UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

REALIZZAZIONE DI UN SISTEMA DI LOCALIZZAZIONE PROBABILISTICO BASATO SU FILTRO PARTICELLARE PER ROBOT MOBILI

Relatore:
Chiar.mo Prof. STEFANO CASELLI

Correlatori:
Ing. FRANCESCO MONICA
Dott. Ing. MONICA REGGIANI

Tesi di laurea di:
FRANCESCO PEDRIELLI

ANNO ACCADEMICO 2003-2004

Alla mia famiglia.

La conclusione del lavoro di tesi, pur lasciando un po' di nostalgia nell'animo, ha regalato molta soddisfazione per aver portato a termine un progetto difficile ma estremamente appassionante. Ringrazio per questo il prof. Stefano Caselli che mi ha permesso di affrontare un tema così interessante, per la sua disponibilità e per i consigli che mi hanno consentito di concludere al meglio il mio lavoro. Non posso di certo dimenticare la disponibilità e la gentilezza con la quale la Dott. Ing. Monica Reggiani mi ha seguito e consigliato in questi mesi. Infine ringrazio l'Ing. Francesco Monica per la pazienza e la simpatia con la quale è riuscito a soddisfare l'incredibile quantità di richieste di chiarimento.

Grazie ai miei genitori Angela e Claudio per l'affetto che ho ricevuto da loro, per i consigli saggi che mi hanno dato e che, qualche volta, ho seguito, per avermi permesso di raggiungere un traguardo importante come questo e per la fiducia (mal riposta ovviamente) che hanno dato a loro figlio.

Non posso dimenticare mia sorella Giulia che negli anni dell'Università è stata per me anche un'amica di cui ci si può fidare e con la quale ho potuto condividere tanti pensieri e tanti segreti.

Ringrazio i nonni Guido e Laura per il loro affetto e per il ragù alla bolognese più buono di tutta la provincia di Bologna.

Un pensiero lo rivolgo a Luciano e Marcello che non vedo spesso, ma a cui voglio super bene. Non posso dimenticare le mie zie preferite ziAnto e ziaMery dato che mi vogliono tanto bene e che so di essere il loro nipote over 23 preferito. Ringrazio zioVittorio per i panzerotti più buoni della provincia di Béri e lo zioMimmo per essere stato il medico più bravo della Puglia quando ha salvato la mia adolescenza.

Bene...bene a questo punto ho concluso. Grazie a tutti... Sto scherzando! Lo so che non c'è cascato nessuno perché si vede che il testo non è finito però non potevo non inserire una GAG.. cercate di capirmi!

Un pensiero speciale per MARGI, che devo ringraziare innanzitutto per avermi aiutato in modo fondamentale nella realizzazione della tesi: senza di lei starei ancora cercando di fare una pianta sensata del corridoio del CUBO 1. Inoltre la devo ringraziare perché è Margi: non c'è mai stato un momento in cui non mi abbia di-

mostrato affetto, non mi sia stata vicina e si merita tutto il bene SMEMMNSQ che le voglio per questo.

Di amici che voglio ringraziare ce ne sono tanti davvero... ringrazio BARUFFA, dalle elementari alla laurea baruf c'è, TONELLO FVIP, per le lunghe chiacchierate e perché mi manca dato che mentre scrivo è in Spagna che pedala come un pazzo, FABIONE, per le studiate insieme, le pause, le discussioni, le birre insieme e l'amicizia, FABIETTO, un amico al quale devo un ringraziamento particolare per essere sempre stato dalla mia parte nei momenti difficili(fortunatamente pochissimi) e perché ogni serata con lui è un successone, il PINNA!, [1] *per la sincerità che mi ha dimostrato in tutti questi anni, FABRI, amico onnipresente che alla fine di ogni giornata di lavoro alla tesi mi mandava un mes che mi apriva nuovi orizzonti per la serata, GALVO, che mi consola un po' perché è l'unico più sbadato di me, SILVIA, perché mi vuole bene anche se ogni tanto la faccio arrabbiare quando non la chiamo per troppo tempo, RENATO, l'uomo che io ho sposato..., GIACOMO, una persona che mi ha insegnato tanto e mi ha fatto vedere tante cose in modo diverso, CARLOTTA, LAURA, FRANCESCA e FEDE, con le quali sono amico da tanti anni e con cui condivido tanti ricordi bellissimi, LIBERA, un'amica ritrovata, BUBBA, per le risate che mi fa fare, MANU, il mio amico fotomodello e BISTEC, MAX, BURO, FAFFO, per il record di vittorie a calcetto, RAPI, SERENA, GIULIA, BIANCA, ALESSANDRA, ANNALISA, VALE, ILARIA, MIKY, CHIARA, MARCHE, per avermi insegnato a studiare, PAOLO, ROSA, la mia paziente compagna di laboratorio, tutti i ragazzi della palazzina, KLAAS GADEYNE, ROBI BAGGIO, la 206, fedele compagna per 80000 km, e la mitica RARINANTES PARMA!*

[1] Minojo Pinnati detto Michele, famoso per le sue camice e ricercato per i seguenti atti: organizzazione di tornei illegali di calcetto elettronico e di gare di salto con rane di carta, affissione di fruttini al muro, stage diving sul pavimento, bagno notturno nella piscina del campus alla festa del Galvo.

Indice

Capitolo 1

Introduzione

L'abilità di navigare in un ambiente è fondamentale per qualsiasi sistema intelligente. La robotica mobile è un'area di ricerca che si occupa del controllo di veicoli autonomi e semiautonomi. Il problema chiave che la distingue dalla robotica dei manipolatori e dalla visione artificiale è la capacità di apprendere lo spazio parzialmente osservabile, ovvero uno spazio che non è possibile osservare interamente da un singolo punto. L'acquisizione incrementale di conoscenza, l'abilità di riconoscere oggetti o luoghi familiari e il saper dare risposte *real time* sono le capacità che permettono la navigazione in ambiente parzialmente osservabile.

"Locomozione", "percezione" e "ragionamento" sono i concetti chiave che guidano la robotica mobile. I primi due termini riguardano la parte fisica del robot: ruote, bracci meccanici, sensori di prossimità sono gli strumenti utilizzati dal robot per navigare nell'ambiente, per interagire con esso e per osservarlo. Il ragionamento, per un robot, è rappresentato da algoritmi, da principi computazionali [1], e occupa una posizione centrale per importanza nell'ambito della ricerca su robot mobili.

Lo studio del moto di un robot può essere affrontato a livello dinamico, identificando le forze applicate al veicolo, oppure a livello cinematico, che implica la determinazione delle relazioni matematiche descriventi il movimento del veicolo attraverso l'utilizzo di grandezze come velocità, accelerazione, tempo e spazio.

La percezione è il requisito fondamentale anche per i più semplici comportamenti robotici. I sensori del robot misurano grandezze come la sua velocità, le forze ad esso applicate oppure la struttura dell'ambiente circostante. I dispositivi

sensoriali e gli algoritmi che interpretano i valori misurati possono raggiungere un elevato grado di complessità. I sensori dedicati all'osservazione dell'ambiente appartengono a due categorie principali: i sensori visivi, che utilizzano la luce riflessa dagli oggetti nello spazio circostante per elaborare la loro struttura, oppure i sensori non visivi, come dispositivi ultrasonici, laser, a radiazione infrarossa (sensori di prossimità) e *bumper* (sensori di contatto).

