposto che l'ambiente nel quale effettuare la localizzazione sia un piano sul quale un robot mobile ha uno stato cinematico (*pose*) descritto da una tripla di variabili $x_t = (x, y, \theta)$; se questo presupposto può essere accettato in un tipico ambiente indoor, tale ipotesi viene meno in un ambiente outdoor ed in particolare con un veicolo adibito al trasporto di persone. Le diverse configurazioni di carico del veicolo e la diversa pressione delle ruote, così come la presenza di avvallamenti, dossi e sconnessioni delle strade, portano ad una situazione in cui la trasformazione tra il sistema di riferimento del veicolo ed il sistema di riferimento dei sensori è incognita (si vedano le figure da 2.9 a 2.12 del Capitolo 2). Una ulteriore difficoltà è data dalla presenza di un sistema di sospensioni. Tale sistema, atto a favorire stabilità e comfort per gli occupanti del veicolo, pone nella condizione di avere una ulteriore trasformazione tra il sistema di riferimento delle ruote (masse non sospese) ed il sistema di riferimento della carrozzeria (masse sospese). I modelli di moto utilizzati nella localizzazione bidimensionale a 3DoF basano il calcolo della posizione sulla presenza di parti statiche all'interno di una mappa nota. Nell'approccio di Thrun *et al.* [5] la tecnica adottata per percepire l'ambiente circostante è quella di dotare il robot mobile di un LIDAR con asse di scansione perpendicolare al piano di navigazione. Tale accorgimento è volto a massimizzare la portata utile dello strumento e quindi a massimizzare la quantità di informazione raccolta dal sensore. Un LIDAR con asse di scansione non perpendicolare al terreno fornirebbe la misura della distanza tra il sensore stesso ed il piano di navigazione del robot, rendendo questa misura costante e perciò priva di informazione utile alla localizzazione sul piano bidimensionale. Inoltre, in assenza di un algoritmo di pre-elaborazione orientato a catalogare o segmentare i dati in input non è possibile classificare le misure rilevate che verranno considerate indistintamente come ostacoli in fase di navigazione. In un ambito tridimensionale eventuali *significative* variazioni nei valori di rollio o beccheggio (si veda Figura 3.1 e Figura 3.2)

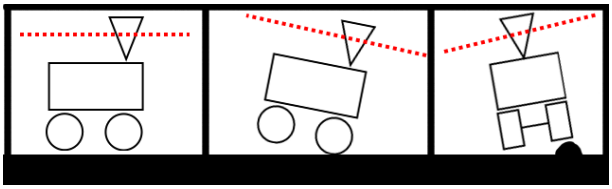


Figura 3.1: *Particolari configurazioni del robot che possono portare ad una scansione dei LIDAR su un piano non parallelo a quello stradale*

dovute a sconnessioni stradali ed alla presenza di ammortizzatori, faranno sì che i LIDAR effettuino le misure su un piano non perpendicolare al piano di navigazione. Tale situazione aumenta la possibilità di ottenere letture indesiderate nell'ambito della localizzazione 2D come ad esempio letture del piano di navigazione. Considerando che la portata dei sensori LIDAR utilizzati in ambiente outdoor variano dalle decine fino al centinaio di metri, anche una inclinazione di pochi gradi può portare ad individuare come ostacolo lo stesso piano stradale. Un esempio di questo problema è stato riscontrato nei test di localizzazione effettuati in Piazza della Scienza, tra gli edifici U1 ed U2, dove il piastrellato è molto deformato. La Figura 3.3a e la Figura 3.3b descrivono questa situazione.

La localizzazione nell'ambiente della piazza si è rilevata molto difficoltosa proprio a causa degli avvallamenti presenti. Nell'ambito di una demo svolta in occasione del programma televisivo Buongiorno Mattina (Rai3) la mancanza di un localizzatore 6DoF, unita alla necessaria rappresentazione bidimensionale dello spazio, ci ha costretti alla scelta di un percorso che non passasse per zone avvallate. In tali zone l'inclinazione del veicolo dovuta sia ai passeggeri sia alle sconnessioni stradali rendeva nella pratica inutilizzabili le misure rilevate dai LIDAR. La ricerca di una configurazione dei parametri del modello in grado di gestire correttamente l'alto livello di incertezza non ha portato risultati soddisfacenti.

Un altro esempio di situazione nella quale un algoritmo di localizzazione basato su un modello a due sole dimensioni fallirebbe è il seguente: immaginiamo di trovarci in un garage multipiano dove ogni piano è collegato da una rampa come quelle in Figura 3.4 e di essere posizionati di fronte ad una di esse. Anche in questo caso ci troveremmo ad avere misure che non

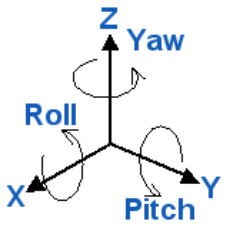


Figura 3.2: *Convenzioni utilizzate per le rotazioni semplici sui tre assi cartesiani. La rotazione attorno all'asse X è chiamata rollio (o Roll), la rotazione attorno all'asse Y è chiamata beccheggio (o Pitch), la rotazione attorno all'asse Z è chiamata imbardata (o Yaw).*

rappresentano ostacoli, ma che fanno riferimento al fondo stradale inclinato (la rampa) senza per di più riuscire a discriminare queste misure da quelle dei muri che delimitano la rampa, ovvero gli elementi della mappa che permettono la localizzazione. Appare quindi evidente come l'approccio basato su mappe bidimensionali e con una cinematica limitata alle tre componenti (x, y, θ) risulti insufficiente per effettuare una localizzazione in un mondo 3D. La crudeltà del modello, unita all'eccessiva semplificazione fisica, pone seri interrogativi sulla affidabilità della procedura di localizzazione e la successiva navigazione in un ambiente urban outdoor.

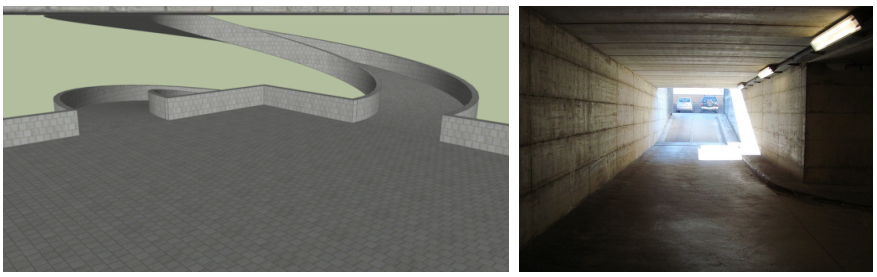
3.2 Analisi del modello di moto a 3DoF

Prima di introdurre la soluzione proposta al problema tridimensionale ci concentreremo sull'analisi dell'algoritmo 3DoF, allo scopo di scomporre il modello nelle sue componenti base e rendere più intuitive le modifiche presentate. Come descritto nel Paragrafo 2.3.2 l'*Odometry Model* è un particolare modello di moto che utilizza le informazioni odometriche provenienti da sensori collocati sulle ruote del robot al fine di descrivere le possibili posizioni assunte dopo un atto di moto (si rimanda al Capitolo 2 per una spiegazione approfondita sul perché venga introdotta una incertezza).

(a)

(b)

Figura 3.3: *In rosso nella figura (a) possiamo osservare le rilevazioni del LIDAR montato sul nostro veicolo e riportate sulla mappa 2D di Piazza della Scienza; i punti non in corrispondenza degli ostacoli rappresentano il fondo piastrellato del piazzale. Come si può vedere in figura (b) il piazzale presenta un avvallamento verso i lucernari e questo determina una inclinazione del veicolo sufficiente a far puntare i LIDAR verso il fondo stradale.*



(a) **(b)**

Figura 3.4: *in Figura (a) un esempio CAD di rampa che collega più piani di un garage, mentre in Figura (b) l'uscita del garage U5, luogo nel quale si sono effettuati i test.*

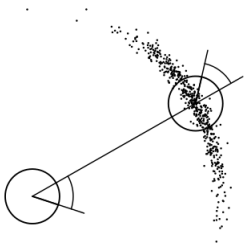


Figura 3.5: *Applicazione dell'odometry model ad una posizione generica del robot: i puntini in nero rappresentano le varie ipotesi di posizione dopo un atto di moto registrato dagli encoder*

3.2.1 Considerazioni alla base delle componenti del modello di moto 3DoF

Ricordando quando descritto nel Capitolo 2 e che riproponiamo in Figura 3.6 l'Odometry Model scompone le informazioni odometriche in una sequenza di tre atti di moto elementari che, composti, rappresentano lo stesso movimento misurato dagli encoder; le formule da (3.1) a (3.3) individuano numericamente i tre parametri di questo modello.

Considerazioni Base

E' interessante notare le seguenti **considerazioni base** che saranno il cardine dei ragionamenti nei paragrafi successivi. Nell'algoritmo 3DoF il calcolo della pose $\overline{x_t}$ [1] a partire dalla pose x_{t-1} è suddivisa in due parti concettuali. Le formule (3.1) e (3.2) permettono di individuare la nuova *position* origine del sistema robot attraverso le formule (3.4) e (3.5), mentre la formula (3.3) rappresenta il secondo componente della scomposizione della *rotation* (3.6); tale somma corrisponde al valore Δ_{rot} misurato dal sensore odometrico (3.7) e completa la definizione della pose x_t.

$$\delta_{rot1} = atan2(\overline{y}' - \overline{y}, \overline{x}' - \overline{x}) - \overline{\theta} \tag{3.1}$$

$$\delta_{trans} = \sqrt{(\overline{x} - \overline{x}')^2 + (\overline{y} - \overline{y}')^2} \tag{3.2}$$

$$\delta_{rot2} = \overline{\theta}' - \overline{\theta} - \delta_{rot1} \tag{3.3}$$

$$x' = \delta_{trans} * cos(\delta_{rot1}) \tag{3.4}$$

$$y' = \delta_{trans} * sin(\delta_{rot1}) \tag{3.5}$$

[1] Nel capitolo useremo $\overline{x_t}$ per indicare una singola predizione del modello e x_t per indicare la pose risultato dell'azione di filtraggio complessiva (vedi Capitolo 4).

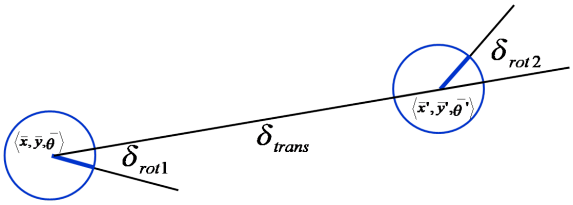


Figura 3.6: *Odometry Model: il movimento del robot nell'intervallo di tempo* $(t-1, t]$ *è approssimato da una rotazione* δ_{rot1} *seguito da una traslazione* δ_{trans} *e una seconda rotazione* δ_{rot2}

$$\theta' = \delta_{rot1} + \delta_{rot2} \tag{3.6}$$

$$\Delta_{rot} = \theta' \tag{3.7}$$

- 1^a categoria. L'errore introdotto sulla parte $position_t$. Viene rappresentato dalle componenti δ_{rot1} e δ_{trans} che permettono di definire una distribuzione di probabilità bivariata ad-hoc, con varianza proporzionale ai parametri ε_{rot1} e ε_{trans}. Una distribuzione normale bivariata non permette la creazione dell'effetto *"lunetta"* in Figura 3.7 che è rappresentativa degli errori reali. La definizione dell'area *"lunetta"* è alla base del modello e rappresenta la distribuzione di probabilità degli stati cinematici entro i quali si assume che il robot si possa essere posizionato, dopo aver effettuato una azione di controllo.

- 2^a categoria: l'errore introdotto sulla parte $orientation_t$. E' composto dalla somma dei due δ_{rot1} e δ_{rot2} con i rispettivi errori ε_{rot1} e ε_{rot2}. Sebbene in questo modello la rotazione finale venga espressa sottoforma di una somma di due rotazioni, ciascuna affetta da un suo errore, è possibile ottenere lo stesso risultato aggiungendo errore direttamente alla rotazione indicata dagli encoder (3.8). La seconda categoria completa la definizione della distribuzione di probabilità sullo stato $\overline{x_t}$ del robot dopo l'azione di controllo u_t. Il modello assume come condizionalmente indipendenti gli errori introdotti sulle parti position e orientation.

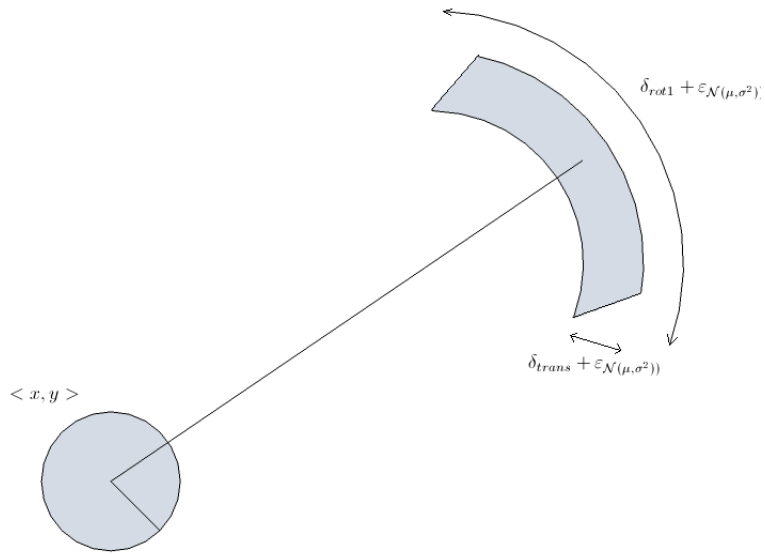


Figura 3.7: *In questa immagine è possibile osservare la zona di incertezza che le componenti δ_{rot1} e δ_{trans} combinate alle loro rispettive incertezze permettono di creare. Si noti come i punti in Figura 3.5 cadano all'interno di una zona di incertezza come quella rappresentata in questa figura.*

$$\sum_i \mathcal{N}(\mu_i, \sigma_i^2) = \mathcal{N}(\sum_i \mu_i, \sum_i \sigma_i^2) \tag{3.8}$$

3.3 Proposta del modello di moto a 6DoF

La soluzione del problema tridimensionale ha richiesto buona parte dello sviluppo del lavoro di tesi. Il modello di moto esteso a 6DoF deve rappresentare in modo corretto i possibili movimenti effettuati dal veicolo ed al tempo stesso deve decomporre il moto generico in una serie di moti elementari che permettano di introdurre un errore parametrizzato su tutte e sei le componenti della pose, ovvero le tre coordinate spaziali (x, y, z), ma anche le tre componenti di rotazione che individuiamo con (ϕ, ψ, θ), chiamati rispettivamente Roll, Pitch e Yaw (si veda Figura 3.2). La nostra proposta include una serie di parametri aggiuntivi che verranno utilizzati nel caso in cui i nuovi parametri di moto (ϕ, ψ e z) non siano disponibili nel robot in uso. Questa peculiarità

rende adattabile il nostro modello a robot non provvisti di sensori specifici per la navigazione 3D.
Nel Paragrafo 3.3.3 descriveremo un modello di moto errato, frutto dei primi tentativi di soluzione del problema di localizzazione, evidenziando l'importanza di una corretta modellizzazione. Un modello di moto errato (come quello presentato nel paragrafo 3.3.3) produrrà ipotesi di moto non veritiere, impedendo la corretta localizzazione del veicolo. Per la corretta creazione del modello di moto si rivelano indispensabili le *considerazioni base* del Paragrafo 3.2.1, che permettono di trattare il mondo a sei dimensioni in modo relativamente semplice.

3.3.1 Modifiche al sistema odometrico

Il sensore odometrico classico posizionato sulle ruote non è in grado di fornire i cambiamenti su tutte e sei le componenti del vettore di stato 6DoF. Per poter misurare gli spostamenti e le rotazioni nello spazio si è reso necessario l'utilizzo di un sensore inerziale (Inertial Measurement Unit, IMU) predisposto ad individuare le tre componenti che la pura odometria proveniente dalle ruote non è in grado di fornire, ovvero i dati di Z, *Roll* e $Pitch$. Abbiamo fatto affidamento ad un sensore inerziale X-Sens-MTi per questi tre nuovi parametri, mantenendo il valore di Yaw come dato letto dalle ruote. Il nuovo sistema inerziale+odometrico è stato definito *"odometro esteso"*.

3.3.2 Il modello a 6DoF

La prima operazione da considerare è l'espansione del vettore di stato: dalle tre componenti (x, y, θ) del modello bidimensionale alle sei componenti che permettono di individuare un corpo rigido in un mondo a tre dimensioni

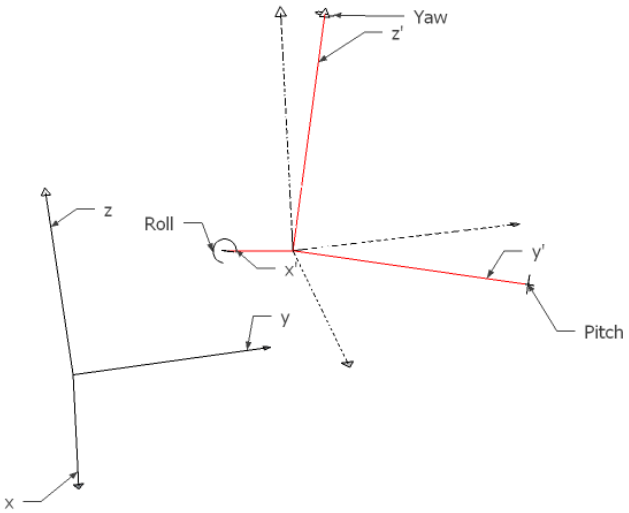


Figura 3.8: *Le sei componenti che identificano una trasformazione tra due sistemi di riferimento* $(\Delta_x, \Delta_y, \Delta_z, \Delta_{Roll}, \Delta_{Pitch}, \Delta_{Yaw},)$

(3.9).

$$x_t = \begin{pmatrix} \left.\begin{matrix} x \\ y \end{matrix}\right\} position \\ \\ \left.\begin{matrix} \theta \end{matrix}\right\} orientation \end{pmatrix} \Longrightarrow x_t = \begin{pmatrix} \left.\begin{matrix} x \\ y \\ z \end{matrix}\right\} position \\ \\ \left.\begin{matrix} \phi \\ \psi \\ \theta \end{matrix}\right\} orientation \end{pmatrix} \tag{3.9}$$

Il nostro sistema propone, a partire da una pose

$$x_{t-1} = (position_{t-1}, orientation_{t-1})$$

una decomposizione del generico atto di moto in sei atti di moto elementari divisi in due categorie: tre atti di moto per definire la nuova $\overline{position_t}$ e tre atti di moto per definire la nuova $\overline{orientation_t}$. A ciascuno dei sei atti di moto verrà inoltre aggiunta una incertezza in modo tale da ricreare le stesse condizioni del modello bidimensionale. Vediamo ora in dettaglio i

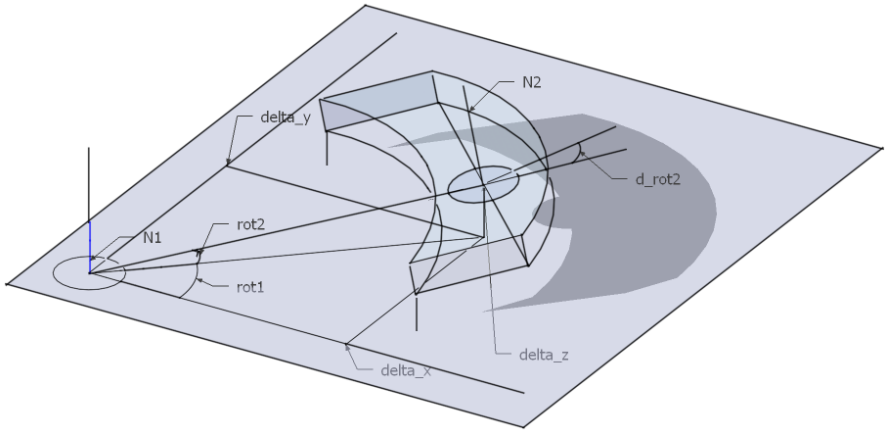


Figura 3.9: *In questa figura si possono osservare le tre componenti del modello utilizzate per il calcolo della position - ($\delta_{rot1}, \delta_{rot2}, \delta_{trans}$). L'errore aggiunto su queste tre componenti permette di individuare la "lunetta" tridimensionale nella quale verrà individuata la nuova position*

sei movimenti elementari che ci permettono di trasformare una $pose_{t-1}$ in una nuova $\overline{pose_t}$, iniziando dalla parte $position_t$ (si veda Figura 3.9 per il confronto con il modello tridimensionale). Tutte le rotazioni e traslazioni sono da intendersi intrinseche.