Oltre al movimento e alla percezione dell'ambiente, il robot deve possedere la capacità di pianificare un moto intelligente. Nell'ambito della robotica mobile sono state sviluppate numerose tipologie di architetture per la realizzazione di una navigazione affidabile nello spazio.
Il problema della pianificazione del moto nasce dall'esigenza di poter chiedere al robot di dirigersi in un punto dello spazio senza dover specificare tutte le traiettorie che il veicolo deve percorrere per arrivare alla meta. Al fine di realizzare algoritmi di questo tipo è necessaria una rappresentazione dello spazio che possa essere elaborata dal robot: con il termine *mapping* viene indicato il processo di realizzazione della mappa dell'ambiente attraverso la navigazione e la percezione dell'ambiente stesso. Mappare l'ambiente implica anche conoscere quali sono le percezioni che il robot ha in ogni punto dello spazio e, quindi, la possibilità di riconoscere facilmente luoghi già visitati.
Una volta che il robot possiede una rappresentazione dello spazio navigabile può essere comandato a muoversi verso un punto identificabile sulla mappa. Raggiunta la meta prestabilita il veicolo potrà fermarsi. Ma come può sapere il robot di aver raggiunto il punto stabilito? Dal quesito appena formulato ne nasce un altro ancora più banale: dove è il robot? Questo problema è affrontato dal settore specifico della robotica mobile che si occupa della localizzazione. In realtà *mapping* e localizzazione sono intimamente legati tra loro perché identificare nello spazio degli oggetti percepiti significa sapere la posizione del punto di osservazione, viceversa il riconoscimento della posizione necessita la conoscenza della rappresentazione dello spazio.

In generale, con la localizzazione di un veicolo autonomo si affrontano due problematiche: la localizzazione globale, ovvero la capacità di localizzarsi senza l'aiuto di nessuna informazione data a priori, e il *position tracking*, con cui si indica la capacità di monitorare l'evoluzione della posizione del robot partendo da un punto

noto dello spazio.

L'obiettivo di questa tesi è studiare il problema della localizzazione di un robot mobile partendo da una rappresentazione dello spazio fornita a priori. La tesi indaga, in particolare, l'utilizzo di una mappa metrica, ossia la planimetria dell'ambiente di sperimentazione, come substrato conoscitivo posseduto dal robot. La conoscenza a priori della mappa elimina il problema del *mapping*, ma introduce difficoltà legate all'assenza di informazioni sulle percezioni del robot nei punti dello spazio navigabile: per riconoscere un luogo, infatti, bisogna conoscerne le caratteristiche. Questo implica che il robot deve sapere per ogni luogo che visita quali dovrebbero essere le sue percezioni.
Il lavoro si è dunque articolato su due fronti: il primo è lo studio di una procedura per prevedere le osservazioni che il robot ha in ogni punto dello spazio. Il secondo è lo sviluppo di un sistema che, partendo dalla conoscenza a priori della rappresentazione metrica dell'ambiente, attui un processo deduttivo al fine di stimare la posizione del robot in termini di coordinate metriche in un sistema si riferimento assoluto.

La difficoltà principale che qualsiasi algoritmo di localizzazione deve superare è la forte rumorosità dell'ambiente di navigazione e dei sensori del robot. Il rumore è difficilmente modellizzabile perché può cambiare fortemente le sue caratteristiche a seconda della porzione di spazio in cui viene effettuata la misura. I problemi di stima dello stato in cui i disturbi hanno un forte impatto sul sistema possono essere affrontati mediante un approccio probabilistico. Nella tesi l'utilizzo dell'approccio probabilistico ha permesso di spostare il problema della localizzazione su un ambito puramente statistico e di utilizzare strumenti matematici evoluti come i filtri bayesiani ricorsivi.

I capitoli che costituiscono la tesi sono organizzati come segue. Nel capitolo II viene illustrato lo stato dell'arte nell'ambito della localizzazione in generale e, più specificamente, dell'approccio probabilistico alla localizzazione. Il capitolo III affronta invece lo sviluppo dell'algoritmo ciclico di stima della posizione. Nel capitolo IV della tesi vengono presentati gli strumenti software utilizzati. Il capitolo V descrive la struttura del programma realizzato. Nel capitolo VI vengono illustrati i risultati sperimentali ottenuti in simulazione e quelli ottenuti nel caso reale facendo navigare il robot all'interno del corridoio del Dipartimento di Ingegneria dell'Infor-

mazione (Pal.1). Infine, sono presentate delle considerazioni conclusive sui risultati sperimentali ottenuti e sui possibili sviluppi futuri del sistema di localizzazione.

Capitolo 2

La localizzazione

2.1 Introduzione al problema

La conoscenza della posizione del robot può essere utile per l'esecuzione di numerose operazioni. Problemi di pianificazione del moto del robot sono risolvibili partendo dal presupposto di conoscere "dove è il robot". Lo stesso si può affermare nel caso debbano essere compiute operazioni su oggetti specifici.

Per un essere umano identificare la propria posizione in un ambiente chiuso è molto semplice: chiunque è in grado di indicare approssimativamente dovi si trovi su una mappa di un ambiente che sta osservando, o di localizzarsi relativamente agli oggetti che lo circondano. Va, però, considerato che l'utilizzo umano di una mappa è caratterizzato da un processo cognitivo di alto livello, attraverso il quale viene creata una corrispondenza tra la rappresentazione dell'ambiente e il mondo reale. Questa corrispondenza può essere resa più semplice modificando l'ambiente, per esempio, attraverso cartelli in cui è indicato il nome del luogo: stazioni ferroviarie, ingressi nelle città o nomi delle vie sono degli indicatori di posizione.

In robotica le prime ricerche sulla navigazione *map based* furono ispirate proprio dall'esperienza umana, ipotizzando la possibilità di correggere gli errori percettivi e motori con processi cognitivi, oppure attraverso la modifica dell'ambiente. Studi etologici hanno fatto emergere la possibilità che gli animali facciano uso di mappe per la navigazione. In questo senso alcuni approcci hanno utilizzato reti neurali per simulare il funzionamento dell'ippocampo dei ratti, ovvero la parte del

cervello da essi utilizzata per localizzarsi [2] [3].

Per un robot ”vedere” significa elaborare i dati che i suoi sensori gli inviano. Per costruire un buon sistema di localizzazione i fattori che risultano determinanti per stimare la posizione di un robot sono due: la qualità delle sorgenti sensoriali e la rappresentazione dello spazio.

2.2 I dati sensoriali

Nella navigazione *map-based* possono essere identificate due distinte sorgenti di informazione [2]:

- *Sorgenti idiotetiche* o *propriocettive*
- *Sorgenti allotetiche* o *eterocettive*

Le sorgenti sensoriali idiotetiche forniscono le informazioni interne che riguardano i movimenti del robot.
Esse possono essere la velocità, l’accelerazione, i movimenti degli arti nonchè la velocità di rotazione delle ruote. Integrando queste grandezze è possibile risalire alla posizione del robot nello spazio 2D.
Se la localizzazione durante il movimento del robot viene effettuata basandosi esclusivamente su questo tipo di informazione sensoriale, allora si parla di *dead-reckoning*.
Il termine idiotetico è proposto da Jean-Arcady Meyer [2] [3] e ha origine dalla letteratura biologica. Esso sta a indicare quell’insieme di dati sensoriali che in robotica vengono identificati con il termine odometria [2].

Le informazioni allotetiche possono essere prodotte da sorgenti visive, olfattive, tattili, che, nel caso dei robot, possono essere dati ricavati da laser, sonar, telecamere. Anche il termine allotetico deriva dalla letteratura biologica e corrisponde a espressioni del tipo ”osservazione”, ”percezione” o dati sensoriali nel contesto della robotica. I dati allotetici possono essere utilizzati per riconoscere un luogo o una situazione, ma difficilmente si può pensare che, traducendoli in un modello metrico ed utilizzandoli come unica fonte sensoriale, si possa ottenere una stima affidabile della posizione [2].

2.3 La rappresentazione della mappa

Si è soliti identificare due tipi di rappresentazione spaziale:

- *Topologica*
- *Metrica*

In uno spazio topologico l'ambiente è rappresentato da un insieme di luoghi distintivi. Ad ognuno di essi il robot associa una particolare informazione sensoriale che li rende riconoscibili.
Questo tipo di rappresentazione è conveniente per il *planning* o per la risoluzione di problemi, perché la grandezza dello spazio di ricerca è limitata rispetto alle possibili traiettorie identificabili nello spazio continuo. Da quest'ultima affermazione s'intuisce che le mappe topologiche hanno una proprietà intrinseca: la discretizzazione dello spazio. Ogni luogo distintivo corrisponde a un nodo all'interno della mappa. I nodi possono essere definiti autonomamente dal robot o specificati manualmente e sono collegati tra di loro da *link*, ai quali può essere legata l'espressione della distanza dei nodi che collegano.
Il vantaggio delle mappe topologiche è che non necessitano di un modello sensoriale metrico per tradurre le informazioni provenienti dai sensori in un sistema di riferimento cartesiano. L'unico requisito è quello di avere un efficiente meccanismo di immagazzinamento dei dati che permetta di stabilire una corrispondenza tra una sensazione sensoriale e un luogo distintivo.

In uno spazio metrico l'ambiente è invece rappresentato come un insieme di oggetti identificabili mediante coordinate in un sistema di riferimento cartesiano. La possibilità di dare informazioni metriche è fornita dai dati idiotetici. Mentre nel caso delle mappe topologiche i dati allotetici e idiotetici possono essere utilizzati alternativamente o parallelamente, le mappe metriche permettono la fusione di queste due tipologie di sorgenti sensoriali. La stima della posizione è continua nello spazio 2D e, perciò, molto più precisa di quella effettuabile in uno spazio topologico. Oltretutto la visualizzazione di una mappa metrica è molto più comprensibile dall'utente umano.
La non ambiguità della definizione dei luoghi fa si che le mappe metriche siano più

semplici da costruire.
Il problema principale che presentano è la forte dipendenza dai dati idiotetici[2].

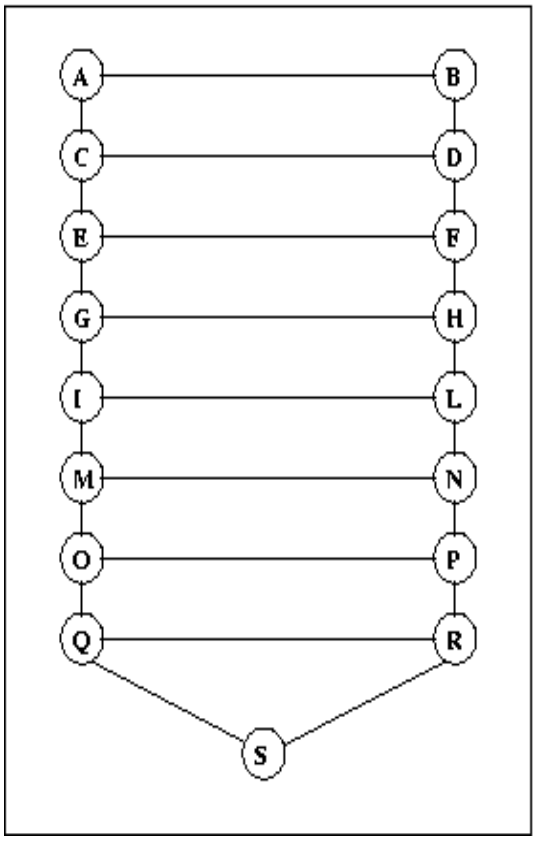

Figura 2.1: Esempio di Mappa topologica del corridoio della Palazzina 1, Sede Scientifica di Ingengeria. I nodi corrispondono a luoghi facilmente riconoscibili: A)Ingresso laboratorio di robotica, B)Ingresso laboratorio di automatica, C)Ingresso laboratorio workstation, D)Ingresso ufficio 1, E)Ingresso bagno 1, F)Ingresso laboratorio Parma2, G)Ingresso bagno 2, H)Ascensore, I)Uscita, L)Scale, M)Ingresso ufficio 2, N)Ingresso ufficio 3, O)Ingresso sala cluster, P)Ingresso ufficio 5, Q)Ingresso laboratorio di visione artificiale, R)Ingresso ufficio 4, S)Ingresso laboratorio AOT

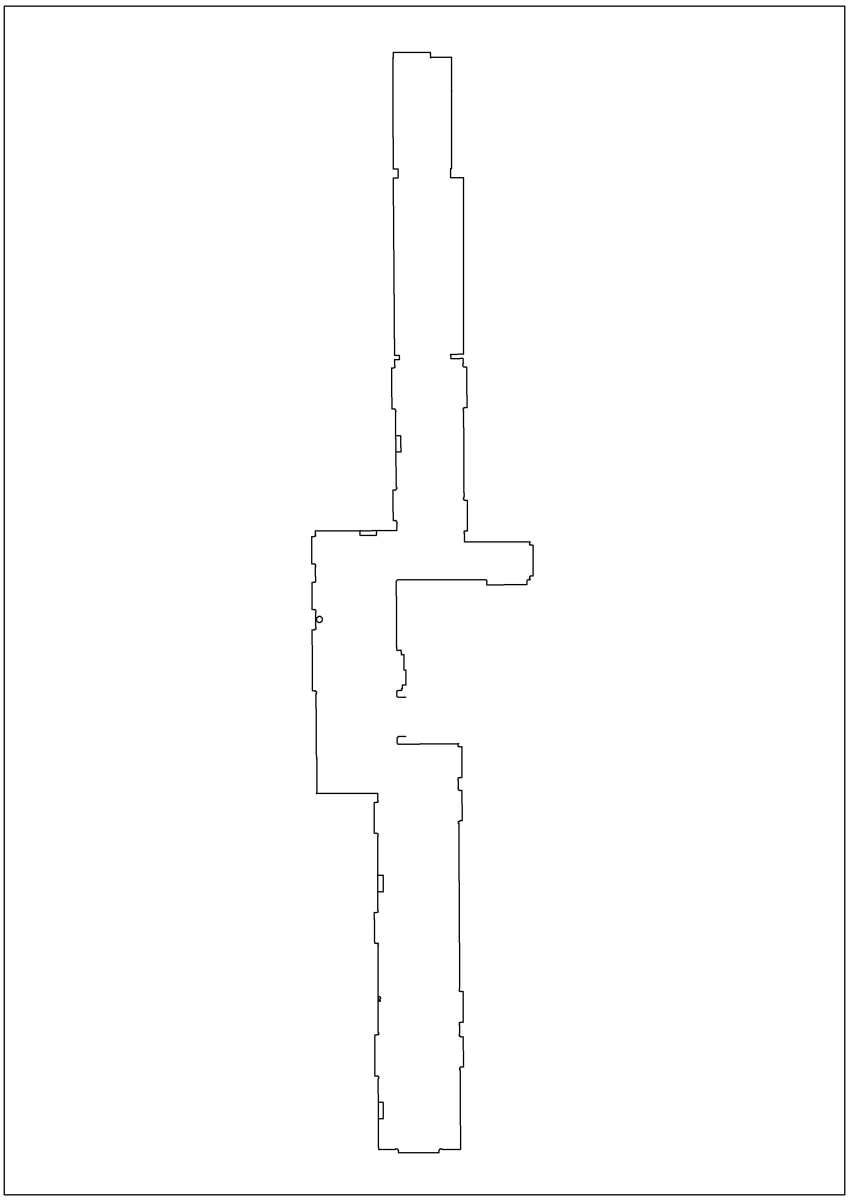

Figura 2.2: Mappa metrica del corridoio della Palazzina 1 Sede Scientifica di Ingengeria.