1. Rotazione δ_{rot1}, di tipo Yaw (attorno all'asse Z), necessaria per allineare l'asse X_{t-1} nella direzione della $position_t$; corrisponde alla rotazione δ_{rot1} del modello bidimensionale; identifichiamo con X'_{t-1} l'orientamento dell'asse ruotato;
   $X'_{t-1} = X_{t-1} \cdot Yaw$.

2. Rotazione δ_{rot2}, di tipo Pitch (attorno all'asse Y), necessaria per allineare l'asse X'_{t-1} nella direzione della nuova $position_t$; questo nuovo movimento introduce la possibilità di cambiamento di quota;
   $X''_{t-1} = X'_{t-1} \cdot Pitch$.

3. Traslazione δ_{trans}, lungo l'asse X''_{t-1}; questa traslazione porterà il sistema di riferimento ruotato nella $position_t$

I tre parametri δ_{rot1}, δ_{rot2} e δ_{trans} identificano le coordinate del sistema sferico mostrato in Figura 3.10. Per per passare dal sistema di riferimento cartesiano

al sistema di sferico vengono usate le equazioni da 3.10 a 3.12 mentre per il passaggio inverso le equazioni da 3.13 a 3.15:

$$\rho = \sqrt{x^2 + y^2 + z^2} \tag{3.10}$$

$$\phi = arctan(\frac{y}{x}) = arcsin(\frac{y}{\sqrt{x^2 + y^2}}) \tag{3.11}$$

$$\theta = arccos(\frac{z}{\sqrt{x^2 + y^2 + z^2}}) = arccot(\frac{z}{\sqrt{x^2 + y^2}}) \tag{3.12}$$

$$x = \rho\ sin\theta\ cos\ \phi \tag{3.13}$$

$$y = \rho\ sin\theta\ sin\ \phi \tag{3.14}$$

$$z = \rho\ cos\theta \tag{3.15}$$

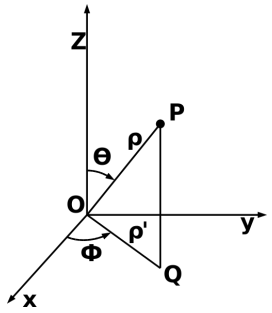


Figura 3.10: *Il sistema di riferimento sferico*

Una volta calcolata la $\overline{position_t}$ per completare la definizione della $\overline{pose_t}$ si passa al calcolo della $\overline{orientation_t}$; la nostra soluzione propone di effettuare un passo indietro per recuperare le informazioni di orientamento di $pose_{t-1}$,

dalla quale estraiamo la parte $orientation_{t-1}$. A tale orientamento applicheremo una rotazione generica composta dalle tre componenti di rotazione $\Delta_{rot} = (\Delta_{Roll}, \Delta_{Pitch}, \Delta_{Yaw})$ individuate dall'odometro esteso.
La rotazione generica da applicare a $pose_{t-1}$ richiede particolare attenzione poiché essa va espressa come composizione di rotazioni semplici. Le rotazioni nello spazio tridimensionale differiscono da quelle nelle due dimensioni per diversi motivi ed esistono molteplici strumenti matematici che permettono di effettuare questo tipo di operazioni (matrici di rotazione, metodo delle rotazioni di Eulero, quaternioni) ciascuno con propri pro e contro. Il metodo più efficace dal punto di vista operativo è rappresentato dalle rotazioni espresse sotto forma di quaternioni. Si procede quindi nell'inserimento delle tre componenti di rotazione semplice all'interno di una struttura *Quaternion*.

1. $Quaternion\ Q_{t-1} \Leftarrow Roll_{t-1}$
2. $Quaternion\ Q_{t-1} \Leftarrow Pitch_{t-1}$
3. $Quaternion\ Q_{t-1} \Leftarrow Yaw_{t-1}$
1. $Quaternion\ Q_{\Delta} \Leftarrow \Delta Roll$
2. $Quaternion\ Q_{\Delta} \Leftarrow \Delta Pitch$
3. $Quaternion\ Q_{\Delta} \Leftarrow \Delta Yaw$

La composizione del quaternione Q_{Δ} con quaternione Q_{t-1} permette di ottenere il nuovo orientamento $\overline{Q_t}$ dal quale possiamo ricavare i nuovi valori di $\overline{Roll_t}$, $\overline{Pitch_t}$ e $\overline{Yaw_t}$. A questo punto il calcolo delle componenti della $\overline{pose_t}$ è concluso e corrisponderà alla composizione della parte *position* e della parte *rotation* separatamente calcolate. Si noti che questo procedimento ricalca la stessa procedura utilizzata nel modello bidimensionale.

3.3.3 Esempio di modello errato

In questa sezione si vuole presentare il primo modello di moto utilizzato nel lavoro di tesi. Questo modello, che contiene un errore concettuale che ha reso impossibile la corretta localizzazione del robot, viene presentato per mostrare quanto importanti siano le considerazioni base viste nel Paragrafo 3.2.1 (che

furono utilizzate nel modello errato) e quanto la definizione di un corretto modello di moto influenzi la localizzazione.
Il modello prevede la scomposizione del generico atto di moto nei seguenti atti di moto elementari (si veda Figura 3.11):

1. rotazione δ_{rot1} attorno ad un asse arbitrario; per individuare l'asse di rotazione si richiede il calcolo della normale al piano individuato dal versore dell'asse X_{t-1} e dal versore orientato verso l'origine del sistema di riferimento finale.

$$\begin{aligned} N_1 &= \overrightarrow{X_{t-1}} \times \overrightarrow{AB} \\ \delta_{rot1} &= acos(|N1| \cdot |X_{t-1}|) \end{aligned} \tag{3.16}$$

2. traslazione di δ_{trans} lungo l'asse $X_{ruotato}$; questa traslazione porterà il sistema di riferimento ruotato nella $position_t$.

$$\delta_{trans} = \sqrt{(\overline{x} - \overline{x}')^2 + (\overline{y} - \overline{y}')^2 + (\overline{z} - \overline{z}')^2} \tag{3.17}$$

3. rotazione δ_{rot2} attorno ad un asse arbitrario: questa operazione richiede nuovamente il calcolo della normale al piano individuato dal versore dell'asse $X_{ruotato}$ e dal versore dell'asse X_t del sistema di riferimento finale.

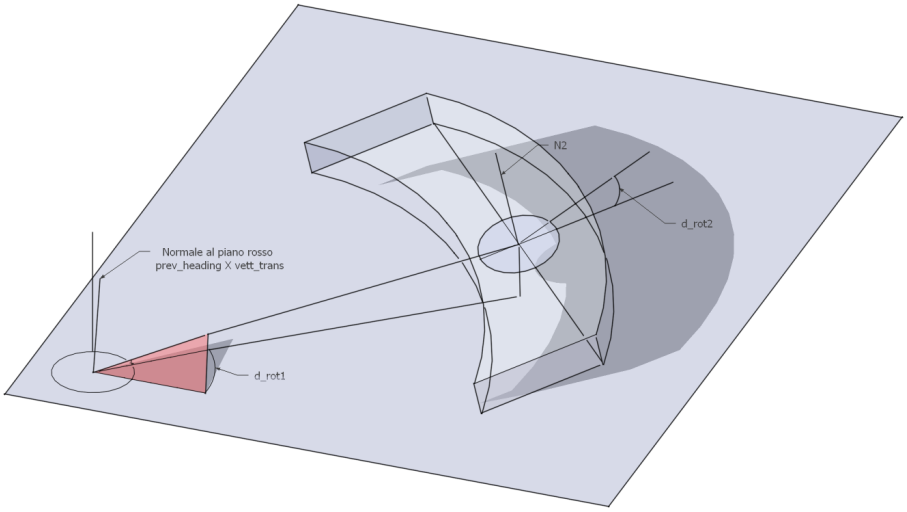
Descriveremo ora nel dettaglio perché questo modello porta con sé effetti indesiderati. Come nel caso di 3.3.2 il modello appena descritto cerca di ottenere una scomposizione del generico atto di moto nelle parti *position* e *rotation*. Il primo ed il secondo atto di moto, responsabili della parte *position*, hanno l'intenzione di traslare l'origine del sistema di riferimento al tempo $t-1$ nell'origine del sistema di riferimento al tempo t. Questa coppia di movimenti nasconde un grave errore concettuale: la creazione di ipotesi sulla nuova *position* pone le basi per l'introduzione di incertezza nei parametri che modellizzano il generico atto di moto, in questo caso su δ_{rot1} e δ_{trans}. Introducendo errore su questi due parametri si ottiene però un'area tridimensionale posizionata non correttamente nello spazio, come è possibile osservare in Figura 3.11a ed in Figura 3.11b. Questo errore viene maggiormente percepito durante le rotazioni del robot: più grande sarà la rotazione

più la "lunetta" verrà inclinata, portando l'intero sistema in uno stato di incoerenza in poche iterazioni. I test effettuati nel garage dell'edificio U5 hanno riscontrato la totale inadeguatezza della proposta vista in questo paragrafo. L'introduzione di una configurazione di parametri atti a tollerare grandi valori di incertezza ha avuto l'effetto di amplificare gli errori introdotti da questo modello.

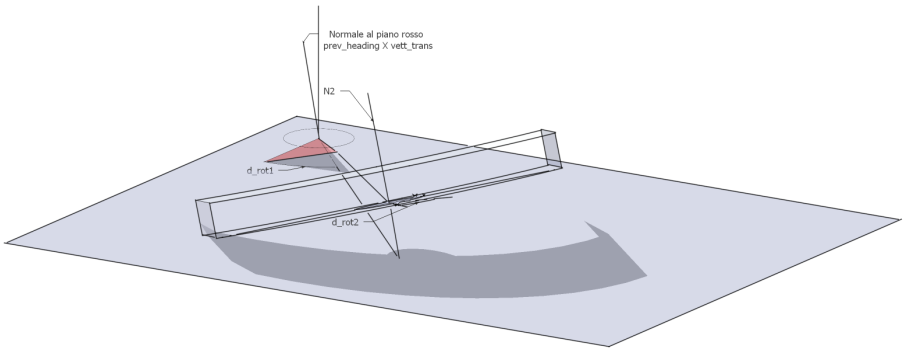
3.3.4 Introduzione degli errori nel modello 6DoF

Ritorniamo ora al modello corretto di odometro esteso. Affinché il modello di moto possa generare ipotesi verosimili è necessario introdurre degli errori sulle componenti vettore di stato x_t; gli errori, applicati sulle singole componenti, verranno generati procedendo ad una estrazione da una distribuzione normale a media nulla e deviazione standard calcolata secondo le seguenti considerazioni specifiche per ogni singolo atto di moto. Iniziamo descrivendo le deviazioni standard sugli errori degli atti di moto che identificano la *position*.

1. il primo atto di moto, ovvero la rotazione sull'asse Z δ_{yaw1} è influenzata:
   - da quanto il veicolo ha ruotato.
   - da quanto spazio è stato percorso dal veicolo: a movimenti più estesi corrisponde una maggiore probabilità di errore cumulato sulle misure odometriche
2. il secondo atto di moto, ovvero la rotazione sull'asse Y δ_{pitch1} è influenzata:
   - da quanto è variata la misura Δ_z misurata dall'odometro esteso
3. il terzo atto di moto, ovvero la traslazione lungo l'asse X δ_{trans} è influenzata:
   - da quanto spazio è stato percorso dal veicolo: a movimenti più estesi corrisponde una maggiore probabilità di errore cumulato sulle misure odometriche



(a)



(b)

Figura 3.11: *Nelle figure sopra si possono osservare gli assi attorno ai quali vengono effettuati i primi due movimenti del modello descritto in 3.3.3; il piano rosa identifica il piano individuato dai due versori X (versore dell'asse x) ed AB (versore sulla retta che collega le due origini dei sistemi di riferimento); in (b) è visibile un esempio di "lunetta" di incertezza generata da un errore introdotto sulle componenti* δ_{rot1} *e* δ_{trans}*.*

- da quanto il veicolo ha ruotato sull'asse Y ovvero dalla variazione Δ_{Pitch} misurata dall'odometro esteso

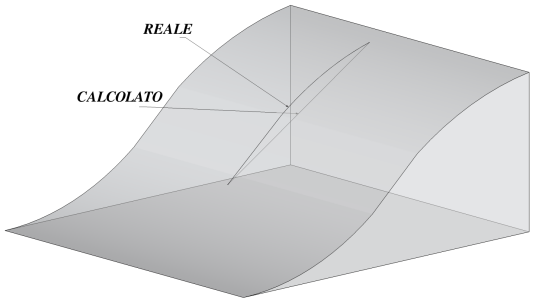


Figura 3.12: *Incertezza dalla componente Pitch.*

Poiché la presenza di cambio di *Pitch* in una traslazione identifica un movimento su una superficie non piana, aggiungiamo una quantità dipendente dal *Pitch* nel tentativo di spiegare l'errore. Nella Figura 3.12 si possono notare la traslazione calcolata e la traslazione reale. Una ulteriore spiegazione del perché viene introdotto un errore in base a variazioni su *Pitch* è possibile osservarlo in Figura 3.13.

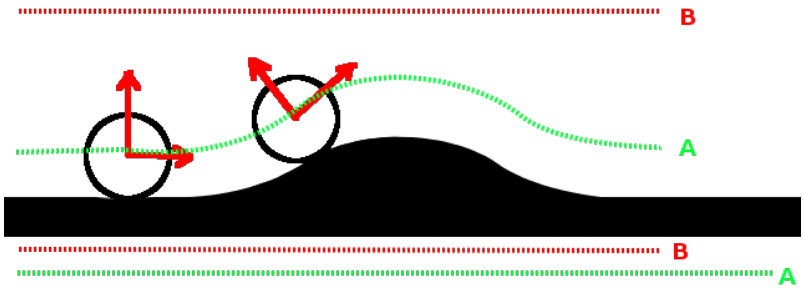


Figura 3.13: *Incertezza dalla componente Pitch.*

- da quanto il veicolo ha ruotato sull'asse X ovvero dalla variazione Δ_{Roll} misurata dall'odometro esteso

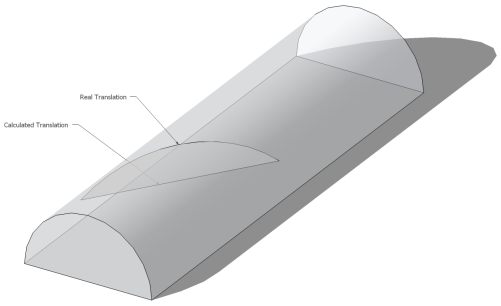


Figura 3.14: *Incertezza dalla componente Roll.*

Poiché la presenza di *Roll* in una traslazione identifica un movimento su una superficie non piana, aggiungiamo una quantità dipendente dal *Roll* nel tentativo di spiegare l'errore. Nella Figura 3.14 si possono notare la traslazione calcolata e la traslazione reale.

- da quanto il veicolo ha ruotato sull'asse Z ovvero dalla variazione Δ_{Yaw} misurata dall'odometro esteso (ricordiamo che il movimento Yaw viene calcolato dagli encoder sulle ruote).

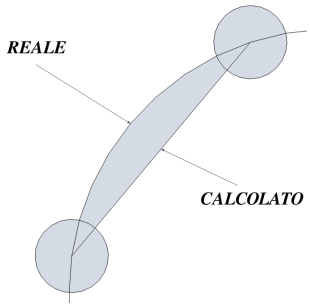


Figura 3.15: *Incertezza dalla componente Yaw.*

Poiché la presenza di *Yaw* in una traslazione identifica un movimento su una superficie non piana, aggiungiamo una quantità dipendente dal *Yaw* nel tentativo di spiegare l'errore. Nella Figura 3.15 si possono notare la traslazione calcolata e la traslazione reale.

Vediamo ora la parte *orientation*.

4. la rotazione finale sull'asse X $\delta_{Roll-Finale}$ è influenzata :
   - da quanto il veicolo ha ruotato sull'asse X ovvero dalla variazione Δ_{Roll} misurata dall'odometro esteso (ricordiamo che il movimento Roll viene calcolato dal sensore inerziale)
5. la rotazione finale sull'asse Y $\delta_{Pitch-Finale}$ è influenzata:
   - da quanto il veicolo ha ruotato sull'asse Y ovvero dalla variazione Δ_{Pitch} misurata dall'odometro esteso (ricordiamo che il movimento Roll viene calcolato dal sensore inerziale)
6. la rotazione finale sull'asse Z $\delta_{Yaw-Finale}$ è influenzata:
   - da quanto il veicolo ha ruotato sull'asse Z ovvero dalla variazione Δ_{Yaw} misurata dall'odometro esteso (ricordiamo che il movimento Yaw viene calcolato dagli encoder sulle ruote).
   - da quanto spazio è stato percorso dal veicolo: a movimenti più estesi corrisponde una maggiore probabilità di errore cumulato sulle misure odometriche

Vengono così definite le deviazioni standard (formule (3.18)...(3.23)) che verranno applicate ai sei atti di moto elementari precedentemente individuati. Infine, per meglio controllare il comportamento del modello, sono state applicate delle componenti α (una per ogni δ) affinché sia possibile dare più o meno peso a ciascuna delle componenti del sistema.

$$\sigma_{yaw1} = \alpha_1 \cdot \delta_{yaw1} + \alpha_2 \cdot \delta_{trans} \tag{3.18}$$

$$\sigma_{pitch1} = \alpha_3 \cdot \Delta_z \tag{3.19}$$

$$\sigma_{trans} = \alpha_4 \cdot \delta_{trans} + \alpha_5 \cdot \Delta_{Yaw} + \alpha_6 \cdot (\Delta_{Roll} + \Delta_{Pitch}) \tag{3.20}$$

$$\sigma_{Roll-Finale} = \alpha_7 \cdot \Delta_{Roll} \tag{3.21}$$

$$\sigma_{Pitch-Finale} = \alpha_8 \cdot \Delta_{Pitch} \tag{3.22}$$

$$\sigma_{Yaw-Finale} = \alpha_9 \cdot \Delta_{Yaw} + \alpha_{10} \cdot \delta_{trans} \tag{3.23}$$

E' opportuno osservare come le *sigma* vengano composte in modo diverso a seconda dell'origine dell'incertezza ovvero essa sia incertezza proveniente dal sistema encoder oppure IMU. Ci aspettiamo che l'incertezza sull'IMU non sia correlata con l'incertezza degli encoder: è il caso di $\sigma_{Roll-Finale}$, $\sigma_{Pitch-Finale}$ e σ_{pitch1} , calcolate esclusivamente dalla parte IMU dell'odometro esteso. Viceversa σ_{trans} è influenzato sia dalle letture degli encoder, sia dalle misure dell'IMU (vedi Figura 3.14 Figura 3.12 e Figura 3.15).

I sei atti di moto elementari composti con le loro incertezze saranno quindi i seguenti:

$$\hat{\delta_{yaw1}} = \delta_{yaw_1} + SAMPLE \underbrace{\{\alpha 1 \cdot \delta_{yaw1} + \alpha 2 \cdot \delta_{trans}\}}_{\sigma_{yaw1}} \tag{3.24}$$

$$\hat{\delta_{pitch1}} = \delta_{pitch_1} + SAMPLE \underbrace{\{\alpha 3 \cdot \Delta z\}}_{\sigma_{pitch1}} \tag{3.25}$$

$$\hat{\delta_{trans}} = \delta_{trans} + SAMPLE \left(\underbrace{\alpha_4 \cdot \delta_{trans} + \alpha_5 \cdot \Delta_{Yaw} + \alpha_6 \cdot (\Delta_{Roll} + \Delta_{Pitch})}_{\sigma_{trans}} \right) \tag{3.26}$$

$$\hat{\delta_{Roll-Finale}} = \delta_{Roll-Finale} + SAMPLE\underbrace{(\alpha_7 \cdot \Delta_{Roll})}_{\sigma_{Roll-Finale}} \tag{3.27}$$

$$\hat{\delta_{Pitch-Finale}} = \delta_{Pitch-Finale} + SAMPLE\underbrace{(\alpha_8 \cdot \Delta_{Pitch})}_{\sigma_{Pitch-Finale}} \tag{3.28}$$

$$\hat{\delta_{Yaw-Finale}} = \delta_{Yaw-Finale} + SAMPLE\underbrace{(\alpha_9 \cdot \Delta_{YAW} + \alpha_{10} \cdot \delta_{trans})}_{\sigma_{Yaw-Finale}} \tag{3.29}$$

Nel caso in cui non si disponga di un odometro esteso come quello descritto nel Paragrafo 3.3.1 i valori ($\Delta_{Pitch} = 0, \Delta_{Roll} = 0$ e $\Delta_z = 0$), vengono utilizzate le aspettative di incertezza a priori su quanto possa cambiare un certo valore.