2.4 Occupancy grid

Il fatto che il robot debba riconoscere la sua posizione mentre si muove, significa che l'algoritmo di localizzazione non deve assorbire una potenza computazionale tale da disturbare l'algoritmo di movimento. Come sottolineato precedentemente la scelta del modello spaziale e, quindi, del tipo di mappa, può influire molto sulla gestione delle risorse computazionali. Chiaramente meno precisa è la mappa e meno l'algoritmo sarà pesante, ma si perderà accuratezza nelle stime. In questa tesi viene indagata una metodologia di rappresentazione metrica dello spazio che determina un carico computazionale accettabile. Essa si basa, inevitabilmente, su un'operazione di discretizzazione.
Si parla di *occupancy grid* nel caso si suddivida lo spazio 2D in celle. Ogni cella ha una dimensione prestabilita, rappresenta una certa area dello spazio e ha forma quadrata. Una cella può trovarsi in soli due stati: piena o vuota. Utilizzare un modello spaziale di questo tipo rende l'elaborazione della mappa piuttosto pesante se questa è di dimensioni notevoli e con elevata risoluzione (rapporto tra numero di celle e dimensione della mappa), inoltre non si evitano i problemi legati alla dipendenza dalle fonti sensoriali idiotetiche.

2.5 Strategie di localizzazione

Il tema della localizzazione si può ricondurre a due problematiche fondamentali: localizzazione locale (*position tracking*) e localizzazione globale.
Nel primo caso il robot ha una stima della sua posizine iniziale che non differisce troppo da quella effettiva. Nel secondo caso, invece, si deve localizzare senza avere informazioni sul suo stato precedente. Ci si riferisce alla localizzazione globale anche con il termine di *lost-robot problem*.
Nel *position tracking*, date le fonti allotetiche, la posizione del robot deve essere corretta per riflettere meglio le due percezioni. Se invece la localizzazione è globale, il robot deve trovare la regione dello spazio che corrisponde meglio ai suoi dati sensoriali.
Nel caso di approccio topologico, la localizzazione locale è più semplice perché il numero di nodi che deve essere preso in considerazione è limitato.

Si possono classificare le strategie di localizzazione in tre categorie [2]:

1. La posizione viene stimata senza l'utilizzo di fonti idiotetiche. Il modello su cui si fonda questa metodologia ipotizza che le informazioni allotetiche siano sufficientemente affidabili e precise da costituire l'unica fonte di informazione.

2. *Single hypothesis tracking*: si utilizzano sia le fonti idiotetiche che quelle allotetiche per effettuare un'unica ipotesi sulla posizione del robot. Si parte da una stima dello stato prededente e dai dati idiotetici per aggiornare la previsione (i dati idiotetici servono a capire quanto il robot si è mosso rispetto alla posizione precedente). Se la stima è scorretta il robot si perde.

3. *Multiple hypothesis tracking*: anche in questo caso vengono utilizzate sia le fonti idiotetiche che quelle allotetiche: invece di mantenere solo l'ipotesi più credibile vengono effettuate una serie di ipotesi sullo stato del robot e aggiornate in parallelo.

Non sempre la posizione e la mappa vengono rappresentate nello stesso modo. Anche in questo caso le categorie identificabili sono tre:

- La mappa è topologica, mentre la posizione è rappresentata come un nodo nella mappa.

- La mappa è topologica, ma la posizione è rappresentata in un *framework* metrico.

- Sia la mappa che la posizione sono rappresentate in uno spazio metrico.

2.6 Definizione dei landmark e problema della corrispondenza

In una rappresentazione topologica della mappa è necessario individuare dei luoghi all'interno dell'ambiente che abbiano caratteristiche che li rendano facilmente riconoscibili. Queste porzioni di spazio sono chiamate *landmark*. La loro definizione è

dipendente dal tipo di sorgenti allotetiche che vengono utilizzate. Quando il robot si trova vicino a un landmark le sue percezioni sensoriali devono subire una forte discontinuità.

Nel caso il robot utilizzi sensori di prossimità come sonar, dispositivi a radiazione infrarossa o laser, un luogo distintivo deve produrre un radicale cambiamento dei valori misurati. Un possibile landmark potrebbe essere una porta aperta, oppure un cambiamento di direzione delle pareti. Come mostato in figura 2.1 tali luoghi equivalgono a nodi della mappa. Le caratteristiche di ogni landmark devono essere conosciute dal robot. Quest'ultimo durante la navigazione, ogni volta che percepisce una forte variazione dei suoi input sensoriali, deve effettuare l'operazione di *matching*. Durante questa fase si confrontano i valori allotetici con quelli associati ai nodi della mappa. Una volta stabilita la corrispondenza tra valore percepito e valore della mappa, il robot calcolerà la sua posizione relativamente al nodo selezionato. Se la posizione deve essere restituita in modo topologico allora l'algoritmo potrà limitarsi a dare informazioni del tipo "sono tra i nodi C D ed E". Se, invece, si vogliono conoscere le coordinate spaziali, si potranno usare tecniche, ad esempio, di triangolazione per trovare la distanza relativa tra robot e landmark. Identificata questa quantità, la posizione assoluta si ricaverà conoscendo le coordinate metriche del luogo distintivo.

Spesso i landmark sono artificiali: l'utente inserisce oggetti facilmente riconoscibili nell'ambiente per aiutare il robot. Se le sorgenti allotetiche sono anche sensori visivi come telecamere, allora un luogo distintivo potrà essere riconosciuto in base al suo colore o agli *edge* visibili sull'immagine che li ritrae.

Se la rappresentazione dell'ambiente non è topologica, il problema del *matching* non riguarda più il riconoscimento dei landmark. Si tratta in questo caso di stimare le percezioni che si avrebbero in ogni punto dello spazio e confrontarle con quelle attuali. Se la mappa è stata costruita dal robot, allora è possibile creare una struttura dati che contenga le percezioni legate a ogni punto dello spazio esplorato e utilizzarle per identificare le corrispondenze durante la navigazione. In caso contrario è necessario individuare una metodologia per stimare le osservazioni del robot partendo dalla mappa metrica dell'ambiente.

2.7 Modellizzazione del sistema e delle misurazioni

Molte applicazioni ingegneristiche richiedono la stima dello stato per poter controllare il sistema. Nel caso della localizzazione esso deve esprimere la posizione del robot. Il vettore di stato è chiamato statico quando non cambia nel tempo, dinamico quando cambia secondo il modello di sistema, in funzione dello stato all'istante precedente, e dell'*input*. Quest'ultimo riguarda la sensorialità interna, che indica come le coordinate esprimenti la posizione evolvono , ma non ne da una visione in un sistema di riferimento assoluto.

Il modello di sistema è soggetto a disturbi rumorosi di cui si suppone essere conosciute le caratteristiche come, ad esempio, la media e la varianza. L'incertezza legata alle informazioni sensoriali interne rende la stima dello stato sempre più incerta con l'evolvere nel tempo. Per compensare ciò si rendono necessarie le percezioni sensoriali esterne, le cui misure danno informazioni sul valore assoluto della posizione.
Quando i sensori non osservano direttamente e accuratamente lo stato, ad esempio perché non c'è una relazione univoca tra stato stesso e osservazioni, allora si utilizzano stimatori o filtri per calcolarlo.
Il filtro contiene informazioni sia sul modello del sistema, che sul modello delle misure. Il primo, come si evince da quanto è stato già descritto, rappresenta la parte inosservabile del sistema, mentre il secondo quella osservabile. Il fine di questo strumento è mettere in relazione lo stato, l'odometria e le osservazioni (o misurazioni). Anche al modello delle misurazioni è associata una certa incertezza, caratterizzabile come rumore di cui si suppongono conosciute le caratteristiche.

Formalizzando quanto precedentemente affermato:
siano x_k, u_k, ν_k rispettivamente lo stato e gli ingressi del sistema, il rumore che affligge il sistema all'istante k, allora l'evoluzione forzata dello stato del sistema è descritta da

$$x_k = f(x_{k-1}, u_{k-1}) + \nu_k \tag{2.1}$$

siano invece z_k, s_k e ϵ_k rispettivamente le misure, i parametri sensoriali del robot, e il rumore che afflice il modello delle misurazioni all'istante k, allora il modello percettivo del sistema è descritto da

$$z_k = h(x_{k-1}, s_{k-1}) + \epsilon_k \tag{2.2}$$

Nella tabella riassuntiva sottostante vengono specificati i parametri necessari a descrivere la dinamica del sistema [4]

Simbolo	**Descrizione**
x	Vettore di stato x, y, θ. E' inosservabile.
z	Vettore delle misurazioni. Contiene i dati derivanti dalle percezioni sensoriali. E' la variabile che descrive la parte osservabile del sistema.
u	Vettore degli *input*. E' la variabile di controllo del sistema.
s	Parametri sensoriali
f	Notazione funzionale che modellizza il sistema legando la sua parte inosservabile a quella controllabile.
g	Notazione funzionale che modellizza le misurazioni legando la parte osservabile del sistema allo stato.
θ_f	Parametro generico contente informazioni sul modello di sistema e sulla sua incertezza.
θ_g	Parametro generico contente informazioni sul modello delle misurazioni e sulla sua incertezza.

Tabella 2.1: Nomi dei simboli

2.8 Approccio bayesiano

Lo stato dell'arte degli algoritmi di localizzazione presenta una caratteristica che li accomuna: sono probabilistici. Alcuni lo sono esplicitamente, in quanto si fondano su modelli stocastici, altri utilizzano tecniche non specificamente probabilistiche, ma che, sotto determinate ipotesi, possono essere interpretate come tali [5].

La ragione della popolarità delle tecniche probabilistiche deriva dal fatto che le misure sensoriali sono fortemente affette da errori. Il rumore che affligge le percezioni è complesso e difficile da modellizzare.

Dato il modello di sistema e quello delle percezioni, gli ingressi, i parametri e le misure sensoriali, l'obiettivo è quello di stimare lo stato $x(k)$. Nel caso di localizzazione metrica quest'ultimo è descrivibile univocamente mediante la terna $\{x, y, \theta\}$.

La rumorosità legata sia al modello di sistema che al modello sensoriale, fa si che l'approccio bayesiano sia un'ottima soluzione perché permette di modellizzare l'incertezza con una funzione densità di probabilità. Se indichiamo la PDF (*Probability Density Function*) con $P(x(k))$ allora $x(k)$ viene considerata una variabile aleatoria. Questo equivale a spostare il problema della localizzazione su un ambito puramente statistico, ovvero a costruire un algoritmo che identifichi il punto sullo spazio tridimensionale degli stati in cui la densità di probabilità raggiunge il suo massimo. Siano

$$X(K) = [x(0) \ ... \ x(k)]; \quad Z(K) = [z(1) \ ... \ z(k)];$$
$$U(K) = [u(0) \ ... \ u(k)]; \quad S(K) = [s(1) \ ... \ s(k)];$$

le evoluzioni dello stato, delle osservazioni, degli ingressi e dei parametri sensoriali,

$$X_K = [x_0 \ ... \ x_k]; \quad Z_K = [z_1 \ ... \ z_k];$$
$$U_K = [u_0 \ ... \ u_k]; \quad S_K = [s_1 \ ... \ s_k];$$

i valori specifici assumibili dalle variabili aleatorie tempo varianti sopra elencate,

$$F_K = [f_0 \ ... \ f_k]; \quad G_K = [g_1 \ ... \ g_k];$$

i valori assumibili dalle funzioni che modellizzano lo stato in funzione degli ingressi, e le osservazioni in funzione dello stato.

La probabilità che una variabile aleatoria assuma un valore specifico x_k è per uno spazio degli stati discreto $P(x(k) = x_k)$, e per uno spazio degli stati continuo $P(x_k \leq x(k) \leq x_k + dx_k) = P(x(k)) = x_k dx_k$. Anche questo tipo di probabilità verrà indicato con $P(x_k)$.
I filtri probabilistici calcolano la pdf di una variabile $x(k)$ date le osservazioni $Z(K) = Z_k$, gli ingressi $U(k-1) = U_{k-1}$, i parametri dei sensori $S(K) = S_k$, i parametri del modello θ_f e θ_g, i modelli di sistema e delle osservazioni F_{k-1} e G_k, e la pdf $P(x(0))$ chiamata in letteratura *prior* [4]

$$Post(x(k)) \equiv P(x(k)|Z_k, U_{k-1}, S_k, \theta_f, \theta_g, F_{k-1}, G_k, P(x(0))) \tag{2.3}$$

Questa PDF condizionata è spesso chiamata *a posteriori pdf* e denotata mediante $Post(x(k))$. L'utilizzo di $Post(x(k))$ viene detto *diagnostic reasoning*: data la causa (i dati), si trovano le variabili interne (stato), che le possono motivare. Questo è un approccio molto più pesante del *casual reasoning*: date le variabili interne (stato), si predicono le cause(dati). La regola di Bayes lega il primo approccio al calcolo di due *causal problems* [4]:

$$\begin{aligned} Post(x(k)) = \alpha P(z_k|x_k, Z_{k-1}, U_{k-1}, S_k, \theta_f, \theta_g, F_{k-1}, G_k, P(x(0))) \\ P(x(k)|Z_k, U_{k-1}, S_k, \theta_f, \theta_g, F_{k-1}, G_k, P(x(0))) \end{aligned} \tag{2.4}$$

dove

$$\alpha = \frac{1}{P(z_k|Z_{k-1}, U_{k-1}, S_k, \theta_f, \theta_g, F_{k-1}, G_k, P(x(0)))}$$

è un fattore di normalizzazione. L'ultimo fattore della 2.4 è la pdf di x al tempo k, appena prima che siano effettuate le misurazioni sensoriali, può essere riscritta come

$$Prior(x(k)) \equiv P(x(k)|Z_{k-1}, U_{k-1}, S_k, \theta_f, \theta_g, F_{k-1}, G_k, P(x(0))). \tag{2.5}$$

2.9 Filtri ricorsivi e ipotesi di Markov

Molti algoritmi di filtraggio sono costruiti ricorsivamente per poter assicurare un tempo noto e fisso di computazione [4]. La formulazione ricorsiva del problema descritto nell'equazione 2.3 è possibile per una specifica classe di problemi: i modelli di Markov.