3.3.5 Algoritmo di Sampling

Per generare un errore da applicare ad ogni componente del sistema si è reso necessario l'utilizzo di un metodo per la generazione di numeri casuali distribuiti secondo una distribuzione normale a media nulla e varianza opportunamente dimensionata. Tra i numerosi metodi presenti in letteratura è stato scelto il metodo di Box Mueller [22] che permette di generare coppie di valori indipendenti e normalmente distribuiti.

3.3.6 Considerazioni aggiuntive e dimensionamento delle deviazioni standard

Verranno ora descritte alcune considerazioni introdotte allo scopo di rendere più robusto il modello.

Soglie minime

Sono utilizzate nel caso in cui l'algoritmo di sampling dia dei valori troppo piccoli. Le soglie minime introducono e garantiscono la minima dispersione

dei dati necessaria per il corretto funzionamento del modello. L'introduzione di questi parametri riflette il tentativo di correzione del fenomeno della *particle deprivation* visto nel Capitolo 2.

parametro	σ_{min}
yaw1_min	0.01
pitch1_min	0.01
trans_min	0.02
roll_min	0.01
pitch_min	0.08
yaw_min	0.05

Tabella 3.1: *Valori di σ_{min} determinati*

Soglie massime

A differenza delle precedenti, le soglie massime sono state inserite affinché il sistema possa gestire situazioni in cui l'odometro esteso non fornisca dei delta a sei dimensioni. In questo caso i valori di soglia rappresentano il valore massimo di incertezza a priori (si prevede che mediante l'utilizzo di un sensore la stima dei movimenti sia migliore di una stima ottenuta senza sensori). Le σ_{max} associate a ciascun parametro del modello devono quindi essere opportunamente grandi affinché possano essere generate misure sufficientemente disperse attorno al valore medio da rappresentare/spiegare i possibili cambi di valore del grado di libertà.
Per dimensionare le σ_{max} si è considerata una velocità massima del veicolo di 25km/h ed una frequenza di acquisizione dati sensoriali di 20hz. Sotto queste ipotesi sono stati calcolati i valori di σ_{max} di Tabella 3.2. Per la determinazione delle σ_{max}, che corrispondono al $\delta_{i-esimo}$ indice di dispersione del modello odometrico, abbiamo utilizzato delle stime a priori sulle incertezze dei sensori utilizzati, considerando la nostra stima come valore di 3σ e la diseguaglianza Chebyshev.

parametro	σ_{max}
yaw1_max	0.26
pitch1_max	0.07
trans_max	0.01
roll_max	0.1
pitch_max	0.1
yaw_max	0.1

Tabella 3.2: *Valori di σ_{max} determinati*

Queste soglie possono essere utilizzate in aggiunta alle soglie minime.

Capitolo 4

Sistema di autolocalizzazione 6DoF

Passiamo ora a descrivere l'algoritmo di localizzazione utilizzato nella tesi. AMCL-6DoF è un algoritmo di localizzazione Monte Carlo ispirato al metodo proposto da Thrun *et al.* [5] e che ricalca la struttura già esistente del software creato da Brian Gerkey [23]. L'implementazione del programma ha richiesto un lungo periodo di analisi del precedente lavoro seguito dall'aggiornamento delle strutture dati e degli algoritmi accessori. Il passaggio dalle tre dimensioni utilizzate nel piano alle sei dimensioni necessarie per rappresentare lo stato del robot in uno spazio tridimensionale ha richiesto lo studio della matematica dei quaternioni e della libreria BulletPhysics, utilizzata dal framework ROS [6] come motore per le trasformazioni geometriche. Anche lo stesso framework ROS è stato oggetto di approfonditi studi, necessari a comprendere alcuni malfunzionamenti dovuti alla gestione delle code di messaggi utilizzate nell'algoritmo originale (si veda il Paragrafo 4.7.2 per una descrizione dettagliata del sistema di interscambio messaggi ROS).

4.1 L'algoritmo AMCL-6DoF

La caratteristica comune a tutti gli algoritmi appartenenti alla famiglia MCL o *Monte Carlo Localization* è la rappresentazione della localizzazione mediante una distribuzione di probabilità $bel(x_t)$ non parametrica. Essa viene infatti

Input: $(\mathcal{X}_{t-1}, u_t, z_t, m)$
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ *to* M **do**
    $x_t^{[m]} = sample\_motion\_model(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = sample\_measurement\_model(z_t, x_t^{[m]}, m)$
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
for $m = 1$ *to* M **do**
    campiona i con probabilità $\propto w_t^{[i]}$
    aggiungi $x_t^{[i]}$ a $\mathcal{X}_t$
clustering di $\mathcal{X}_t$
restituisci $pose_t, \mathcal{X}_t$

Tabella 4.1: *L'algoritmo MCL*

descritta da un set di $\mathcal{M}$ particelle $\mathcal{X}_t = \{x_t^{[1]}, x_t^{[2]}, \cdots, x_t^{[\mathcal{M}]}\}$ e rappresenta una applicazione reale del filtro a particelle visto nei Paragrafi 2.1.2 e successivi, al problema della localizzazione [21]. L'accuratezza con cui la distribuzione è rappresentata è direttamente proporzionale al numero di particelle utilizzate. La Figura 4.4 mostra un esempio di MCL in un ambiente monodimensionale come quello rappresentato nel Paragrafo 2.2.1. Qui l'incertezza iniziale è rappresentata attraverso l'introduzione di un set di particelle distribuite in modo uniforme sull'intero spazio degli stati; ciascuna particella rappresenta diverse ipotesi di $pose_t$.
L'algoritmo MCL, presentato nella sua forma base in in Tabella 4.1, è uno degli algoritmi di localizzazione più utilizzato in robotica; può essere adoperato sia per problemi di *global localization* che per problemi di *local localization* e l'utilizzo di campioni per descrivere la distribuzione di probabilità della *pose* permette all'algoritmo di creare ipotesi distinte e concorrenti di localizzazione (in termini probabilistici, adattarsi a distribuzioni unimodali e multimodali). Per stimare la corretta posizione del robot l'algoritmo effettua ciclicamente una serie di operazioni sul set di particelle $\mathcal{X}_t$.

In primo luogo viene effettuata la fase di *predizione* del moto del robot. Per quanto concerne questa fase le azioni di controllo vengono descritte attra-

verso il modello proposto in questa tesi e visto nel Paragrafo 3.3.2. La linea 4 della Tabella 4.1 aggiorna le posizioni del vettore $\mathcal{X}_t$ richiamando la procedura opportuna che implementa l'*odometry model 6DoF*. Successivamente viene effettuato un passaggio di *integrazione* delle misure effettuate dai sensori. Il risultato dell'applicazione della misura sensoriale z_t è visibile nella Figura 4.4 (b) dove ad ogni particella è stato applicato il proprio *importance weight* ovvero la verosimiglianza del x_t^i i-esima particella nei confronti della misura rilevata $p(z_t|x_t^i, m)$; nell'algoritmo di Tabella 4.1 la misura sensoriale viene integrata nel filtro mediante l'applicazione del modello sensoriale Beam Model visto nel Paragrafo 2.4.1 (riga 5 di Tabella 4.1). Una volta calcolato il peso delle singole particelle viene eseguito un terzo passaggio, il *resampling*; questo procedimento consiste nella generazione di un nuovo set di particelle non pesate. La scelta delle pose associate alle nuove particelle è influenzata dal peso delle particelle stesse prima del resampling (si veda Figura 4.4 (c)). Il procedimento di resampling è descritto in Tabella 4.1 dalle righe 7-9.

Infine, prima di restituire un valore di localizzazione, l'algoritmo deve effettuare un ultimo passaggio che consiste nella valutazione delle nuove particelle generate dal processo di resampling. Questa operazione, ottenuta mediante un processo di clustering (si veda il successivo Paragrafo 4.1.3), permette di generare la migliore stima $pose_t$ ottenuta dal set di particelle $\mathcal{X}_t$ (righe 10 e 11 di Tabella 4.1).

4.1.1 Applicazione del Beam Model

L'algoritmo Beam Model visto nel Paragrafo 2.4.1, utilizzato dal filtro MCL per le operazioni di peso delle particelle, prevede la creazione di una distribuzione condizionale $p(z_t|x_t, m)$ basata sulla lettura sensoriale. Affinché questo modello possa generare tale distribuzione, l'algoritmo ha la necessità di confrontare i valori di scansione ottenuti dai LIDAR con le corrispondenti misure z_k^{t*} calcolate nella $\overline{pose_t}$ (ovvero dopo la fase di predizione) indicata da ciascuna particella. Le misure z_k^{t*} vengono determinate attraverso un procedimento di raycasting all'interno della mappa tridimensionale.

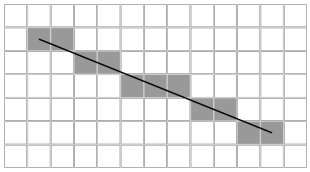


Figura 4.1: *Bresenham 2D*

4.1.2 Raycasting 3D con l'algoritmo di Bresenham

L'algoritmo di Bresenham [24] viene utilizzato per il calcolo del parametro z_t^{k*} utilizzato nel modello sensoriale Beam Model (si veda Paragrafo 2.4.1) e, nella sua versione bidimensionale, è descritto in Tabella 4.2. L'algoritmo permette, dati due punti, di determinare quali celle dello spazio discretizzato vengono percorse dal raggio laser, verificando in modo iterativo la presenza di ostacoli nella mappa. Questa procedura permette al Beam Model di effettuare una valutazione probabilistica della misura ottenuta. L'idea alla base dell'algoritmo di Bresenham è raffigurata in Figura 4.1.
Nel caso dell'implementazione su AMCL-6DoF i punti che vengono utilizzati corrispondo all'origine da cui il raggio laser viene emesso (la pose del LIDAR in 6DoF, considerando inoltre la variazione angolare dell'$i-esima$ scansione) ed al punto più lontano accertabile (ovvero la massima portata del sensore). Partendo dalla cella corrispondente alla pose del LIDAR, ogni singola cella della mappa, individuata con l'algoritmo di Bresenham ed attraversata dal raggio laser, verrà controllata fino a quando non ne verrà trovata una occupata oppure si raggiunge la cella corrispondente alla massima portata. A questo punto la distanza z_k^*, calcolata in modo geometrico, verrà confrontata con la misura z_k restituita dal LIDAR, permettendo il calcolo della distribuzione $p(z_t|x_t, m)$.
Per il calcolo della pose di origine del beam del LIDAR sono state utilizzate le procedure di rototraslazione descritte nel Paragrafo 4.4.

4.1.3 Clustering

Sebbene l'idea alla base degli algoritmi a particelle consista nel rappresentare la distribuzione continua dello stato stimato del robot con una sua approssimazione discreta, il fine dell'algoritmo AMCL-6DoF è quello di generare una

```
1  Input: (x_0, y_0, x_1, y_1)
2  δ_x = x_1 − x_0
3  δ_y = y_1 − y_0
4  error=0
5  deltaerr = ‖δ_y / δ_x‖
6  y = y0
7  for x = x_0 to x_1 do
8      error = error + deltaerr
9      if error ⩾ 0.5 then
10         y = y + 1
11         error = error - 1.0
```

Tabella 4.2: *L'algoritmo Bresenham 2D nella sua versione più semplice*

pose per il robot. Si rende dunque necessaria una valutazione del belief campionato, da comunicare come risultato finale. Queste valutazioni richiedono un raggruppamento (clustering) delle particelle. Le metodologie che effettuano questo tipo di operazione sono note in letteratura come algoritmi di Density Extraction e si differenziano per la tipologia di risultato che si vuole ottenere e per la complessità di calcolo. Il clustering è l'ultimo blocco logico che viene effettuato dall'algoritmo AMCL-6DoF ed è introdotto proprio per l'estrazione di una unica pose da un insieme di particelle. L'algoritmo di clustering utilizzato in questo lavoro si basa sulla distanza, sia in termini spaziali che in termini di rotazione, che una pose può assumere e ricalca lo pseudocodice di Tabella 4.3. Descriveremo ora la procedura di clustering utilizzata, che consiste nell'assegnare ad ogni particella un valore corrispondente al cluster a cui appartiene. La prima operazione che l'algoritmo effettua è quella di contrassegnare il cluster di appartenenza di ciascuna cella ad un valore che identifica il non assegnamento (linea 2 e 3 di Tabella 4.3). Successivamente l'algoritmo effettua un confronto tra ogni coppia di elementi del set di particelle non appartenente a nessun cluster, incrementando, se necessario, il numero di cluster ad ogni iterazione dell'intero procedimento di confronto (linee 5-6 di Tabella 4.3). Se la distanza spaziale ed angolare rien-

```
1  Algorithm Clustering(ParticleSet X);
2  for i=0 to N do
3      X[i].cluster ← −1;
4  for i=0 to N do
5      if X[i].cluster = -1 then
6          X[i].cluster ← cluster_count + +;
7      for j=0 to N do
8          if X[j].cluster ≠ -1 then
9              continue;
10         if i==j then
11             continue;
12         translation = calculate_euclidean_distance(X[i], X[j]);
           rotation = calculate_angle(X[i], X[j]);
13         if traslation <= traslation_threshold then
14             if rotation <= rotation_threshold then
15                 X[j].cluster ← X[i].cluster;
```

Tabella 4.3: *Pseudo codice dell'algoritmo di clustering utilizzato in AMCL-6DoF*

tra al di sotto di una soglia preimpostata e parametrizzabile le due particelle verranno inserite all'interno dello stesso cluster, contrassegnandole come già clusterizzate. Le linee 13-15 implementano questo confronto.

Una volta effettuato il processo di clustering sulle particelle si presenta il problema dell'estrazione di una unica *pose* dall'insieme dei cluster. Il problema viene affrontato nel modo seguente: utilizzando i pesi associati alle singole particelle viene innanzitutto determinato il cluster con maggior peso, che corrisponde all'area più densamente popolata da particelle. Successivamente il valore finale della *pose* viene calcolato effettuando una media pesata (viene utilizzato il peso della particella) delle singole variabili di stato che compongono le pose appartenenti al cluster selezionato. Sorge a questo punto la necessità di calcolare una media sia della parte spaziale (ovvero delle

componenti (x, y, z)) che della parte di rotazione tridimensionale attraverso le componenti (ϕ, ψ, θ), media che deve essere pesata su entrambe le componenti. Se la parte di calcolo della media pesata spaziale può considerarsi semplice, l'utilizzo delle componenti *roll pitch* e *yaw* per la parte di rotazione rende più complessa questa elaborazione, in quanto l'ordine in cui vengono composte le rotazioni elementari influisce sull'orientamento finale generato.

4.1.4 Medie di orientamenti

L'utilizzo della matematica dei quaternioni per la rappresentazione delle rotazioni in AMLC-6DoF permette di evitare le problematiche che derivano dalla composizione delle rotazioni attraverso le componenti *roll pitch* e *yaw*. La procedura consiste nella semplice somma delle singole componenti del quaternione seguita da una normalizzazione finale (4.1) [25] [26].

$$Q_{mean} = \frac{(Q^1 + Q^2 + \cdots + Q^N)}{\|Q^1 + Q^2 + \cdots + Q^N\|} \tag{4.1}$$

Nella Tabella A.3 è disponibile un esempio di calcolo di medie di quaternioni in linguaggio C++ ed utilizzando le strutture dati di ROS.

4.1.5 SLERP

Slerp [26] è l'acronimo di *Spherical Linear Interpolation* ed è un metodo per l'interpolazione di quaternioni. Questa operazione è necessaria nel caso in cui si voglia dare un peso a ciascuna ipotesi, durante il calcolo della media degli orientamenti. Ancora una volta l'utilizzo dei quaternioni rende semplice questa operazione. La procedura di SLERP applicata agli oggetti *quaternion* prende in input un secondo quaternione ed uno scalare t che corrisponde al rapporto da utilizzare nell'interpolazione. Nel caso di t=0 il risultato corrisponderà al quaternione sul quale è stata richiamata la procedura di SLERP, mentre nel caso di t=1 il risultato corrisponderà al secondo quaternione.

Nel caso di AMCL-6DoF e del problema del peso delle rotazioni il quaternione associato alle singole particelle viene scalato per la quantità corrispondente al peso della particella, utilizzato il quaternione identità come secondo quaternione

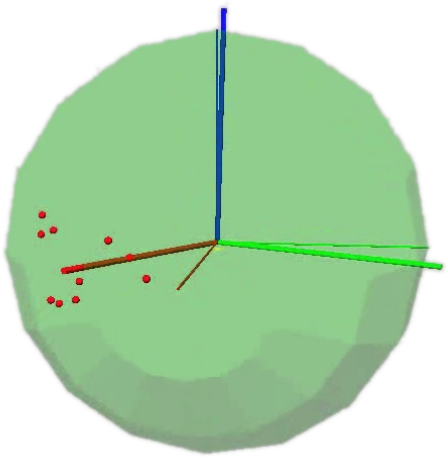


Figura 4.2: *Il risultato di una media di quaternioni. Nella figura in rosso gli orientamenti corrispondenti alle particelle. Il sistema di riferimento ruotato corrisponde alla media pesata degli orientamenti. Il sistema di riferimento più sottile corrisponde al quaternione identità utilizzato per il peso dei singoli orientamenti. Il codice sorgente è disponibile in Appendice A.3*

```
1 cluster->workspace_quaternion += tf::createQuaternionFromRPY(
    sample->pose.v[4],sample->pose.v[5],sample->pose.v[2]).slerp(
    tf::createIdentityQuaternion(),sample->weight);
```

Tabella 4.1: *Nell'esempio viene calcolato il quaternione pesato da aggiungere alla media degli orientamenti; esempio tratto dal codice di AMCL-6DoF, parte di clustering*

La procedura di SLERP è fornita dal framework BulletPhysics.

4.2 Varianti del filtro MCL

Affinché l'algoritmo visto fino ad ora possa risolvere situazioni come il problema del *kidnapped robot* (si veda il Paragrafo 2.2), è necessario adottare alcuni accorgimenti. Tipicamente, durante l'esecuzione, le particelle si concentrano intorno alla zona più probabile. Questo comportamento fa sì che le particelle *sopravvissute* siano concentrate attorno ad una unica *pose* rendendo l'algoritmo incapace di recuperare la posizione, nel caso questa diventi

ad un certo punto errata. Questo problema, che si presenta solitamente con l'utilizzo di un esiguo numero di particelle nel filtro, è molto significativo in quanto il processo stocastico che porta alla determinazione della corretta *pose* potrebbe accidentalmente scartare tutte le particelle situate nei pressi della corretta posizione, impedendo di fatto di ottenere una corretta localizzazione. Il fenomeno può essere affrontato in modo molto semplice introducendo particelle completamente a caso nel set $\mathcal{X}_t$. L'introduzione di particelle casuali ha il vantaggio della semplicità e richiede minime modifiche da parte del software; il numero di particelle da aggiungere può essere fisso oppure calcolato da alcune euristiche.

4.2.1 Augmented-MCL

L'algoritmo Augmented-MCL proposto in Tabella 4.4 [5] utilizza una euristica che si basa sul peso che viene associato ad ogni set di particelle dopo l'incorporazione delle misure sensoriali, tenendo traccia delle medie dei pesi a breve e lungo termine associate al set di particelle. Durante la fase di *resampling* viene aggiunta a caso una particella con una probabilità influenzata dalla divergenza tra la media a lungo ed a breve termine. L'idea alla base di questo algoritmo è la seguente: nel caso si abbia una media delle verosimiglianze a breve termine peggiore di quella a lungo termine, possiamo ragionevolmente affermare che la stima della posizione del robot sta diventando inaffidabile. In questo caso viene aggiunto un numero di particelle casuali proporzionale al rapporto tra le due medie.

$$max\{0.0, (1.0 - w_{fast}/w_{slow})\} \tag{4.2}$$

4.2.2 L'algoritmo KLD-Sampling

Come abbiamo precedentemente enunciato, la numerosità delle particelle utilizzate da questa famiglia di algoritmi è direttamente proporzionale alla accuratezza con cui la distribuzione di probabilità delle pose viene rappresentata, ma l'incremento del numero di particelle porta con sé un notevole dispendio computazionale sia in termini di risorse di calcolo, sia in termini di memoria. Se nel caso di una localizzazione globale è solitamente necessario un grande

numero di particelle per contemplare tutte le possibili pose del robot, le fasi di localizzazione locale ne richiedono in genere un numero molto inferiore (si veda Figura 4.3). L'algoritmo KLD-Sampling [3] [4] [5] descritto in Tabella 4.5 è stato ideato per adeguare dinamicamente il numero di particelle utilizzate. L'intuizione alla base di questo algoritmo è la seguente: generare nuove particelle nella fase resampling fintantoché siano state generate *sufficienti* particelle nei dintorni della posizione stimata. A tal fine lo spazio degli stati viene discretizzato in celle di uguale dimensione e il procedimento segue queste considerazioni: le prime particelle generate, ossia il valore k corrispondente alle celle non occupate, cresce pressoché con ogni nuova aggiunta; l'incremento di k corrisponde perciò ad in un incremento anche del valore M_x che identifica il numero di particelle richieste dalla particolare situazione di localizzazione. Tuttavia, nel tempo, sempre più celle risulteranno non vuote e quindi il numero di particelle richieste M_x aumenterà con meno frequenza facendo sì che il numero di particelle generate M raggiunga la soglia M_x. Quanto questo succeda velocemente dipende da quanto sono sparse le particelle nello spazio degli stati: man mano che la localizzazione migliora verranno utilizzate sempre meno particelle, adattandone quindi il numero in base alle effettive necessità.

In questa tesi abbiamo utilizzato un approccio ibrido che unisce le capacità di adattamento dell'algoritmo KLD alle potenzialità di recupero da soluzioni critiche fornito dall'approccio Augmented-MCL. Queste due tecniche risultano fondamentali in un ambiente 3D, nel quale lo spazio degli stati viene aumentato considerevolmente portando da 3 a 6 i parametri necessari per l'identificazione della posizione e dell'orientamento del robot. La Tabella 4.6 mostra lo pseudocodice dell'intero algoritmo e corrisponde all'utilizzo contemporaneo di tutti gli algoritmi visti in precedenza.

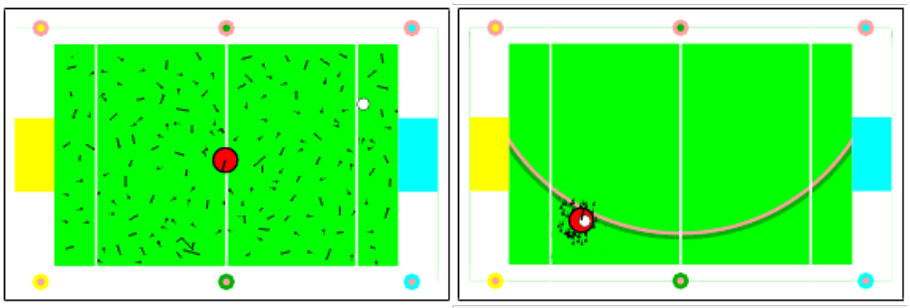


Figura 4.3: *A sinistra una situazione di localizzazione globale; a destra una situazione di localizzazione locale; il numero di particelle necessarie per descrivere il belief, che ha come dominio lo spazio degli stati, è molto minore nel caso di localizzazione locale.*

Input: $(\mathcal{X}_{t-1}, u_t, z_t, m, w_{slow}, w_{fast})$
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ *to* M **do**
    $x_t^{[m]} = sample\_motion\_model(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = sample\_measurement\_model(z_t, x_t^{[m]}, m)$
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    $w_{avg} = w_{avg} + \frac{1}{M} w_t^{[m]}$
$w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
$w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
for $m = 1$ *to* M **do**
    if *con probabilità* $max\{0.0, 1.0 - w_{fast}/w_{slow}\}$ **then**
        aggiungi una pose casuale ad $\mathcal{X}_t$
    else
        campiona i con probabilità $\propto w_t^{[i]}$
        aggiungi $x_t^{[i]}$ a $\mathcal{X}_t$
clustering di $\mathcal{X}_t$
restituisci $pose_t, \mathcal{X}_t, w_{slow}, w_{fast}$

Tabella 4.4: *L'algoritmo Augmented-MCL*

Input: $(\mathcal{X}_{t-1}, u_t, z_t, m, \epsilon, \sigma)$
$\bar{\mathcal{X}}_t = \emptyset$
$M = 0, M_x = 0, k = 0$
forall *binH* **do**
    b=empty
repeat
    campiona i con probabilità $\propto w_t^{[i]}$
    $x_t^{[m]} = sample\_motion\_model(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = sample\_measurement\_model(z_t, x_t^{[m]}, m)$
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
    if $x_t^{[m]}$ *appartiene ad un bin b vuoto* **then**
        k=k+1
        b=non-vuoto
        if $k > 1$ **then**
            $M_x = \frac{k-1}{2\epsilon}\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\sigma}\}^3$
    M=M+1
until $M < M_x$ *or* $M < M_{xmin}$
clustering di $\mathcal{X}_t$
restituisci $pose_t, \mathcal{X}_t$

Tabella 4.5: *L'algoritmo KLD-Sampling*

Input: LIDAR-SCAN, X_{t-1}
Output: $pose_t$, X_t
foreach *particle* X_i *in a separate thread* **do**
    apply odometry model
foreach *particle* X_i *in a separate thread* **do**
    weight=0
    foreach *beam, in a separate thread* **do**
        weight += Bresenham algorithm
$w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$
$w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$
repeat
    if *con probabilità* $max\{0.0, 1.0 - w_{fast}/w_{slow}\}$ **then**
        aggiungi una pose casuale ad $\mathcal{X}_t$
    else
        campiona i con probabilità $\propto w_t^{[i]}$
        $x_t^{[m]} = sample\_motion\_model(u_t, x_{t-1}^{[m]})$
        $w_t^{[m]} = sample\_measurement\_model(z_t, x_t^{[m]}, m)$
        $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
        if $x_t^{[m]}$ *appartiene ad un bin b vuoto* **then**
            k=k+1
            b=non-vuoto
            if $k > 1$ **then**
                $M_x = \frac{k-1}{2\epsilon}\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\sigma}\}^3$
        M=M+1
until $M < M_x$ *or* $M < M_{xmin}$
clustering X_t

Tabella 4.6: *L'algoritmo AMCL-6DoF*

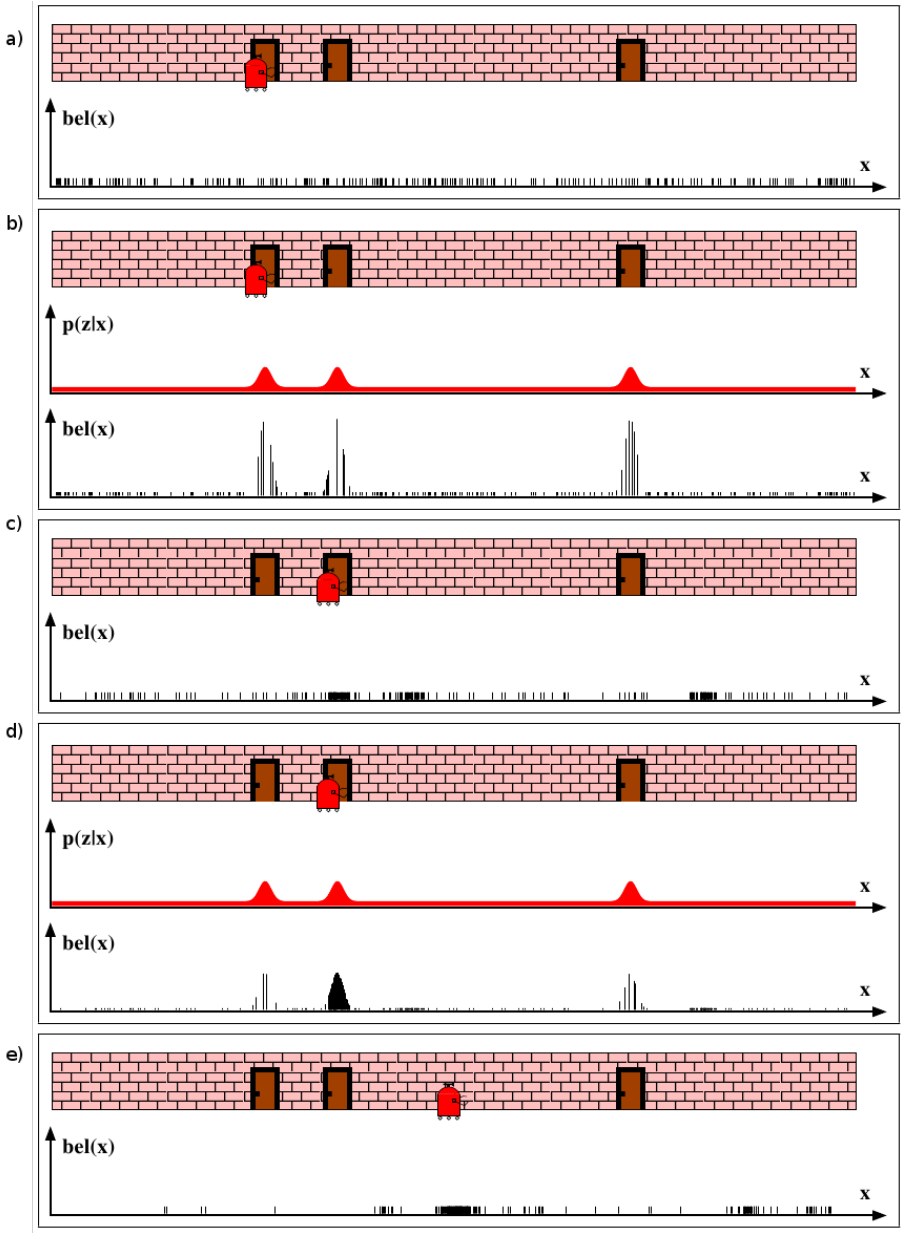


Figura 4.4: *Monte Carlo Localization: applicazione di un filtro a particelle per il problema della localizzazione*

4.3 Creazione di mappe 3D

Lavorare in un ambiente tridimensionale ha innanzitutto comportato il problema della rappresentazione dello spazio. Il nodo Map Server [27] di ROS è infatti ideato per l'utilizzo di sole mappe bidimensionali dove l'ambiente viene uniformemente suddiviso in una matrice di celle, ciascuna identificata dalla coppia di coordinate cartesiane (x, y), ma non prevede la possibilità di rappresentare anche la terza coordinata spaziale. Si è pensato quindi di ampliare la modalità di rappresentazione a celle introducendo la terza dimensione, passando dalla *cella* bidimensionale all'analogo *voxel* identificato da una terna (x, y, z). La struttura dati che rappresenta il singolo voxel è definita in Tabella 4.2.

```
typedef struct
{
  char  occ_state;
} map_cell_t;

typedef struct
{
  double origin_x, origin_y, origin_z;
  double scale;
  int size_x, size_y, size_z;
  map_cell_t *cells;
} map_t;
```

Tabella 4.2: *Struttura dati per la mappa dei voxel*

In assenza di strumenti per la costruzione di una mappa di voxel realistica, le modalità con cui sono state create le mappe sono due e sono state utilizzate rispettivamente in modalità di sviluppo e simulazione ed in modalità di test in condizioni reali. Per quanto riguarda la prima fase è stato utilizzato un procedimento di estrusione da una mappa bidimensionale che permettesse di verificare la correttezza degli algoritmi sviluppati (Odometry Model, Beam Model, Clustering).
Il procedimento di estrusione non può essere però adottato negli ambienti reali. Le informazioni sulla pendenza stradale risulterebbero, infatti, completamente assenti, rendendo inefficace la procedura di localizzazione 6DoF.

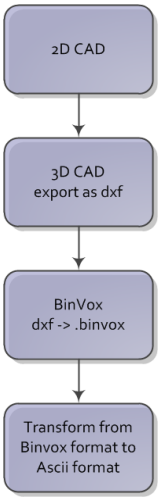


Figura 4.5: *Il processo di creazione delle mappe voxel 3d*

In questo caso la creazione delle mappe ha seguito una procedura più lunga ed elaborata. Partendo da modelli CAD bidimensionali sono stati creati i rispettivi CAD tridimensionali sui quali è stato effettuato un processo di *voxeling*. In questa fase l'ambiente 3D viene scomposto in una matrice di voxel mediante l'utilizzo del tool BinVox [28]. Una volta generato il file BinVox, questo è stato convertito in formato ASCII, pronto per essere letto da AMCL-6DoF. Questa lunga catena di operazioni porta con sé le limitazioni dell'ambiente BinVox, ovvero il numero massimo di voxel in cui può essere suddivisa la mappa CAD. Questo valore è attualmente fissato a 1024, per ciascuna dimensione. Si rimanda all'Appendice D per un esempio di procedura di voxeling. Tale procedura sarà in futuro sostituita da algoritmi di mapping 3D con l'ausilio di sensori LIDAR, con il fine di ottenere ricostruzioni ad alta fedeltà anche di parti del mondo per le quali non è possibile pensare di ottenere informazioni da un disegno CAD (come ad esempio la reale pendenza della pavimentazione oppure aree per le quali non esiste un disegno). Un esempio di ambiente ricostruito con tecniche LIDAR è visibile in Figura 4.6.

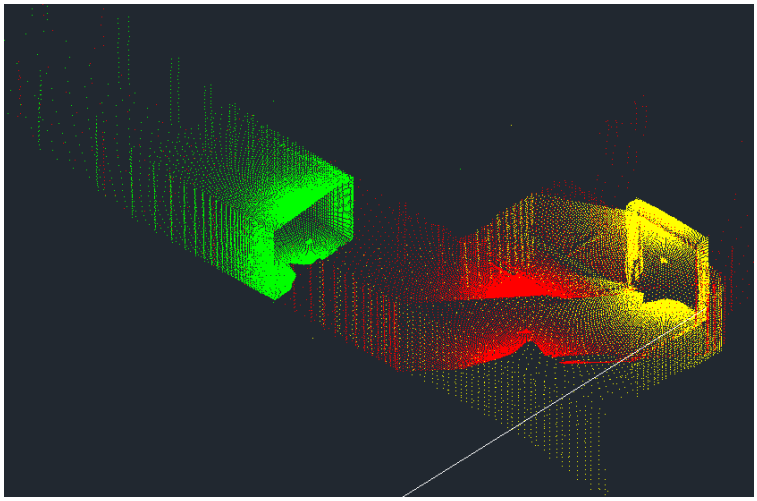


Figura 4.6: *In questa figura si può osservare una ricostruzione dell'uscita del garage U5 ottenuta con un LIDAR. I punti rossi verdi e gialli corrispondono a tre scansioni successive.*

4.4 Utilizzare ROS per applicare le rototraslazioni

Prima di introdurre l'algoritmo di raycasting è indispensabile descrivere come vengono effettuate le trasformazioni geometriche all'interno del framework ROS. Per rappresentare una rototraslazione tra due sistemi di riferimento si utilizzano le seguenti strutture dati presenti all'interno dello stack *Ros Geometry*.

- btQuaternion: un quaternione rappresenta la relazione tra lo stesso vettore in due sistemi di riferimento (la struttura dati è derivata dal framework BulletPhysics); è possibile costruire un quaternione dalle componenti Roll, Pitch e Yaw utilizzando le funzioni fornite dal nodo TF di ROS:

```
1 btQuaternion temp = createQuaternionFromRPY(Roll,Pitch,Yaw)
2 btQuaternion temp = createQuaternionFromYaw(Yaw)
```

Tabella 4.3: *Esempio di creazione rotazione*

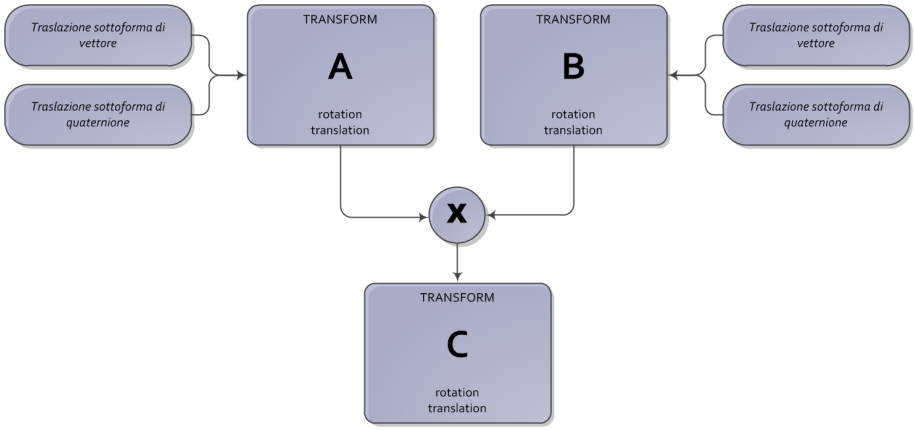


Figura 4.7: *In questa figura la composizione delle trasformazione*

- btVector3: derivata nuovamente dal framework BulletPhysics rappresenta una traslazione, identificata dai delta sulle singole componenti (x, y, z).

```
1 Vector3 traslation = Vector3(DeltaX,DeltaY,DeltaZ)
```

Tabella 4.4: *Esempio di creazione traslazione*

- tf::Transform: rappresenta una rotazione seguita da una traslazione, in una unica struttura dati.

```
1 cart_frame.setOrigin(tf::Vector3(DeltaX,DeltaY,DeltaZ))
2 cart_frame.setRotation(tf::createQuaternionFromYaw(Yaw))
```

Tabella 4.5: *Esempio di creazione trasformazione*

Una volta definite le operazioni base per la creazione di una trasformazione, possiamo comporre le trasformazioni attraverso l'operazione di moltiplicazione. Questa procedura sarà alla base di tutte le trasformazioni a sei gradi di libertà utilizzate nel progetto. Si veda Appendice A, sezione A.4, per un esempio di composizione di trasformazioni.

4.5 Multithreading

L'aumento delle particelle occorrenti per la descrizione della distribuzione di probabilità associata alla pose del robot accresce notevolmente sia i tempi di calcolo che lo spazio occupato in memoria. Oltre a ciò, l'introduzione del calcolo tridimensionale, sia nelle procedure relative al modello di moto che nelle procedure di raycasting necessarie al modello sensoriale, fanno sì che la singola esecuzione dell'algoritmo AMCL-6DoF richieda una quantità elevata di calcoli. La particolare natura degli algoritmi MCL e Bresenham, unita alla moderna presenza di piattaforme hardware multi processore, ci ha spinto a ridefinire gli algoritmi descritti nei paragrafi precedenti in ottica multithreading. E' stato così possibile parallelizzare sia il calcolo delle convoluzioni delle particelle, sia i vari raycasting effettuati con l'algoritmo di Bresenham. Questo sforzo ci ha ampiamente ripagato: abbiamo infatti verificato una riduzione del tempo di calcolo all'incirca lineare con l'aumento del numero di processori disponibili. L'utilizzo delle operazioni multithreading ha permesso quindi di avere una ottima scalabilità della soluzione proposta nella tesi.
Le procedure di localizzazione concorrenti sono state implementate utilizzando la libreria Threading Building Blocks (TBB) [29] sviluppata da Intel®, che consiste in una collezione di algoritmi e strutture dati C++ atta a semplificare l'utilizzo delle procedure concorrenti, ottimizzando dinamicamente l'utilizzo di risorse cache. Si faccia riferimento all'Appendice A per un dettaglio sui particolari algoritmi utilizzati.

In questo capitolo presenteremo il robot mobile e le componenti sia hardware che software che si è reso necessario sviluppare durante la tesi.

4.6 Il robot ed i sensori

Il robot mobile urban outdoor utilizzato in questo progetto è un Golf Cart elettrico serie Alpaca, prodotto da Ecology Runner [1] (si veda Figura 4.8). Il veicolo, denominato semplicemente *Cart*, è parte integrante dell'attività di ricerca condotta dal laboratorio IRA con il progetto USAD [2], che si pone l'o-

[1] http://www.ecologyrunner.it
[2] http://irawiki.disco.unimib.it

biettivo di permettere ad un veicolo autonomo di effettuare la navigazione in ambito urban outdoor. A livello tecnico il movimento del Cart è controllato da una scheda di controllo ad-hoc realizzata internamente al laboratorio IRA e che si interfaccia con il nostro sistema per mezzo di una connessione USB; la scheda consiste in un microcontrollore della serie DS-Pic di Microchip. Questo, tra le altre attività, controlla le letture effettuate da una coppia di emettitori/ricevitori infrarossi montati sull'asse delle ruote ed una ruota ottica collocata nel lato interno della ruota (si veda Figura 4.9). Per quanto riguarda il posizionamento dei sensori LIDAR sul Cart è stata sviluppata una base di sostegno denominata *u-rovesciata* che si configura come un paraurti supplementare sul quale assicurare i sensori laser. Durante la fase di realizzazione di questo sostegno è stata posta particolare attenzione alla scelta dei punti di ancoraggio al telaio del Cart ed al materiale da utilizzare per la costruzione, allo scopo di minimizzare l'effetto molla che tale struttura avrebbe introdotto una volta posizionati i laser alle estremità.