L'ipotesi di Markov stabilisce che $x(k)$ dipende unicamente da $x(k-1)$ (oltre che, naturalmente, da u_{k-1}, θ_f e f_{k-1}) e che $z(k)$ dipende unicamente da $x(k)$ (oltre che, naturalmente, da s_k, θ_g e g). Questo significa che $Post(x(k))$ contiene tutte le informazioni sui dati sensoriali precedenti l'istante temporale corrente [4] [6]. Formalizzando quanto affermato

$$Post(x(k)) \;=\; P(x(k)|z_k, u_{k-1}, s_k, \theta_f, \theta_g, f_{k-1}, g_k, Post(x(k-1))) \qquad (2.6)$$

analogamente dalla 2.4

$$\begin{aligned} Post(x(k)) = \alpha P(z_k|x(k), u_{k-1}s_k, \theta_f, \theta_g, f_{k-1}, g_k, Post(x(k-1))) \qquad (2.7)\\ P(x(k)|u_{k-1}, s_k, \theta_f, \theta_g, f_{k-1}, g_k, Post(x(k-1))) \;= \\ \alpha P(z_k|x(k), s_k, \theta_f, \theta_g, g_k)\; P(x(k)|u_{k-1}, \theta_f, f_{k-1}, Post(x(k-1))) \end{aligned}$$

Tipicamente i filtri di Markov risolvono questa equazione in due passi

1. Aggiornamento del sistema e predizione

$$\begin{aligned} Prior(x(k)) \;&=\; P(x(k)|u_{k-1}, \theta_f, \theta_g, Post(x(k-1))) \\ &= \int P(x(k)|u_{k-1}, \theta_f, f_{k-1}), Post(x(k-1)) \qquad (2.8) \end{aligned}$$

2. Aggiornamento delle osservazioni (fase di correzione)

$$Post(x(k)) \;=\; \alpha\; P(z_k|x(k), s_k, \theta_g, g_k)\; Prior(x(k)). \qquad (2.9)$$

Più semplicemente in una prima fase viene fatta una stima della stato corrente $x(k)$ in funzione dello stato nell'istante precedente, degli input e dei parametri, concordemente al modello del sistema. Successivamente viene stabilità l'attendibilità della stima utilizzando la seconda equazione.

2.10 I filtri particellari

L'idea chiave che sta alla base dei filtri particellari è quella di modellare la pdf dello stato, sicuramente non gaussiana, mediante campionamento. Ossia anzichè descrivere nello spazio continuo degli stati l'andamento di $Post(x(k))$, essa viene descritta attraverso un certo numero di particelle identificabili univocamente dalla coppia $\{x_i, w_i\}$, dove x_i è un vettore contenente la posizione della particella i-esima nello spazio degli stati, w_i invece è il *fattore d'importanza* o peso della particella i-esima. La distribuzione iniziale dei campioni rappresenta la conoscenza iniziale di $P(x(0))$, circa lo stato del sistema dinamico. Per esempio, nel caso di *global localization*, la probabilità dello stato iniziale è rappresentata con un insieme di posizioni generate uniformemente nell'universo del robot e pesate con un fattore $\frac{1}{m}$. Se la posizione iniziale è conosciuta con un piccolo margine di errore, allora essa può essere inizializzata con campioni distribuiti su una gaussiana centrata nella posizione corretta.

I filtri particellari analogamente ai filtri bayesiani, sono basati su una specializzazione dei modelli di Markov: gli *Hidden Markov Models*. Il termine anteposto *Hidden* si riferisce al fatto che lo stato è inosservabile, ovvero nascosto, e vuole enfatizzare come questi modelli vengano utilizzati per stimare lo stato partendo delle uniche grandezze visibili: le percezioni sensoriali e le grandezze interne.

Gli *HMM* aggiungono altre due ipotesi a quelle base:

- Lo spazio degli stati è discreto, ovvero c'è un numero finito di possibili stati inosservabili.
- Lo spazio osservabile è discreto.

La differenza principale rispetto ai modelli di Markov normali è che mentre in quest'ultimi c'è una relazione univoca tra stato e osservazioni, i modelli di Markov *hidden* possiedono un'incertezza intrinseca che viene modellizzata con una densità di probabilità.

I filtri particellari sono ricorsivi con un *loop* divisibile in tre fasi principali

1. Il campione $x_{k-1}^{(i)}$ viene scelto nell'insieme di campioni pesati distribuiti nello spazio inosservabile degli stati secondo $Post(x(k-1))$.

2. $x_k^{(i)}$ viene creato sulla $Post(x(k))$ che possiano porre uguale alla $P(x(k)|x_{k-1}^{(i)}, z_{k-1})$. Indichiamo con

$$qt \equiv P(x_k|x_{k-1}, z_{k-1})\ Post(x_{k-1}) \tag{2.10}$$

la quantità a cui la letteratura si riferisce come *proposal distribution*. Il suo ruolo è quello di proporre campioni secondo la distribuzione $Post$ desiderata, pur non essendovi equivalente.

3. L'ultima fase è quella della correzione del divario fra la distribuzione di probabilità *proposal* e la distribuzione *posterior*, ottenuta cercando di massimizzare la quantità

$$\alpha\ P(z_k|x_k^{(i)})\ P(x_k|x_{k-1}, z_{k-1})\ Post(x_{k-1}) \tag{2.11}$$

dove

$$w^{(i)}\ =\ P(z_k|x_k^{(i)}) \tag{2.12}$$

è il fattore d'importanza ottenuto come quoziente della distribuzione obiettivo e la distribuzione proposta, a meno di un fattore di normalizzazione. Si ottiene

$$\frac{\alpha\ P(z_k|x_k^{(i)})P(x_k^{(i)}|x_{k-1}^{(i)}, z_{k-1})Post(x_{k-1})^{(i)}}{P(x_k^{(i)}|x_{k-1}^{(i)}, z_{k-1})Post(x_{k-1})}\ =\ \alpha\ P(z_k|x_k^{(i)}) \tag{2.13}$$

che, con α costante, è una quantità proporzionale a w_i.

Per i filtri particellari si suppone che le osservazioni condizionate siano indipendenti dallo stato. Se pensiamo di voler stimare la funzione valor medio dell'evoluzione dello stato delle particelle $h(X)$, allora

$$E[h(X)|Post(X(k))] \; = \; \int h(X(k)) \; Post(X(k))dX(k) \tag{2.14}$$

dove $E[f|p]$ denota il valore dell'aspettazione di f rispetto a p e $Post(X(k))$ definita in 2.3. Utilizzando un approccio basato su campionamento, la distribuzione a posteriori della stima attraverso N campioni risulta

$$Post(X(k)) \; \sim \; \frac{1}{N} \sum_{i=1}^{N} \delta_{x_k^{(i)}}(X_k) \tag{2.15}$$

dove $x_k^{(i)}$ corrisponde all'i-esimo campione generato da $Post(X(k))$ e δ denota la funzione di Dirac.

2.10.1 Vantaggi e svantaggi

Recentemente i ricercatori che lavorano nell'ambito della visione artificiale hanno proposto gli algoritmi basati su filtro particellare sotto il nome di algoritmo di condensazione.
Nell'ambito della localizzazione la rappresentazione particellare ha un insieme di caratteristiche che la distinguono dagli approcci precedenti. Di seguito vengono elencati i principali vantaggi offerti dai filtri particellari [6]:

1. Possono utilizzare caratteristiche sensoriali, dinamiche motorie, e distribuzioni di rumore arbitrarie.

2. Sono degli approssimatori universali di densità di probabilità, che riescono a superare le ipotesi molto restrittive sulla forma della *posterior pdf* poste da altri approcci.

3. Focalizzano le risorse computazionali nelle aree più rilevanti, campionando proporzionalmente alla densità a posteriori.

4. Portarli su un programma è piuttosto agevole.

Va, però, specificato che i filtri particellari sono un metodo subottimo dettato dalla difficoltà di trovare delle relazioni che descrivano il sistema in tutte le sue caratteristiche. L'utilizzo di filtri probabilisti puri necessita la conoscenza della forma delle funzioni densità di probabilità, che spesso sono estremamente complesse e definite su un dominio pluridimensionale. perché l'approccio particellare risulti una soluzione valida bisogna considerare che [6] [4]

- La scelta della funzione proposta q_t è cruciale per la generazione delle funzioni di probabilità.