Figura 4.8: *Il Cart allo stato di sviluppo attuale. Sul frontale la u-rovesciata sulla quale sono attaccati i 3 LIDAR.*

Figura 4.10: *Le diverse fasi della creazione del sostegno per i LIDAR: creazione CAD del modello (a), prototipo di cartone (b) e struttura finale sul Cart (c)*

Figura 4.9: *Il sistema di encoder montato sull'asse delle ruote. Nella figura sono indicati i due emettitori/ricevitori infrarossi che permettono di individuare il movimento della ruota.*

Per quanto concerne la parte sensoriale sono stati utilizzati i seguenti due LIDAR:

- Sick LMS111 (Appendice C.1). Questo sensore permette l'acquisizione di ostacoli su un singolo piano di scansione con un campo visivo di 270° (grazie ad uno specchio rotante interno) alla risoluzione angolare di 0.5° per una frequenza impostabile di 25hz o 50hz. Abbiamo utilizzato due di questi LIDAR posizionandoli alle estremità destra e sinistra del supporto *u-rovesciata* in modo tale da avere un campo di scansione praticamente di 360°, fatta eccezione per l'area posteriore del cart; affinché la grande ampiezza di scansione non creasse autoletture del veicolo stesso si è reso però necessario ridurre il campo visivo per una ampiezza di 25° per estremità. I sensori sono stati montati in modo da risultare sia allineati tra di loro sia allineati al piano stradale in

condizioni di riposo del veicolo (senza carichi aggiuntivi). La portata fino a 20 metri di questi sensori, insieme all'ampio campo visivo ed alla loro affidabilità in diverse condizioni di luce ambientale, permettono di ottenere ottime localizzazioni in ambienti bidimensionali; per quanto concerne gli ambienti tridimensionali l'utilizzo dei soli Sick111 non è stato sufficiente: le numerose approssimazioni introdotte dai vari modelli che cooperano nell'incarico di localizzazione (discretizzazione della mappa, aggiunta di rumore alle misure del sensore tramite il beam model, la considerazione del raggio laser come raggio privo di dimensione e con dimensione costante [9]) portano ad una perdita di almeno un grado di libertà, ovvero la quota z in ambienti con pareti perpendicolari al pavimento.
Per quanto concerne l'interfacciamento al nostro sistema sono stati utilizzati inizialmente i driver open source forniti dal pacchetto ROS LMS1xx [3], per poi procedere all'aggiornamento dei driver scritti da Paolo Surricchio e Francesco Sacchi all'interno dell'attività del laboratorio IRA; le modifiche proposte sono state di livello concettuale e si rimanda alla descrizione del successivo Paragrafo 4.7.3

Figura 4.11: *Il sensore LIDAR Sick LMS111-10100*

- Il Sick LDMRS-4001 (Paragrafo C.2) il campo visivo è limitato a 110°, ma la peculiarità di questo sensore è la disponibilità di quattro livelli di scansione contemporanei (tecnologia *Sick multilayer*) orientati a ventaglio con una spaziatura angolare di 0.8° per ciascun layer (si veda Figura 4.12b). Per quanto concerne la portata, il Sick LDMRS-4001 è capace di individuare oggetti fino ad una distanza di 100 metri, con una risoluzione di profondità costante di 4 cm ed una risoluzione angolare

[3] http://www.ros.org/wiki/LMS1xx

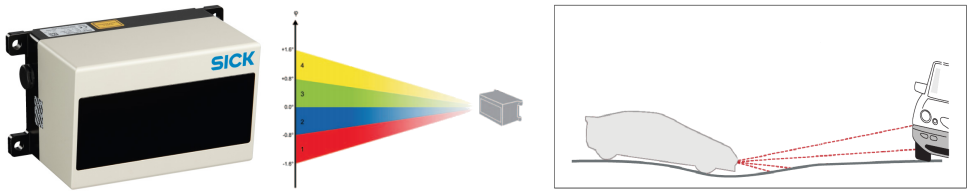


Figura 4.12: *Il sensore LIDAR Sick LD-MRS400001 (a); indicazione dei quattro piani di scansione (b); l'utilizzo della tecnologia multilayer per l'individuazione simultanea di ostacoli e orientamento del veicolo (c)*

di 0.25°. L'utilizzo di questo sensore è molto utile al fine di ottenere una corretta localizzazione in un mondo tridimensionale. Il sensore è stato posizionato sulla parte anteriore della *u-rovesciata* inclinato verso il basso, facendo sì che il primo fascio dall'alto risulti parallelo all'asse del piano stradale in condizioni di riposo del veicolo ed allineato al piano dei due Sick 111.

La configurazione del Sick LDMRS-4001 in aggiunta ai due laser Sick LMS111 ci consente di avere sei fasci laser su vari orientamenti: tre verso il piano stradale e tre orizzontali; l'insieme di queste misure porta ad avere una scansione tridimensionale dell'ambiente sufficiente alla localizzazione in 6DoF. E' fondamentale notare come l'assenza del sensore multilayer non permetta di disambiguare due pose con differente quota, nelle condizioni di piano di scansione parallelo al fondo stradale dei due Sick LMS111.

- Per misurare le informazioni relative all'assetto del veicolo si è l'utilizzato un sensore inerziale. La scelta è caduta sul sensore MTI-xsens che offre in un unico package diversi sensori quali giroscopi, accelerometri e magnetometri; l'MTI-xsens offre i dati di rollio e beccheggio necessari per un migliore funzionamento del modello di moto Odometry Model 6DoF descritto nel Paragrafo 4. L'interfacciamento al sistema avviene attraverso connessione USB ed il nodo ROS xsens-mti [4].

[4] http://www.ros.org/wiki/xsens_mti

Figura 4.13: *Il sensore inerziale MTI-xsens*

4.7 Linguaggio, middleware, librerie e calcolatori utilizzati

Tutto il software prodotto nell'ambito di questa tesi è stato sviluppato in linguaggio C++ in ambiente Linux (Ubuntu 10.10, kernel 2.6.35-32-generic-pae a 32 bit) e compilato con gcc 4.4.5 (repository Ubuntu/Linaro 4.4.4-14ubuntu5). Per quanto concerne le tecniche di parallelizzazione/multithreading del codice sono state utilizzate le librerie Intel® Threading Building Block (TBB [5]) (si veda l'Appendice A) versione 3.0 e 4.0.

4.7.1 I calcolatori utilizzati

I calcolatori che sono stati utilizzati sono:

Intel® Core2Duo™ (dual core)
Intel® Core™ i7-740QM, 6mb cache, 8gb ram (quad core)
Intel® Core™ i7-940, 8mb cache, 8gb ram (quad core)

La scelta dei calcolatori è stata fatta in modo da permettere una adeguata fase di simulazione nella quale il numero di particelle utilizzate dagli algoritmi MCL è stato volutamente tenuto molto alto. Durante i test abbiamo riscontrato come i processori di categoria inferiore (dual core) nell'ambito di global localization all'interno del garage U5 non riescano ad eseguire l'algoritmo in tempistiche ottimali, introducendo ritardi sulla localizzazione dell'ordine dei secondi. Segnaliamo come tale fenomeno venga parzialmente attenuato

[5] http://threadingbuildingblocks.org/

grazie all'adattabilità del numero di particelle utilizzate dal nostro sistema, che porta ad una drastica riduzione delle particelle necessarie per una localizzazione locale. Questo problema non è stato riscontrato con i processori di categoria superiore seppure a più bassa frequenza di lavoro. Confermiamo quindi l'ottima scalabilità degli algoritmi introdotta con le librerie di parallelizzazione Intel® TBB (l'utilizzo di 1 core o 4 core porta ad una diminuzione effettiva del tempo di calcolo di circa 1/4, si faccia riferimento alla Appendice A.5.3).

4.7.2 Il middleware

Nella scelta del middleware abbiamo analizzato svariate soluzioni presenti ed utilizzate in ambito internazionale (Orocos, Aria, Carmen, Orca, Ros). La scelta è caduta sull'ambiente ROS [6] [6] (Robot Operating System) ed è stata dettata dall'ottima disponibilità di documentazione e di software aggiuntivo open-source creato dalla community: driver per sensori, soluzioni di localizzazione e navigazione, presenza di un ambiente di simulazione 3D.
Il core di ROS propone un insieme di funzioni ad alto livello che permettono di astrarre l'hardware sensoriale utilizzato, scambiare messaggi tra processi concorrenti attraverso paradigmi sincroni ed asincroni come services e publisher/subscriber, dividere il carico di esecuzione tra diversi nodi di rete ed una serie di strutture dati ed algoritmi base che permettono di affrontare in modo elegante ed efficiente la creazione di software robotico.

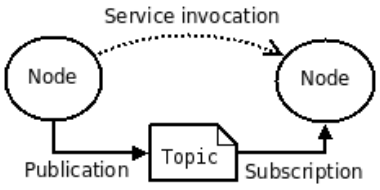


Figura 4.14: *I paradigmi di comunicazione tra i nodi in ROS*

Per capire il funzionamento di ROS possiamo distinguere i seguenti concetti:

[6] http://www.ros.org/

- ROS Master (o *Core*): il nome del servizio principale che effettua le operazioni di routing tra le chiamate ai servizi offerti dai nodi.
- Nodo: un nodo è semplicemente il programma sviluppato dal programmatore. Un nodo si interfaccia il con Master per comunicare con altri nodi.
- Topics: i canali attraverso i quali i nodi comunicano con altri nodi. Un topic ha un tipo di dato che può essere sia uno dei tipi predefiniti di ROS (integer, string, boolean ...) che un tipo creato ad-hoc. Il nodo può sia pubblicare un topic che sottoscriversi ad un topic per riceverne i messaggi.
- Messages: rappresentano i dati che vengono utilizzati quando un nodo si sottoscrive o pubblica in un topic.

Problemi relativi alle code di messaggi

Una caratteristica di ROS è la presenza di code di messaggi agganciate ai topics. La pubblicazione di un messaggio da parte di un nodo consiste nell'introduzione del dato all'interno di una coda (disponibile sia in uscita che in ingresso); una volta eseguita l'operazione di inserimento in coda il nodo master ROS provvederà a consegnare il messaggio al nodo destinatario a seconda delle disponibilità dei suoi thread interni; questo meccanismo consente ad un nodo di poter aggiungere dati in uscita e di non preoccuparsi dell'effettiva consegna ai nodi in ascolto sul topic. Viceversa, le code in ingresso garantiscono ai nodi di non perdere un dato nel caso in cui fossero impegnati in altre operazioni. Questo comportamento, caratteristico dei sistemi operativi, rende necessaria una particolare attenzione in ambito robotico e nelle fasi di dimensionamento delle code. I robot per loro natura sistemi realtime. L'introduzione di code di lettura e scrittura porta con sé l'introduzione di ritardi nella propagazione delle informazioni. Per dimostrare quanto siano rischiose le code, proponiamo un semplice esempio: sia dato un nodo adibito alla lettura e pubblicazione dei dati LIDAR che produce messaggi contenenti le scansioni ad una frequenza di 25 hertz; sia dato un secondo nodo adibito al filtraggio dei dati sensoriali che abbia una coda in grado di contenere 100

messaggi (la coda è di tipo FIFO; a coda piena l'elemento più vecchio viene scartato in caso arrivi un nuovo dato) e supponiamo che il tempo di esecuzione dell'algoritmo del nodo richieda 0.1 secondi. Ne consegue che appena dopo 10 secondi di esecuzione la coda del nodo che effettua il filtraggio verrà saturata ed è in questo momento che si verifica un grave errore concettuale. Supponiamo che l'output del secondo nodo sia l'input mediante il quale un terzo nodo adibito alla fermata di emergenza del veicolo controlli l'impianto di frenata e di accelerazione e che il sensore rilevi una persona accidentalmente sul piano stradale; sebbene la rilevazione richieda solamente 0.04 secondi il tempo necessario affinché il veicolo si fermi con una coda di ricezione satura sul secondo nodo sarà dato da:

0.04 secondi	tempo necessario per il completamento di una scansione, creazione ed inserimento del messaggio nella coda di uscita
0.01 secondo	tempo assunto per eccesso affinché il dato venga preso in consegna da ROS e recapitato
$0.1 \cdot 99 = 9.9$ secondi	affinché il robot completi l'analisi dei 99 messaggi in coda che **non** segnalano la presenza di un ostacolo
0.14 secondi	tempo necessario ad elaborare il messaggio contenente l'ostacolo; viene inoltre creato il messaggio da inviare al nodo di navigazione e messo in coda di uscita
0.01 secondi	affinché ROS prenda in consegna e recapiti il nuovo messaggio
0.04 secondi	il tempo minimo assunto affinché il nodo di navigazione valuti il messaggio di emergenza ed inizi a frenare
10.14 secondi	tempo totale prima di iniziare a frenare

Dalle prove effettuate ad una velocità di soli 5 km/h questo intervallo di tempo non stato è neppure sufficiente ad attivare l'impianto di frenata del Cart. Data la natura quadratica della legge di accelerazione. si intuisce come un simile tempo di reazione non possa essere accettato, tantomeno su un veicolo autonomo in ambito urbano. Un effetto di questo tipo è stato

riscontrato in molti dei nodi ROS che abbiamo inizialmente utilizzato sul cart ed anche lo stesso AMCL di Brian Gerkey [23] ne è afflitto: in tal caso il problema si manifestava attraverso la localizzazione effettuata con svariati secondi di ritardo ed un effetto molla sull'effettiva posizione; questo grave problema ci ha portato ad intervenire manualmente, in fase di navigazione autonoma, per evitare collisioni del veicolo con l'ambiente statico, in quanto il robot non era correttamente localizzato.
La soluzione a questa spinosa problematicità consiste nella drastica riduzione della lunghezza delle code arrivando fino a lunghezza unitaria. Nei test effettuati in U5 abbiamo riscontrato che la versione AMCL [7] presente sul repository ROS dopo la modifica per rendere unitaria la lunghezza delle code non è afflitta dal problema del ritardo.

4.7.3 Intraprocess comunication

I nodi di ROS rappresentano dei processi di sistema. Nell'ambito della comunicazione tra diversi nodi, ROS prevede la serializzazione/deserializzazione delle strutture dati C++ al fine di comporre un messaggio. Nel caso di messaggi contenenti grandi quantità di dati, come può essere la scansione completa di un LIDAR (contenente non solo misure di distanza, ma anche di intensità dei valori letti), accade che la procedura di invio messaggi introduca un ritardo nella comunicazione. Una soluzione a questo problema è rappresentato dalle tecniche di *intraprocess publishing* fornite dalla versione ROS *Cturtle* in avanti e consiste nella definizione di un nuovo tipo di nodo: il *nodelet*. I nodelet sono progettati per permettere l'esecuzione di diversi algoritmi su una stesso calcolatore sottoforma di thread, senza incorrere nei costi di copia dei dati quando i messaggi sono passati all'interno dello stesso processo; le ottimizzazioni riguardano i passaggi di puntatori al posto dei dati veri e propri. Al fine di poter gestire i nodelet ROS fornisce un meccanismo di plugin dinamici (libreria pluginlib [8]); i nodelet, visti come plugin, vengono quindi *caricati* su un *nodelet master* ed eseguiti come thread al suo interno.

[7] http://www.ros.org/wiki/amcl
[8] http://www.ros.org/wiki/pluginlib

4.8 I sistemi di simulazione

Nel lavoro di tesi si è reso necessario effettuare delle simulazioni riguardanti principalmente il modello di moto odometrico a 6DoF. Sono stati generati due ambienti di simulazione. Il primo riguarda esclusivamente il modello odometrico, mentre il secondo ricrea una situazione completa per il test di localizzazione nel quale sono state testate tutte le componenti software (gli algoritmi MCL, il raycasting 3D, l'algoritmo di clustering). Grazie a questi ambienti è stato possibile individuare gli errori descritti nel Paragrafo 3.3.3.

Il primo simulatore è stato creato per verificare l'esattezza del modello odometrico; l'ambiente di simulazione permette la visualizzazione degli effetti dei singoli parametri sull'effettivo comportamento della simulazione. L'ambiente permette di impostare tutti i parametri del modello, ovvero:

- 6 parametri relativi all'odometro esteso, ovvero i delta odometrici
- 10 parametri relativi agli *alpha* che pesano i 6 parametri del modello
- 6 parametri relativi ai *sigma* minimi in riferimento all'errore aggiunto sui 6 parametri del modello
- il numero di steps da simulare
- il numero di ipotesi da calcolare attraverso la procedura di sampling

Il simulatore si è rivelato molto utile per capire l'influenza delle singole componenti sul comportamento finale del modello ed è risultato uno strumento indispensabile per il corretto dimensionamento e la successiva verifica delle *sigma-minime o massime* da utilizzare. Nella Figura 4.15 e nella Figura 4.16 è visibile il simulatore in azione con la sua schermata di modifica dei parametri.

I test effettuati con il secondo ambiente di simulazione hanno dimostrato che nel caso bidimensionale l'algoritmo a 6DoF si comporta molto bene, al pari con l'algoritmo 3DoF presente in ROS. Era infatti richiesto che si avesse almeno la performance del sistema precedente, nelle condizioni tipiche

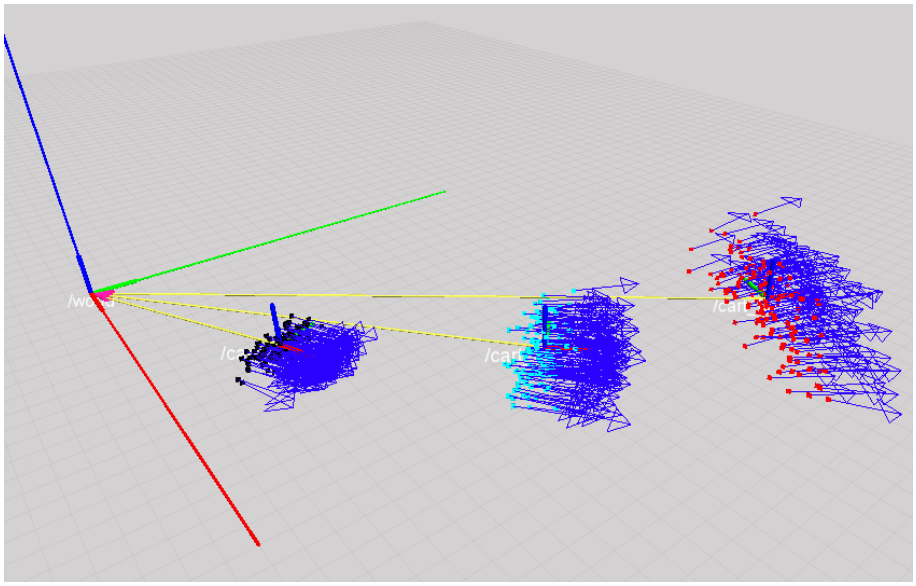


Figura 4.15: *Un esempio di simulazione del modello 6DoF nella quale sono state simulate 3 evoluzioni (il delta applicato ad ogni evoluzione è costante).*

del sistema precedente. Inoltre, disattivando i laser a quattro piani, abbiamo verificato l'atteso incremento di incertezza su tutto l'asse di quota (vedi Figura 4.17).

4.9 Ambiente reale di test

Dopo il lungo testing in laboratorio, siamo passati ad una verifica sul campo del software creato. L'ambiente nel quale abbiamo effettuato gli esperimenti è la zona del parcheggio dell'edificio U5. I test si sono svolti nel seguente ordine: inizialmente abbiamo verificato la corretta localizzazione nell'ambito di movimento quasi esclusivamente bidimensionale, effettuando dei giri nel parcheggio sotterraneo ed ottenendo risultati in linea quello della con la Figura 4.17. Una volta effettuati questi test, siamo passati ad effettuare, sempre con successo, manovre che comprendessero l'uscita verso il parcheggio esterno attraverso la rampa che collega i due parcheggi. Nonostante la disponibilità dei dati di rollio ed il beccheggio del Cart ottenuti abilitando il sensore inerziale X-sens (vedi Appendice C.3) abbiamo verificato che le sole soglie minime e massime del modello odometrico proposto nel Capitolo 3, in-

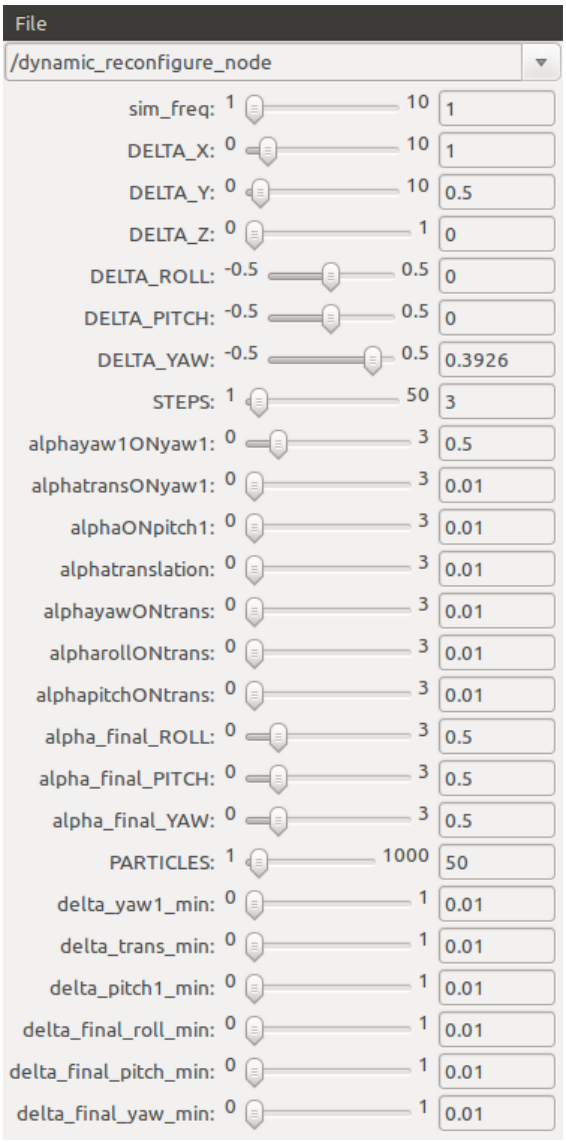


Figura 4.16: *I parametri impostabili dal simulatore del modello 6DoF. I parametri che riguardano i delta cartesiani sono espressi in metri, mentre le misure angolari sono espresse in radianti.*

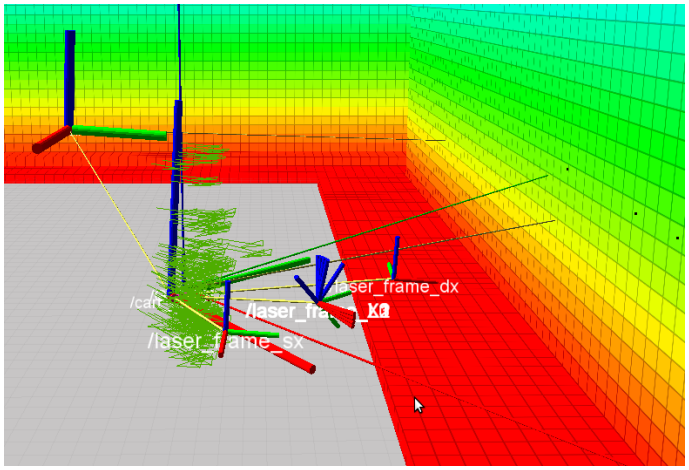


Figura 4.17: *Nella figura il risultato di una localizzazione con soli laser montati parallelamente al piano di moto del robot. L'incertezza che ne deriva è rappresentata dallo spargimento di particelle sull'asse z.*

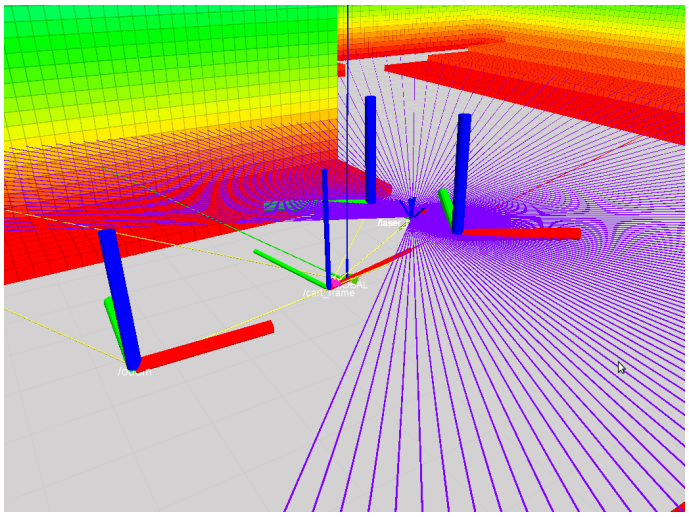


Figura 4.18: *In questa figura il risultato della procedura di raycasting effettuata del simulatore.*

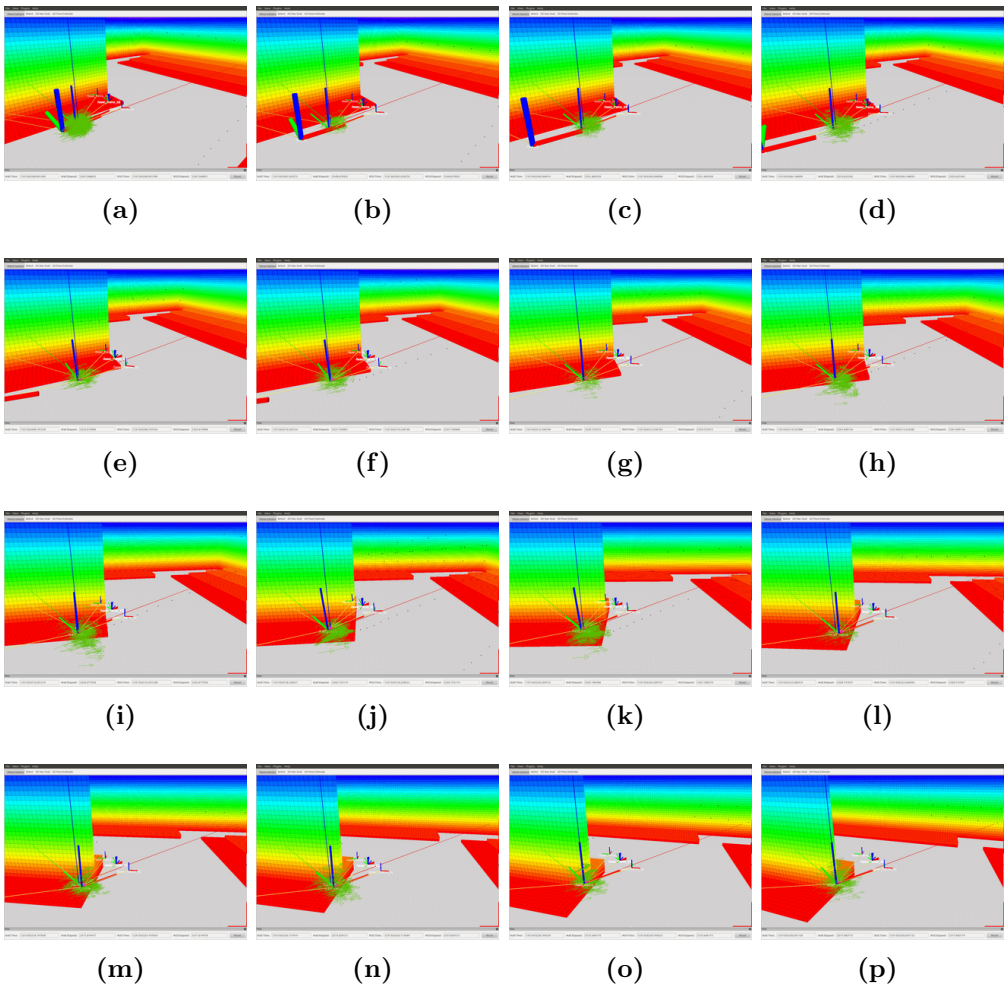
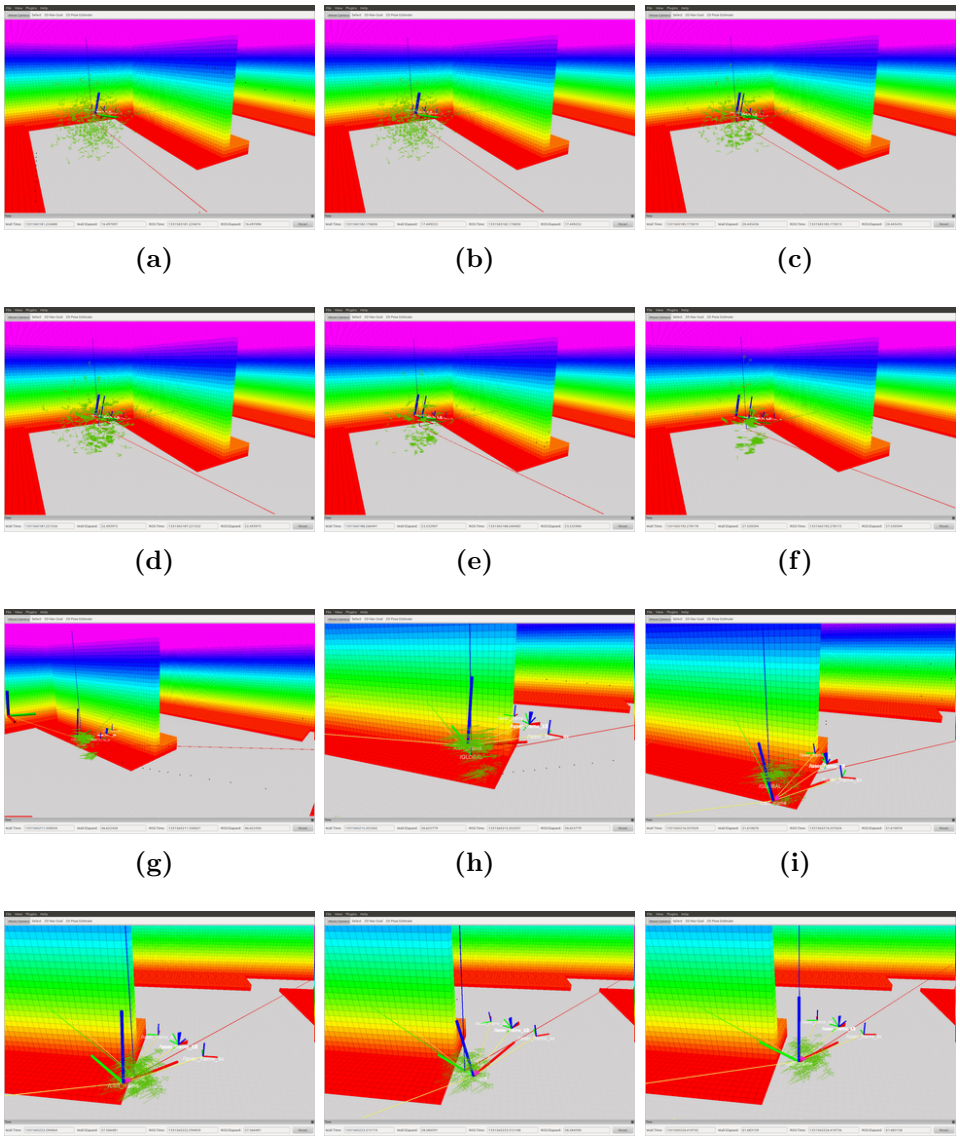


Figura 4.19: *Nella figura un esempio di simulazione all'interno dell'ambiente utilizzato per il test del lavoro di tesi. L'asse più grande centrale (in grassetto) nella (a) indica l'origine del movimento del robot. Man mano che si vengono registrati movimenti, il sistema di riferimento del robot viene spostato, come si può vedere in (c). Il numero di particelle di questo particolare esempio è stato impostato a 100. Il resampling è stato effettuato ogni 5 fasi di integrazione del movimento del robot. Si può notare come l'azione di resampling concentri le particelle attorno alla posizione corretta (indicata dall'asse sottile e sempre visibile) in (e) e (l).*



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

Figura 4.20: *Una seconda simulazione con una incertezza iniziale maggiore. Si può vedere come in pochi passi il sistema riesca correttamente a localizzarsi.*

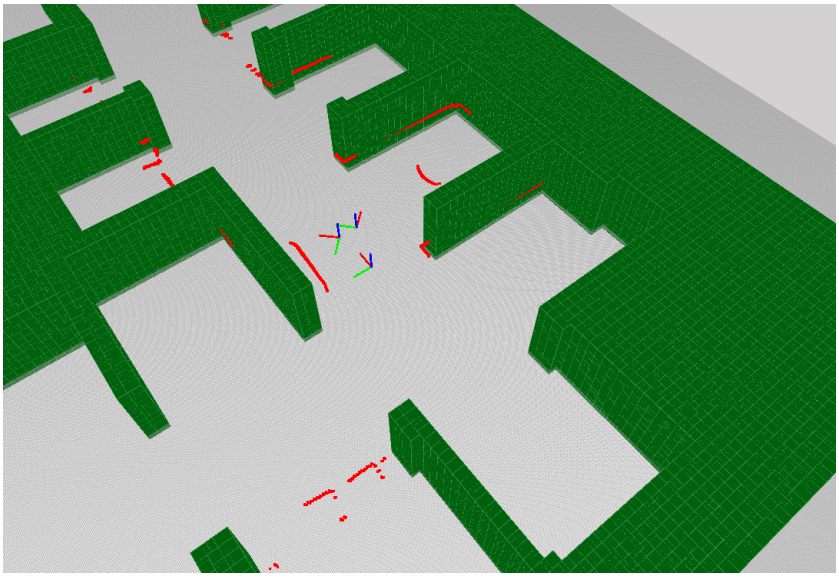


Figura 4.21: *Nella figura un istante di localizzazione del robot all'interno della mappa estrusa del parcheggio U5. I punti rossi identificano le misure degli scanner (le misure non in corrispondenza dei muri corrispondono a macchine ed oggetti non presenti nella mappa).*

sieme all'utilizzo contemporaneo di tutti i LIDAR a disposizione, hanno permesso una corretta localizzazione. Ricordiamo che l'utilizzo contemporaneo delle due tipologie di LIDAR si rivela utile in quanto effettuano misurazioni su piani di scansione diversi (si veda l'Appendice C per una descrizione del posizionamento dei LIDAR sul Cart).

4.9.1 Risultati

I risultati sono stati ottimi. L'algoritmo ha effettuato una corretta localizzazione in tutte le fasi del percorso effettuato; le fasi critiche, ovvero l'avvicinamento alla rampa e la successiva salita hanno evidenziato la correttezza del modello di moto proposto e la rampa è stata correttamente individuata come oggetto appartenente al piano stradale. Anche nel caso degli ambienti reali, l'utilizzo contemporaneo dei Sick LMS-111 e del Sick LDMRS-4001 si è rivelato utile per una corretta localizzazione grazie ai 6 piani di scansione differenti forniti.

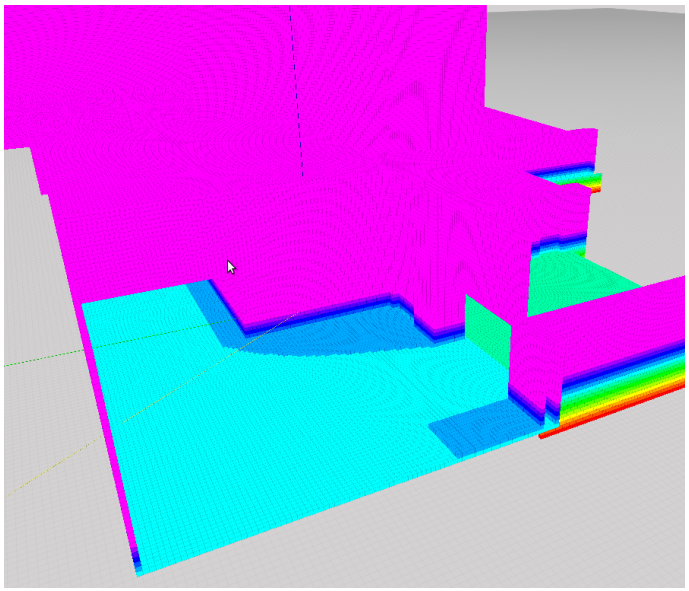


Figura 4.22: *L'uscita del garage sotterraneo dell'edificio U5.*

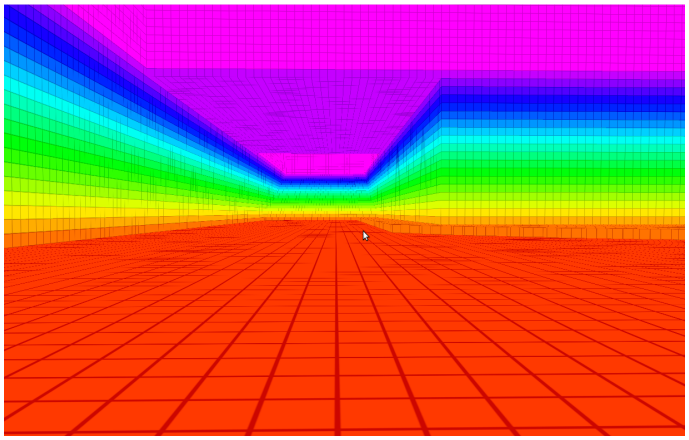


Figura 4.23: *In lontananza la rampa di uscita dai garage sotterranei dell'edificio U5. Il cambio dei colori verso il centro dell'immagine rappresenta la salita e non un muro.*

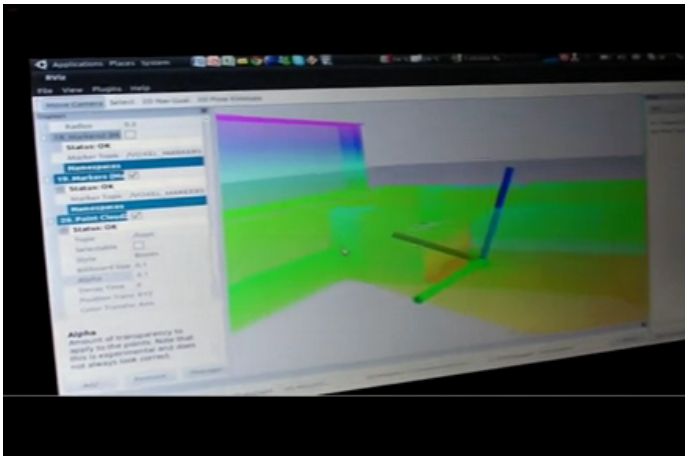


Figura 4.24: *Un istante del video registrato durante i test di uscita dal garage. Nell'immagine è visibile il frame di riferimento del Cart inclinato sulla rampa di salita. I muri sono visualizzati in trasparenza affinché fosse possibile osservare il frame e le particelle, altrimenti chiusi all'interno della struttura dell'edificio.*

Capitolo 5

Conclusioni

In questo progetto abbiamo realizzato un sistema di autolocalizzazione per robot mobili operante in un ambiente 3D ovvero un sistema per la determinazione dei 6 parametri che caratterizzano il posizionamento di un corpo rigido nello spazio. Questo ha richiesto la definizione di un modello di moto probabilistico per la predizione del movimento. A sua volta ciò ha richiesto lo sviluppo di un sistema di simulazione nel quale effettuare i test. Il lavoro è parte integrante del progetto *Urban Shuttles Autonomously Driven* (USAD) sviluppato dal laboratorio IRA che si pone l'obiettivo di permettere ad un veicolo autonomo di effettuare la navigazione in ambito outdoor urbano, ad esempio all'interno del campus universitario.

Il lavoro svolto è stato efficacemente impiegato in diverse situazioni pubbliche come la fiera *Electrical Intelligent Vehicles 2010* e la demo del programma televisivo di Rai3 *Buongiorno Regione* effettuata nel piazzale tra gli edifici U1 ed U2 (Piazza della Scienza). Dal punto di visto della ricerca scientifica il lavoro è di massima attualità e, al meglio delle nostre conoscenze, è quanto di più avanzato sia stato sviluppato nel suo genere, tanto per correttezza scientifica che per efficacia. E' in fase di stesura un articolo sul lavoro svolto da sottomettere alla principale conferenza del settore.