- Un numero elevato di campioni causa un assorbimento maggiore di potenza computazionale.

- I campioni possono degenerare nel caso uno di essi assuma un peso molto maggiore rispetto agli altri. Questo causerebbe una perdita di diversità che indebolirebbe l'algoritmo.

2.10.2 Importance sampling

L'obiettivo di tutti i filtri particellari è di stimare le caratteristiche di $Post(X(k))$ campionandola. Poichè $Post(X(k))$ è sconosciuta (e anche se non lo fosse, sarebbe molto difficile campionarla data la sua complessità), si utilizza il metodo *importance sampling* per approssimarla: sarà il peso dei campioni a caratterizzare le loro differenze. Queso significa che il valore dell'aspettazione descritta in 2.14 diviene

$$\begin{aligned} E[h(X)|Post(X(k))] &= \int h(X(k))\ Post(X(k))dX(k) \\ &= \int h(X(k))\ \frac{Post(X(k))}{Prop(X(k))} Prop(X(k))dX(k) \end{aligned} \tag{2.16}$$

dove $Prop(X(k))$ è la distribuzione *proposal* definita in precedenza.

Il peso del campione i-esimo all'istante k potrà essere espresso come

$$w_i \;=\; \frac{Post(x_k^{(i)})}{Prop(x_k^{(i)})} \tag{2.17}$$

Se consideriamo $w(X(k))$ una funzione di $X(k)$ allora

$$E[h(X)|Post(X(k))] \;=\; \int h(X(k))\; w(X(k))\; Prop(X(k))dX(k) \tag{2.18}$$

Segue che

$$E[h(X)|Post(X(k))] \;=\; \frac{1}{N}\sum_{i=1}^{N} h(x_k^{(i)})\; w(x_k(i)) \tag{2.19}$$

La soluzione ricorsiva al problema si ottiene mediante il *Sequential Importance Sampling*. Indicando con

$$H_{k-1} \;=\; [u_{k-1}, \theta_f, f_{k-1}] \tag{2.20}$$

$$I_k \;=\; [s_k, \theta_g, g_k] \tag{2.21}$$

La 2.3 può essere riscritta come

$$Post(X(k)) = P(X(k)|Post(X(k-1)), H_{k-1}, I_k, z_k) \tag{2.22}$$

Ora, se riscriviamo la distribuzione proposta q_t come

$$q_t = Q(x_k|x_{k-1}, z_k, H_{k-1}, I_k) \tag{2.23}$$

si dimostra che il peso della particella i-esima può essere riscritto come

$$w(x_k^{(i)}) \;=\; \frac{Post(x_k^{(i)})}{Prop(x_k^{(i)})} \;=\; \alpha w(x_{k-1}^{(i)}) \frac{P(z_k|x_k, I_k)P(x_k|x_{k-1}, H_{k-1})}{Q(x_k|x_{k-1}, z_k, H_{k-1}, I_k)} \tag{2.24}$$

dove $\alpha = P(z_k)$. Il fattore di normalizzazione è indipendente dallo stato, per cui può essere portato fuori dal segno d'integrale nella 2.19. Purtroppo, però, è una quantità sconosciuta. Per rimediare questo problema si utilizzano pesi normalizzati:

$$\tilde{w}(x_k^{(i)}) \;=\; \frac{w(x_k^{(i)})}{\sum_{i=1}^{N} w(x_k^{(i)})} \tag{2.25}$$

2.10.3 Resampling

Nel seguito della trattazione l'algoritmo di localizzazione verrà spiegato approfonditamente, comunque è necessario introdurre il concetto di ricampionamento o *resampling*. E' vero, infatti, che il peso delle particelle può essere utilizzato per calcolare la stima dello stato, per cui quelle troppo leggere influirebbero in minima parte. E' altrettanto vero che prendere in considerazione particelle oramai trascurabili è sconveniente: anzichè aggiornare ad ogni ciclo le caratteristiche legate a quella particella, se ne potrebbe creare un'altra che si trovi in uno stato più probabile.

La generazione di nuovi campioni, eventualmente sostitutivi ad altri, viene detta ricampionamento ed è fondamentale per massimizzare le probabilità di successo e l'utilizzo delle risorse.
La filosofia guida del *resampling* stabilisce che le particelle degeneri, ovvero con peso molto basso, debbano essere scartate, mentre quelle più pesanti debbano moltiplicarsi. Inevitabilmente dopo il ricampionamento ci saranno particelle con le stesse coordinate di stato, ma il modello di sistema farà si che la loro evoluzione le porti verso strade diverse: il rumore gaussiano presente nelle equazioni del modello si traduce in scostamenti anche significativi nel tempo dal percorso ideale.

Ogni N cicli dell'algoritmo di stima dello stato deve essere eseguito il ricampionamento per depurare l'insieme delle particelle da quelle meno probabili. Queste

verranno sostituite da altre più probabili e che, quindi, potranno aiutare le stime successive.

Questo tipo di procedura non è lontana dal *tracking* multiipotesi: ogni particella può essere vista come un'ipotesi che si aggiorna ad ogni ciclo. Il creatore di nuove ipotesi è il rumore, che devia il percorso da quello ideale secondo una distribuzione di probabilità gaussiana. Il peso delle particelle funge da decisore.

Lo stato stimato equivarrà alla media pesata degli stati campionati.

2.10.4 Metodo Monte Carlo

Attraverso i metodi Monte Carlo vengono risolti problemi fisici e matematici utilizzando generatori di numeri casuali. Nel nostro caso, essi permettono la realizzazione dell'*importance sampling* attraverso la stima della quantità

$$I = \int h(x)p(x)dx \tag{2.26}$$

Da notare che 2.26 è simile alla 2.19. Questa equazione è facilmente risolvibile una volta che si è in grado di campionare la $p(x)$:

$$I \sim \sum_{i=1}^{N} h(x^{(i)}) \tag{2.27}$$

La relazione sopra è dimostrabile con $x^{(i)}$ campione della $p(x)$. Partendo da un insieme di campioni pesati, deve essere costruito un vettore che associ ad ogni campione una grandezza ottenuta da quella che viene denominata in letteratura funzione di distribuzione cumulativa (*Cumulative distribution function (CDF)*)[4]. Se si pensa che l'insieme dei campioni possa essere contenuto in un vettore, allora sarà possibile costruire un nuovo vettore Cdf cui

$$Cdf^{(i)} \; = \; CDF(x^{(i)}) \tag{2.28}$$

Cdf sarà utilizzato per effettuare il ricampionamento Nella sezione sottostante viene descritto un metodo di *importance resampling* Monte Carlo con utilizzo della funzione di distribuzione cumulativa.
Se la densità di probilità $p(x)$ è una gaussiana centrata in $x = 0$, allora i campioni si distribuiscono sullo spazio degli stati, che qui rappresentiamo come unidimensionale, come mostrato in figura 2.3.

```
Costruisci la distribuzione cumulativa(CDF)
Preleva N campioni u[i] (1 <i <= N) da
una densità uniforme U[0,1]
Esplora la PDF cumulativa
for i:=1 a N
  j:=0
  while u[i] > CDF(x[j]) do
    j++
  end while
  Aggiungi x[j] alla lista dei campioni.
end for
```

Listato 2.1: Importance resampling con il metodo Monte Carlo.

I campioni scelti tendono a essere quelli più pesanti: dove, infatti, il peso è maggiore, l'intervallo su cui può cadere l'i-esimo numero *random* è più largo.
L'utilizzo di queste tecniche permette di simulare meglio l'aleatorietà dei sistemi utilizzando strumenti disponibili in un calcolatore elettronico.