Appendice A

Codice

Questa appendice mostra l'architettura implementativa degli pseudocodici visti nella tesi e le tecniche di parallelizzazione utilizzate.

A.1 Clustering

```
1  double soglia_euclidea=0.50f;
2  double soglia_angolare=0.05f;
3
4  for (int num=0;num<set->sample_count;num++)
5  {
6    scor=set->samples+num;
7    scor->incluster=-1;
8  }
9
10 for (int num=0;num<set->sample_count;num++)
11 {
12   scor=set->samples+num;
13
14   if (scor->incluster==-1)
15     scor->incluster=indice_cluster++;
16
17   for (int num2=0;num2<set->sample_count;num2++)
18   {
19     scor2=set->samples+num2;
20
21     if (num==num2)
```

```
22        continue;
23
24      if (scor2->incluster!=-1)
25      continue;
26
27      distanza_euclidea=sqrt( (scor->pose.v[0]-scor2->pose.v[0])*(
            scor->pose.v[0]-scor2->pose.v[0])    +   (scor->pose.v[1]-
            scor2->pose.v[1])*(scor->pose.v[1]-scor2->pose.v[1]) + (
            scor->pose.v[3]-scor2->pose.v[3])*(scor->pose.v[3]-scor2
            ->pose.v[3]));
28
29      a=tf::createQuaternionFromRPY(scor->pose.v[4],scor->pose.v
            [5],scor->pose.v[2]);
30      b=tf::createQuaternionFromRPY(scor2->pose.v[4],scor2->pose.v
            [5],scor2->pose.v[2]);
31
32      a.normalize();b.normalize();
33
34      distanza_angolare=a.angleShortestPath(b);
35
36
37      if (distanza_euclidea< soglia_euclidea)
38        if(distanza_angolare<soglia_angolare)
39    scor2->incluster=scor->incluster;
40    }
41  }
```

Tabella A.1: *L'algoritmo di clustering utilizzato in AMCL-6DOF*

A.2 Bresenham Implementation

```
1  x0 = MAP_GXWX(map,ox); //World To Map
2  y0 = MAP_GYWY(map,oy); //World To Map
3  z0 = MAP_GZWZ(map,oz); //World To Map
4
5  tf::Vector3 axis(range_max,0.0,0.0);
6  tf::Vector3 rotationAxis;
7  tf::Transform translation, particle_pose, endpoint, rotation;
8
```

```
9  particle_pose.setOrigin(tf::Vector3(ox,oy,oz));
10 particle_pose.setRotation(tf::createQuaternionFromRPY(oroll,
       opitch, oyaw));
11
12 translation.setOrigin(axis);
13 translation.setRotation(tf::createQuaternionFromYaw(0.0));
14
15 rotation.setOrigin(tf::Vector3(0.0,0.0,0.0));
16 btQuaternion q1;
17 q1.setRotation(tf::Vector3(0.0,0.0,1.0).normalize(),bearing);
18 rotation.setRotation(q1);
19
20 endpoint= particle_pose * rotation * translation;
21
22 x1 = MAP_GXWX(map,endpoint.getOrigin().getX()); //World To Map
23 y1 = MAP_GYWY(map,endpoint.getOrigin().getY()); //World To Map
24 z1 = MAP_GZWZ(map,endpoint.getOrigin().getZ()); //World To Map
25
26
27 dx = x1 - x0;
28 dy = y1 - y0;
29 dz = z1 - z0;
30
31 if (dx < 0)
32  x_inc = -1;
33 else
34  x_inc =  1;
35
36 if (dy < 0)  y_inc = -1; else  y_inc =  1;
37 if (dz < 0)  z_inc = -1; else  z_inc =  1;
38
39 adx = abs(dx);  ady = abs(dy);  adz = abs(dz);
40 dx2 = adx*2;  dy2 = ady*2;  dz2 = adz*2;
41
42 if ((adx>= ady) && (adx>= adz)) {
43 err_1 = dy2 - adx;
44 err_2 = dz2 - adx;
45 x=x0; y=y0; z=z0;
46
47 while (x0!=x1)
48 {
```

```
49      if (err_1 > 0) {
50          y0+= y_inc;
51          err_1 -= dx2;
52      }
53
54      if (err_2 > 0) {
55          z0+= z_inc;
56          err_2 -= dx2;
57      }
58
59      err_1 += dy2;
60      err_2 += dz2;
61      x0+= x_inc;
62
63      if(   !MAP_VALID(map,x0,y0,z0) ||
64    map->cells[MAP_INDEX(map,x0,y0,z0)].occ_state >
          value_threshold)
65      {
66
67        return sqrt((x-x0)*(x-x0) + (y-y0)*(y-y0) + (z-z0)*(z-z0))
            * map->scale;
68      }
69  }
70  }
71
72  if ((ady> adx) && (ady>= adz)) {
73  err_1 = dx2 - ady;
74  err_2 = dz2 - ady;
75  x=x0; y=y0; z=z0;
76
77  while(y0!=y1)
78  {
79
80      if (err_1 > 0) {
81          x0+= x_inc;
82          err_1 -= dy2;
83      }
84
85      if (err_2 > 0) {
86          z0+= z_inc;
87          err_2 -= dy2;
```

```
88      }
89
90      err_1 += dx2;
91      err_2 += dz2;
92      y0+= y_inc;
93
94      if(   !MAP_VALID(map,x0,y0,z0)
95     || map->cells[MAP_INDEX(map,x0,y0,z0)].occ_state >
           value_threshold)
96      {
97        return sqrt((x-x0)*(x-x0) + (y-y0)*(y-y0) + (z-z0)*(z-z0))
              * map->scale;
98      }
99  }
100 }
101
102 if ((adz> adx) && (adz> ady)) {
103 err_1 = dy2 - adz;
104 err_2 = dx2 - adz;
105 x=x0; y=y0; z=z0;
106
107 while(x0!=z1)
108 {
109     if (err_1 > 0) {
110         y0+= y_inc;
111         err_1 -= dz2;
112     }
113
114     if (err_2 > 0) {
115         x0+= x_inc;
116         err_2 -= dz2;
117     }
118
119     err_1 += dy2;
120     err_2 += dx2;
121
122     z0+= z_inc;
123
124     if(   !MAP_VALID(map,x0,y0,z0)
125    || map->cells[MAP_INDEX(map,x0,y0,z0)].occ_state >
           value_threshold)
```

```
126     {
127       return sqrt((x-x0)*(x-x0) + (y-y0)*(y-y0) + (z-z0)*(z-z0))
              * map->scale;
128     }
129 }
130 }
131
132 return range_max;
```

Tabella A.2: *Raycasting con l'algoritmo di Bresenham 3d*

A.3 Media di Quaternioni

```
1  btTransform t1,t2,t3,ttemp;
2  btVector3 axis(1,0,0);
3  btQuaternion q1,q2,qs,q3,q_random;
4  unsigned char number_quats = 10;
5
6  // base rotation in RPY
7  double roll = 0.0f;
8  double pitch = 0.0f;
9  double yaw = 0.0f;
10
11 t1.setOrigin(axis);
12 t1.setRotation(tf::createQuaternionFromRPY(roll,pitch,yaw));
13 br.sendTransform(tf::StampedTransform(t1, ros::Time::now(), "/
       world","/base"));
14
15 qs=btQuaternion::getIdentity();
16
17 for (unsigned char i = 0; i<number_quats; i++)
18 {
19   geometry_msgs::Point p; //for visualization
20   float sigma = 0.08;
21   roll += pf_ran_gaussian(sigma).v[0];
22   pitch += pf_ran_gaussian(sigma).v[0];
23   yaw += pf_ran_gaussian(sigma).v[0];
24
25   q_random = tf::createQuaternionFromRPY(roll, pitch, yaw);
```

```
26    q_random.normalize();
27
28    ttemp.setIdentity();
29    ttemp.setOrigin(tf::Vector3(0,0,0));
30    ttemp.setRotation(q_random);
31
32    ttemp=ttemp*t1;
33    p.x=  ttemp.getOrigin().x(); p.y= ttemp.getOrigin().y(); p.z=
          ttemp.getOrigin().z();
34    sphere_list.points.push_back(p);
35
36    br.sendTransform(tf::StampedTransform(ttemp*t1, ros::Time::now
          (), "/world","/random"));
37    ros::Duration(0.1).sleep();
38    qs+=q_random; // somma di quaterioni
39
40    vis_pub.publish( sphere_list);
41  }
42
43  qs.normalize();
44
45  t2.setIdentity();
46  t2.setOrigin(tf::Vector3(0,0,0));
47  t2.setRotation(qs);
```

Tabella A.3: *In questo esempio viene calcolata la media di 10 orientamenti espressi sottoforma di quaternioni; il codice fa parte del nodo QuaternionMean*

A.4 Simulatore 6DoF

```
1
2    // store initial pose of the particle
3    Transform temp_particle;
4    temp_particle.setOrigin(particles[i].getOrigin());
5    temp_particle.setRotation(particles[i].getRotation());
6
7    ...
8    ...
9
```

```
10  vector <Transform> motions;
11  Transform motion;
12
13  // first motion -  Odometry Model 6DOF
14  motion.setOrigin(Vector3(0.0f,0.0f,0.0f));
15  motion.setRotation(quaternion_for_yaw1);
16  motions.push_back(motion);
17
18  // second motion - Odometry Model 6DOF
19  motion.setOrigin(Vector3(0.0f,0.0f,0.0f));
20  motion.setRotation(quaternion_for_pitch1);
21  motions.push_back(motion);
22
23  // third motion - Odometry Model 6DOF
24  motion.setOrigin(Vector3(FINAL_TRANSLATION,0.0f,0.0f));
25  motion.setRotation(createIdentityQuaternion());
26  motions.push_back(motion);
27
28  //compose the motions & apply the composed motion to
29  //the particle, this part is for the POSITION
30  particles[i]=particles[i]*motions[0]*motions[1]*motions[2];
31
32  //create the particle! First the POSITION
33  p.x = pose.position.x = particles[i].getOrigin().getX();
34  p.y = pose.position.y = particles[i].getOrigin().getY();
35  p.z = pose.position.z = particles[i].getOrigin().getZ();
36
37  //and then apply the ORIENTATION
38  btQuaternion temp;
39  temp=createQuaternionFromRPY(Final_Roll,Final_Pitch,Final_Yaw)
        ;
40  temp_particle.setRotation(temp_particle.getRotation()*temp);
41  particles[i].setRotation(temp_particle.getRotation());
```

Tabella A.4: *Nell'esempio troviamo una parte del codice del simulatore del modello odometrico 6DoF; la position della particella viene calcolata concatenando i primi tre atti di moto elementari mentre la parte rotation è applicata in un secondo momento.*

A.5 Intel Threading Building Blocks

Gli algoritmi della libreria TBB [1] permettono di creare in modo semplice codice concorrente. Gli *algoritmi* rappresentano un insieme di pattern concorrenti che si interfacciano con uno scheduler che permette la generazione a runtime di threads; il numero di threads viene adattato dinamicamente a seconda delle risorse disponibili permettendo una ottima scalabilità a seconda delle richieste software e delle disponibilità hardware. Tra gli svariati pattern disponibili sono stati utilizzati i design pattern *parallel_for* e *parallel_reduce* dei quali segue una breve descrizione con codice.

A.5.1 L'algoritmo Parallel For

L'algoritmo Parallel For rappresenta l'esecuzione parallela del semplice ciclo For dove le operazioni svolte in parallelo non coinvolgono risultati di operazioni concorrenti. E' il caso della applicazione dell'Odometry Model al set di particelle che individuano le ipotesi di localizzazione del cart. Ciascuna particella può essere trattata in modo separato.
TBB: $parallel\_for(first, last, step, f)$
C++: $for(i = first; i < last; i+ = step)f(i);$

```
1  #include "tbb/tbb.h"
2  #include <stdio.h>
3  #include <iostream>
4
5  using namespace tbb;
6
7  class ApplyFoo
8  {
9  private:
10   int temp;
11   int *const ptemp;
12 public:
13   void operator() (const blocked_range<size_t>&r ) const
14   {
15     printf("Intel Threading Building Blocks! temp=%d\n",temp);
```

[1] http://threadingbuildingblocks.org/

```
16      for (size_t i=r.begin(); i!=r.end(); ++i)
17        sleep(*ptemp);
18    }
19
20    ApplyFoo( int tempo, int *ptempo) : temp(tempo), ptemp(ptempo)
21    {
22
23    }
24
25  };
26
27  int main(){
28    tick_count serial_t0 = tick_count::now();
29
30    int temp=1;
31    int *ptemp=&temp;
32    size_t n=5;
33
34    for(int i=0;i<n;i++)
35    {
36      sleep(temp);
37    }
38    tick_count serial_t1 = tick_count::now();
39    printf("%f\n", (serial_t1 - serial_t0).seconds() );
40    serial_t0 = tick_count::now();
41
42    parallel_for(blocked_range<size_t>(0,n), ApplyFoo(temp,ptemp))
          ;
43
44    serial_t1 = tick_count::now();
45    printf("%f\n", (serial_t1 - serial_t0).seconds() );
46
47    printf("End\n");
48  }
```

A.5.2 L'algoritmo Parallel Reduce

L'algoritmo Parallel Reduce viene utilizzato quando è necessario ottenere un unico risultato da una serie di calcoli svolti in parallelo. E' il caso del

calcolo del peso totale associato alle particelle, dove ogni particella viene pesata in base ai dati sensoriali in modalità concorrente. La variabile di peso è concorrente ai vari thread che eseguono il calcolo del singolo peso e viene aggiornata in automatico da questo algoritmo.

```
1  #include "tbb/tbb.h"
2  #include <stdio.h>
3  #include <iostream>
4  #include <stdio.h>
5  #include <math.h>
6  #include <sys/time.h>
7  #include <stdio.h>
8  #include <unistd.h>
9
10 using namespace std;
11 using namespace tbb;
12
13 class SumFoo {
14 unsigned int* internal_vector;
15 public:
16 unsigned long result_sum;
17   void operator()( const blocked_range<size_t>& r )
18   {
19     unsigned int*a = internal_vector;
20     unsigned long sum = result_sum;
21     size_t end = r.end();
22
23     for( size_t i=r.begin(); i!=end; ++i )
24     {
25       sum += a[i];
26     }
27
28     result_sum = sum;
29   }
30
31   SumFoo( SumFoo& x, split ) : internal_vector(x.internal_vector
       ), result_sum(0)
32   {
33
34   }
35
```

```
36    void join( const SumFoo& join_values )
37    {
38      result_sum+=join_values.result_sum;
39    }
40
41    SumFoo(unsigned int a[] ) : internal_vector(a), result_sum(0)
42    {
43
44    }
45  };
46
47
48  int main(int argc, char** argv){
49
50    struct timeval start, end;
51    long mtime, seconds, useconds;
52
53    unsigned int temp=119999339;
54    unsigned int local=0.0;
55
56    unsigned int *a=new unsigned int[temp];
57    for (int i=0;i<temp;i++)
58      a[i]=i;
59
60    gettimeofday(&start, NULL);
61    for (unsigned int i=0;i<temp;i++)
62      local+=a[i];
63    gettimeofday(&end, NULL);
64
65    seconds  = end.tv_sec  - start.tv_sec;
66    useconds = end.tv_usec - start.tv_usec;
67    mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
68    printf("Elapsed time: %ld milliseconds\t\t%u\tserial execution
        \n\n", mtime,local);
69
70
71    SumFoo somma(a);
72    gettimeofday(&start, NULL);
73    parallel_reduce( blocked_range<size_t>(0,temp),somma );
74    gettimeofday(&end, NULL);
75    seconds  = end.tv_sec  - start.tv_sec;
```

```
76    useconds = end.tv_usec - start.tv_usec;
77    mtime = ((seconds) * 1000 + useconds/1000.0) + 0.5;
78    printf("Elapsed time: %ld milliseconds\t\t%lu\tparallel
          execution\n", mtime,somma.result_sum);
79
80    printf("End\n");
81  }
```

A.5.3 Risultati Sperimentali

Al fine di verificare il guadagno di prestazioni ottenuto con le procedure di parallelizzazione TBB forniamo i seguenti risultati, effettuati su un calcolatore Intel® Core™ i7-740QM, 6mb cache, 8gb ram (quad core). Si può osservare un incremento di prestazioni nell'esecuzione dell'algoritmo SENSOR MODEL (Beam Model, Bresenham) vicino al numero di core disponibili. Negli algoritmi a più basso carico computazionale, sebbene l'incremento sia presente, non raggiunge gli stessi valori ottenuti con il SENSOR MODEL.

```
TEST BEAM MODEL WITH BRESENHAM WITH 1000
PARTICLES AND 400 BEAM PER SCAN

SERIAL EXECUTION ON INTERNAL FOR (bresenham)
SENSOR_MODEL Elapsed time: 753 milliseconds
SENSOR_MODEL Elapsed time: 746 milliseconds
SENSOR_MODEL Elapsed time: 736 milliseconds
SENSOR_MODEL Elapsed time: 744 milliseconds
SENSOR_MODEL Elapsed time: 732 milliseconds
SENSOR_MODEL Elapsed time: 749 milliseconds
SENSOR_MODEL Elapsed time: 736 milliseconds
SENSOR_MODEL Elapsed time: 747 milliseconds
SENSOR_MODEL Elapsed time: 746 milliseconds
SENSOR_MODEL Elapsed time: 759 milliseconds
SENSOR_MODEL Elapsed time: 826 milliseconds
SENSOR_MODEL Elapsed time: 789 milliseconds
SENSOR_MODEL Elapsed time: 741 milliseconds
SENSOR_MODEL Elapsed time: 747 milliseconds
```

```
SENSOR_MODEL Elapsed time: 756 milliseconds
SENSOR_MODEL Elapsed time: 763 milliseconds
SENSOR_MODEL Elapsed time: 758 milliseconds
average: 752 ms


TBB ENABLED ON INTERNAL FOR (bresenham),
SENSOR_MODEL Elapsed time: 211 milliseconds
SENSOR_MODEL Elapsed time: 217 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 222 milliseconds
SENSOR_MODEL Elapsed time: 204 milliseconds
SENSOR_MODEL Elapsed time: 212 milliseconds
SENSOR_MODEL Elapsed time: 207 milliseconds
SENSOR_MODEL Elapsed time: 211 milliseconds
SENSOR_MODEL Elapsed time: 208 milliseconds
SENSOR_MODEL Elapsed time: 211 milliseconds
SENSOR_MODEL Elapsed time: 210 milliseconds
SENSOR_MODEL Elapsed time: 208 milliseconds
SENSOR_MODEL Elapsed time: 197 milliseconds
SENSOR_MODEL Elapsed time: 201 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 201 milliseconds
SENSOR_MODEL Elapsed time: 201 milliseconds
average: 208ms


1000 particles + TBB + AFFINITY PARTITIONER
SENSOR_MODEL Elapsed time: 195 milliseconds
SENSOR_MODEL Elapsed time: 198 milliseconds
SENSOR_MODEL Elapsed time: 203 milliseconds
SENSOR_MODEL Elapsed time: 214 milliseconds
SENSOR_MODEL Elapsed time: 208 milliseconds
SENSOR_MODEL Elapsed time: 209 milliseconds
SENSOR_MODEL Elapsed time: 213 milliseconds
SENSOR_MODEL Elapsed time: 194 milliseconds
```

```
SENSOR_MODEL Elapsed time: 219 milliseconds
SENSOR_MODEL Elapsed time: 212 milliseconds
SENSOR_MODEL Elapsed time: 232 milliseconds
SENSOR_MODEL Elapsed time: 207 milliseconds
SENSOR_MODEL Elapsed time: 205 milliseconds
SENSOR_MODEL Elapsed time: 236 milliseconds
SENSOR_MODEL Elapsed time: 221 milliseconds
SENSOR_MODEL Elapsed time: 218 milliseconds
SENSOR_MODEL Elapsed time: 202 milliseconds
average: 208ms

WITHOUT AFFINITY PARTITIONER
SENSOR_MODEL Elapsed time: 207 milliseconds
SENSOR_MODEL Elapsed time: 208 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 202 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 207 milliseconds
SENSOR_MODEL Elapsed time: 202 milliseconds
SENSOR_MODEL Elapsed time: 205 milliseconds
SENSOR_MODEL Elapsed time: 201 milliseconds
SENSOR_MODEL Elapsed time: 213 milliseconds
SENSOR_MODEL Elapsed time: 207 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 205 milliseconds
SENSOR_MODEL Elapsed time: 202 milliseconds
SENSOR_MODEL Elapsed time: 206 milliseconds
SENSOR_MODEL Elapsed time: 211 milliseconds
average: 205ms

SAME TEST WITH ODOMETRY MODEL:

NO PARALLELISM
Elapsed time: 47 milliseconds
Elapsed time: 37 milliseconds
```

```
Elapsed time: 40 milliseconds
Elapsed time: 45 milliseconds
Elapsed time: 41 milliseconds
Elapsed time: 44 milliseconds
Elapsed time: 43 milliseconds
Elapsed time: 50 milliseconds
Elapsed time: 49 milliseconds
Elapsed time: 46 milliseconds
Elapsed time: 39 milliseconds
Elapsed time: 39 milliseconds
Elapsed time: 46 milliseconds
Elapsed time: 42 milliseconds
Elapsed time: 38 milliseconds
Elapsed time: 42 milliseconds
Elapsed time: 50 milliseconds
Elapsed time: 42 milliseconds
Elapsed time: 27 milliseconds
Elapsed time: 46 milliseconds
average: 42.65

TBB
Elapsed time: 40 milliseconds
Elapsed time: 21 milliseconds
Elapsed time: 37 milliseconds
Elapsed time: 34 milliseconds
Elapsed time: 31 milliseconds
Elapsed time: 34 milliseconds
Elapsed time: 31 milliseconds
Elapsed time: 28 milliseconds
Elapsed time: 30 milliseconds
Elapsed time: 33 milliseconds
Elapsed time: 30 milliseconds
Elapsed time: 23 milliseconds
Elapsed time: 33 milliseconds
Elapsed time: 39 milliseconds
average: 31.71
```

Appendice B

Nodi ROS

B.1 Nodi creati

- Package: simulation
  Nome Nodo: voxel
  Il nodo implementa un simulatore odometrico ed un simulatore di LIDAR, data una mappa 3D generata con BinVox e salvata in forma di file di testo. Insieme al nodo amcl_node, ovvero il vero e proprio sistema di localizzazione, permette di simulare il comportamento dell'intero sistema.

- Package: simulation
  Nome Nodo: quaternionMean.
  Il nodo crea un set di orientamenti ed effettua la media utilizzando la procedura di SLERP fornita dalla libreria BulletPhysics.

- Package: simulation
  Nome Nodo: odometry6DOF
  Implementa il primo simulatore visto nel Capitolo 4.5, utilizzato per testare il modello odometrico ed il comportamento al variare dei parametri.

- Package: CART
  Nome Nodo: crunchlaser
  Lo scopo di questo nodo è quello di analizzare le scansioni provenienti

dai laser ed eliminare i dati che non si trovano ad una certa soglia dagli ostacoli sulla mappa. Richiede che sia già avvenuta una corretta localizzazione. Sviluppato sottoforma di nodelet, l'obiettivo di questo nodo è quello di consentire al localizzazione in ambienti dove la presenza di elementi dinamici è molto alta. Abbiamo utilizzato questo nodo alla fiera EIV2010, nella quale dovevamo effettuare una navigazione a bassa velocità durante gli affollati orari di apertura.

- Package: CART
  Nome Nodo: CART
  Questo nodo ha il compito di leggere le informazioni dell'odometro esteso (odometria ruote + sensore inerziale).

- Package: CART
  Nome nodo: LMS100.
  Il nodo rappresenta il driver per la lettura dei LIDAR Sick LMS111. Rappresentano l'evoluzione nodelet dei driver scritti da Paolo Surricchio e vengono utilizzati insieme al nodo crunchlaser.

- Package: CART
  Nome nodo: Sick4000
  Sviluppato sia in formato nodo che in formato nodelet, consiste nell'implementazione del driver per la lettura del LIDAR Sick LDMRS4001.

- Package: CART.
  Nome nodo: lookforward
  Il nodo effettua un costante controllo dell'ambiente di fronte al cart. Permette di riconoscere eventuali ostacoli presenti nella carreggiata e di rallentare o arrestare il veicolo a seconda della loro vicinanza, utilizzando una finestra dinamica (variabile a seconda della velocità del veicolo).

- Package: CART.
  Nome nodo: recordpath
  Utilizzato durante le fasi di guida manuale, permette di registrare un percorso. E' possibile far riprodurre il percorso al Cart mediante il nodo cartpilot.

- Package: CART.
  Nome nodo: cartpilot
  Effettua la navigazione autonoma, se localizzato. Legge i setpoint creati con il nodo recordpath ed invia i comandi necessari alle schede di controllo del motore e dello sterzo. Utilizza il nodo cartplanner per individuare correttamente i setpoint da seguire.

- Package: CART.
  Nome nodo: cartplanner
  Il nodo permette l'individuazione del setpoint del path più vicino (in comunicazione con lookforward).

- Package: AMCL2
  Nome nodo: amcl_node
  Rappresenta il nodo contenente il vero e proprio algoritmo di localizzazione, denominato in onore del lavoro di tesi:

  A*ugusto***M***onte***C***arlo***L***ocalization*.

Appendice C

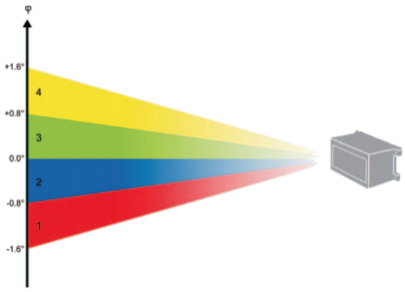
Sensori

Tipologie di sensori utilizzati e posizionamento sul Cart.

C.1 Sick LMS111-10100

Features	
Field of application	Outdoor
Version	Short Range
Light source	Infrared (905 nm)
Laser class	1 (IEC 60825-1 (2007-3))
Field of view	270 °
Scanning frequency	25 Hz / 50 Hz
Angular resolution	0.25 ° 0.5 °
Operating range	0.5 m ... 20 m
Max. range with 10 % reflectivity	18 m
Operating voltage	10.8 V DC ... 30 V DC
Enclosure rating	IP 67 (EN 60529, Section 14.2.7)
Dimensions	105 mm x 102 mm x 162 mm
Weight	1.1 kg, without connecting cables
Protocol (Ethernet)	TCP/IP, OPC
Data transmission rate (Ethernet)	10/100 Mbit

C.2 Sick LD-MRS400001



Features	
Field of application	Outdoor
Version	Long Range
Laser class	1
Field of view	85 °, operating angle with 4 measurement layers, 25° work area expansion with 2 measurement layers (total 110°)
Scanning frequency	12.5 Hz ... 50 Hz
Angular resolution	0.125 ° 0.25 ° 0.5 °
Operating range	0.5 m ... 250 m
Max. range with 10 % reflectivity	50 m
Operating voltage	9 V DC ... 27 V DC
Enclosure rating	
IP 69K	
Dimensions	94 mm x 165 mm x 88 mm
Weight	1 kg
Protocol (Ethernet)	TCP/IP
Data transmission rate (Ethernet)	100 Mbit/s

Nella parte destra della figura possiamo osservare i quattro fasci laser sui quali vengono fatti le scansioni di questo sensore.

C.3 MTI-xsense

Features	
Static accuracy (roll/pitch)	$\leq 0.5°$
Static accuracy (heading)	$\leq 1°$
Digital interface	USB with external converter
Update rate	256hz
Dimensions	58 x 58 x 22 mm
Weight	50 g

C.4 Configurazione dei sensori sul Cart

La configurazione spaziale del posizionamento dei sensori sul Cart segue lo standard ISO 8855 *Road vehicles – Vehicle dynamics and road-holding ability* che prevede l'asse x orientato in avanti, l'asse y orientato verso sinistra e conseguentemente l'asse z orientato verso l'alto. Abbiamo così definito il sistema di riferimento del Cart posizionato come in Figura C.1.

Figura C.1: *La convenzione usata nella tesi per definire il sistema di riferimento del Cart. L'origine è posta in corrispondenza dell'asse posteriore e rappresenta il sistema delle masse sospese.*

I LIDAR sono stati posizionati su supporto metallico progettato, disegnato e sviluppato nel laboratorio IRA appositamente per sostenere per questi sensori. Il supporto è stato denominato *u-rovesciata*. La struttura è composta con barre di alluminio ANTICORODAL 6060 con spessore 8 mm, sufficientemente robuste e leggere per non introdurre oscillazioni dovute al peso dei LIDAR (circa 3 kg) e della struttura stessa. Per quanto concerne i due Sick LMS111 ad un singolo piano di scansione sono stati collocati nelle parti laterali, inclinati di 45° rispetto all'asse z del Cart, in modo tale da permettere una scansione dell'intera area frontale e laterale del Cart. Il LIDAR Sick LDMRS-4001, posizionato nella parte anteriore e centrale del supporto, è

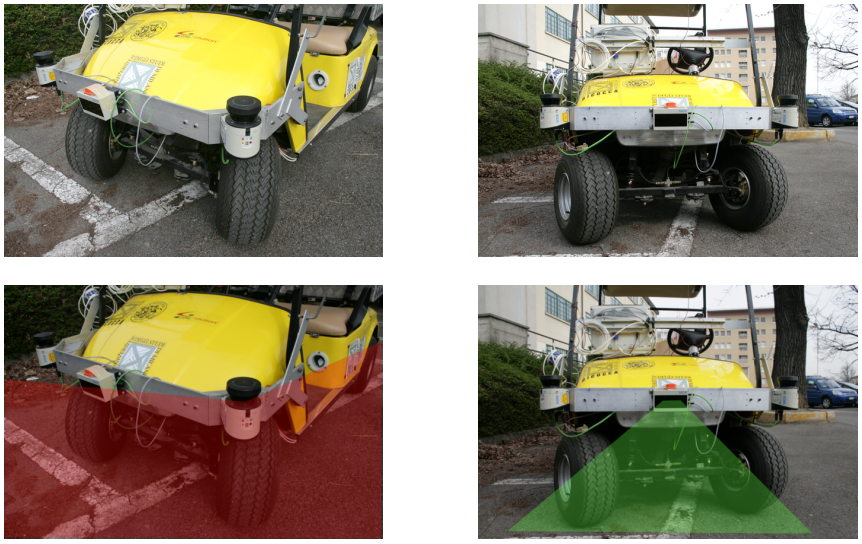


Figura C.2: *Il posizionamento dei LIDAR sul Cart. In (c) e (d) è stata evidenziata l'area di scansione (per quanto riguarda il sensore a quattro piani è stato indicato il piano inferiore).*

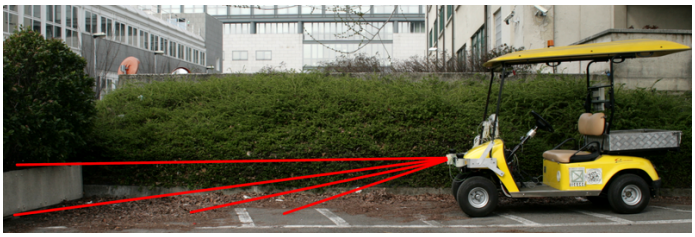


Figura C.3

stato invece orientato con una inclinazione sull'asse y di circa 3°. Questo orientamento è stato scelto in modo tale da avere il piano di scansione superiore parallelo al piano stradale e quindi fornire misure a grande distanza (Figura C.3). I tre piani inferiori permettono di ottenere le distanze dal suolo e risultano fondamentali al fine della corretta localizzazione tridimensionale (si veda il Paragrafo 4.6).

Appendice D

Voxeling

Il procedimento di voxeling è necessario in quanto il lavoro in questa tesi utilizza mappe 3D discretizzate. In questa appendice descriveremo come trasformare mappe vettoriali create con software CAD (ad esempio Autodesk Autocad o Google Sketchup Pro) in mappe raster binarie. Con il termine *voxel* (**vo**lumetric pi**xel**) viene indicata la controparte tridimensionale del bidimensionale *pixel*. Se il pixel può essere rappresentato come un quadrato il voxel a sua volta può essere inteso come un cubo. Le mappe formate da voxel descrivono l'ambiente 3D in formato raster.

Figura D.1: *Esempio di una tazza rappresentata sottoforma di voxel. Immagine tratta da http://www.patrickmin.com/minecraft/*

Per effettuare la trasformazione abbiamo utilizzato il tool Binvox[1] [2]. Questo software supporta la trasformazione in voxel dei seguenti formati Wavefront OBJ, Geomview OFF, Autocad DXF, PLY and STL e VRML 1.0; abbiamo riscontrato risultati differenti a seconda del formato di origine ed i

[1] http://www.cs.princeton.edu/~min/binvox/

[2] Ulteriori info su binvox disponibili su www.minecraftwiki.net/wiki/Binvox

migliori risultati si sono ottenuti con i formati dxf ed VRML9. Vediamo ora il procedimento utilizzato nell'ambito del lavoro di tesi.

Il primo passo necessario consiste nel creare una mappa 3D: nel caso di un ambiente reale è necessaria estrema cura nel dimensionamento di ogni dettaglio in quanto una mappa non fedele all'ambiente renderà meno efficace la corretta localizzazione. Abbiamo utilizzato prevalentemente Google Sketchup Pro 8[3] per la semplicità d'uso ed i bassi requisiti in termini di hardware di questo prodotto. In Figura D.2 è raffigurato l'ambiente di test utilizzato nell'ambito delle simulazioni del modello odometrico 6DoF. L'ambiente prevede un piano di movimento del veicolo con dei rialzi laterali (per la simulazione dei marciapiedi di una strada) ed una rampa.

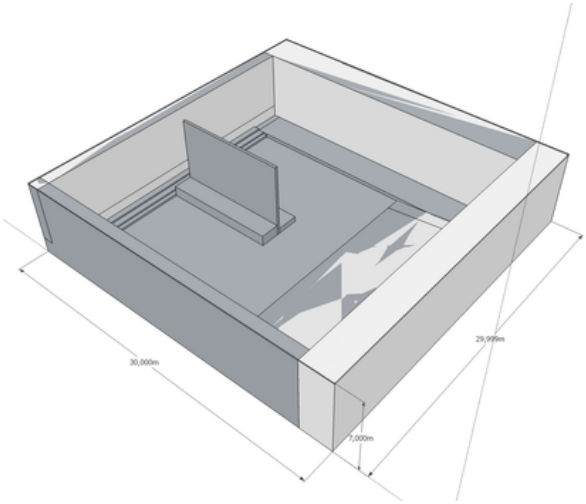


Figura D.2

Una volta disegnata la mappa con Google Sketchup Pro 8 è necessario effettuare dei passaggi intermedi prima di poter creare un file gestibile da Binvox. Abbiamo riscontrato diversi problemi con i file esportati da Sketchup, sia in formato VRLM9 che dxf. Le soluzioni a questi problemi consistono nell'apertura dei file VRLM9/dxf rispettivamente con Blender/Autocad e quindi procedere nuovamente al salvataggio di tali file [4] [5] nello stesso formato.

[3] http://sketchup.google.com/

[4] http://irawiki.disco.unimib.it/irawiki/index.php/Creating3dVoxelMaps

[5] http://www.ros.org/wiki/octomap/Tutorials

Una volta effettuato questo passaggio è possibile utilizzare Binvox efficacemente. Il tool prevede i seguenti parametri di configurazione:

```
Usage:
      binvox [-d <voxel dimension>] [-t <voxel file type>] [-c]
      [-v] <model filespec>

  -d: specify voxel grid size (default 256, max 1024)
  -t: specify voxel file type (default binvox, also
      supported: hips, mira, vtk, raw, schematic, msh)
  -c: z-buffer based carving method only
  -dc: dilated carving, stop carving 1 voxel before
      intersection
  -v: z-buffer based parity voting method only (default
      is both -c and -v)
  -e: exact voxelization (any voxel with part of a triangle
      gets set)(does not use graphics card)
```

Abbiamo ottenuto i migliori risultati utilizzando i parametri $-dc$ ed $-e$. E' inoltre necessario indicare il parametro $-d$ che specifica la scala da utilizzare nella creazione dei voxel. Il limite di 1024 voxel influisce sulla minima dimensione dei voxel: per mappe molto grandi come ad esempio l'area del garage U5 si riescono ad ottenere voxel di circa 10 cm. In Figura D.3 è possibile osservare l'ambiente di Figura D.2 ottenuto dopo il processo di *voxeling*.

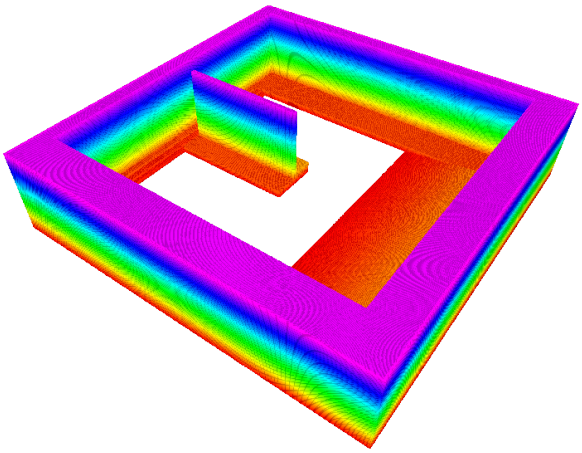


Figura D.3: *L'ambiente sottoforma di voxel. I colori variano con la quota.*

Bibliografia

[1] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[2] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots, 2001.

[3] D. Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[4] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22:2003, 2003.

[5] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press, 2005.

[6] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[7] A. Sakai, Y. Tamura, and Y. Kuroda. An efficient solution to 6dof localization using unscented kalman filter for planetary rovers. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4154–4159, oct. 2009.

[8] Rainer Kümmerle, Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Monte carlo localization in outdoor terrains using multi-level sur-

face maps. In *In Proc. of the International Conference on Field and Service Robotics (FSR*, 2007.

[9] Anna Petrovskaya and Sebastian Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26:123–139, 2009. 10.1007/s10514-009-9115-1.

[10] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. pages 343–349, 1999.

[11] A. Gelb. *Applied optimal estimation*. MIT Press, 1974.

[12] D.W. Ruck, S.K. Rogers, M. Kabrisky, P.S. Maybeck, and M.E. Oxley. Comparative analysis of backpropagation and the extended kalman filter for training multilayer perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(6):686–691, 1992.

[13] V. M. Becerra, P. D. Roberts, and G. W. Griffiths. Applying the extended kalman filter to systems described by nonlinear differential-algebraic equations. *Control Engineering Practice*, 9(3):267–281, March 2001.

[14] T. Bellemans, B. De Schutter, G. Wets, and B. De Moor. Model predictive control for ramp metering combined with extended Kalman filter-based traffic state estimation. In *Proceedings of the 2006 IEEE Intelligent Transportation Systems Conference (ITSC 2006)*, pages 406–411, Toronto, Canada, sep 2006.

[15] Zia Khan, T. Balch, and F. Dellaert. Mcmc-based particle filtering for tracking a variable number of interacting targets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(11):1805–1819, 2005.

[16] J. Carpenter, P. Clifford, and P. Fernhead. An improved particle filter for non-linear problems, 1997.

[17] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[18] Andrew D. Bagdanov, Fabrizio Dini, Alberto Del Bimbo, and Walter Nunziati. Improving the robustness of particle filter-based visual trackers using online parameter adaptation. In *Proc. of IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS)*, London, UK, September 2007. IEEE Computer Society.

[19] Jiang Li and Chin-Seng Chua. Transductive local exploration particle filter for object tracking. *Image Vision Comput.*, 25(5):544–552, 2007.

[20] Xue Wang, Sheng Wang, and Jun-Jie Ma. An improved particle filter for target tracking in sensor systems. *Sensors*, 7(1):144–156, 2007.

[21] Sebastian Thrun, Dieter Fox, and Wolfram Burgard. Monte carlo localization with mixture proposal distribution. In *in Proc. 17th National Conf. on Artificial Intelligence (AAAI-2000). AAAI Press/The*, pages 859–865. MIT Press, 2000.

[22] G.E.P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *Ann. Math. Stat.*, 29:610–611, 1958.

[23] Brian Gerkey. Amcl. http://ros.org/wiki/amcl.

[24] Jack Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[25] M. Humbert, N. Gey, J. Muller, and C. Esling. Determination of a Mean Orientation from a Cloud of Orientations. Application to Electron Back-Scattering Pattern Measurements. *Journal of Applied Crystallography*, 29(6):662–666, Dec 1996.

[26] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '85, pages 245–254, New York, NY, USA, 1985. ACM.

[27] Brian Gerkey and Tony Pratkanis. Map server. http://ros.org/wiki/map_server.

[28] Patrick Min. Binvox. http://www.cs.princeton.edu/ min/binvox/.

[29] J. Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Series. O'Reilly, 2007.