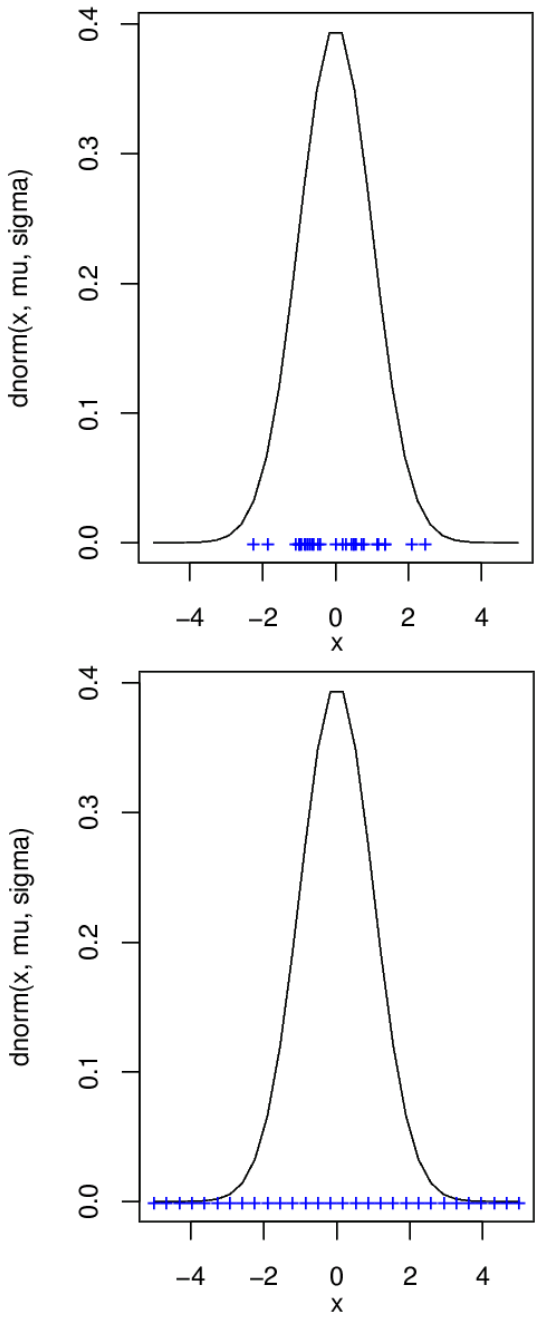

Figura 2.3: Importance Sampling e Uniform Sampling.

Capitolo 3

Progettazione del sistema

3.1 La mappa

Come è stato sottolineato nel capitolo precedente, la mappa su cui lavora l'algoritmo di localizzazione è di tipo metrico. Questo significa che l'utente deve fornire una planimetria dell'ambiente in cui il robot si deve muovere che sia più accurata possibile (fig.2.2). Per permettere al programma di elaborarla con semplicità, il formato elettronico dovrà essere unico e l'utente dovrà indicare le zone accessibili al robot e quelle inaccessibili.

Partendo dalla planimetria deve essere realizzata l'*occupancy grid*. Ogni cella della mappa che si vorrà ottenere sarà settata a "libera" nel caso in cui lo spazio da essa rappresentato sia totalmente accessibile, "occupata" in caso contrario. Il numero di pixel della planimetria contenuti in una cella definirà la risoluzione della mappa. In figura 3.1 è stata rappresentata la fase di creazione di una *occupancy grid* su una parte della pianta dell'ambiente.

I motivi per cui si rende necessaria una rappresentazione dello spazio sono almeno tre:

- come già sottolineato, serve a capire quali sono le parti dell'ambiente libere per la navigazione, ovvero a identificare quello che in letteratura viene definito *free space*,
- è fondamentale per riconoscere i luoghi appartenenti l'ambiente,

- può essere utilizzata anche per il riconoscimento di oggetti.

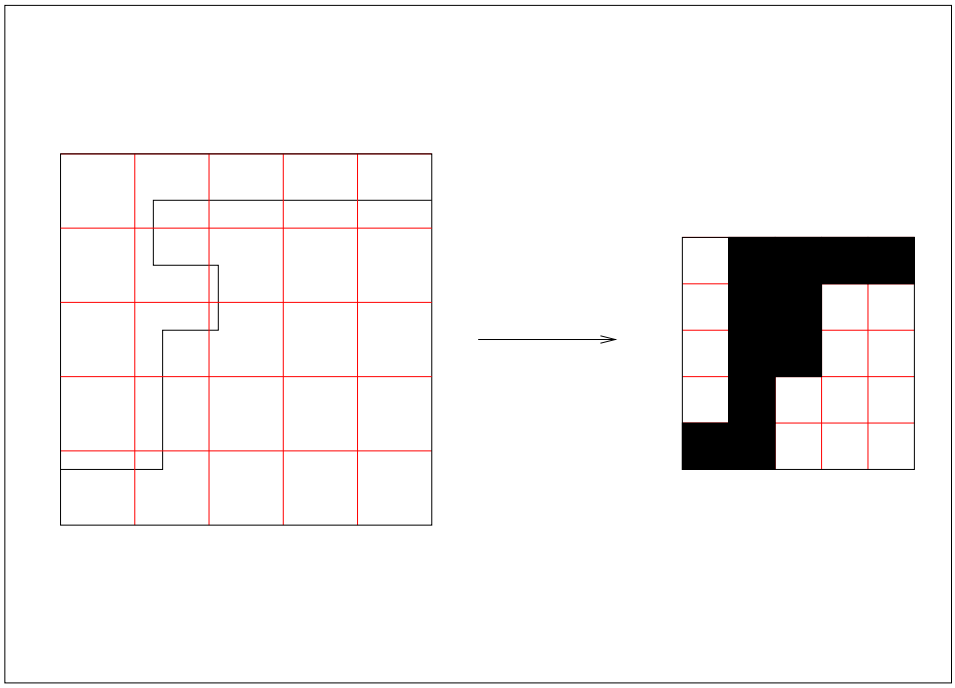

Figura 3.1: Creazione dell'*occupancy Grid*.

Come si vedrà in seguito, nella fase di sperimentazione è stato utilizzato un simulatore. Quest'ultimo utilizza solamente file .pnm in scala di grigi e bicolore per importare la pianta degli ambienti. Si è deciso, per ragioni di uniformità, di utilizzare questo formato anche nel software di localizzazione rappresentando in nero il *free space* e in bianco lo spazio non navigabile.

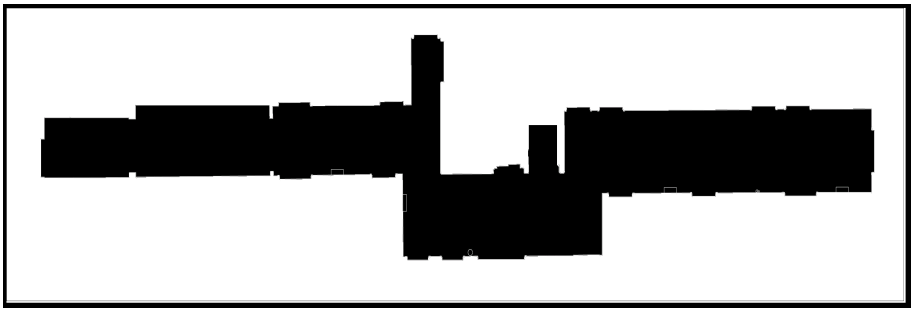

Figura 3.2: Planimetria dell'ambiente di sperimentazione: il *free space* è lo spazio nero.

3.2 I dati sensoriali

Già nel precedente capitolo sono state distinte le informazioni sensoriali ”interne” da quelle ”esterne”. Le prime riguardano l’odometria del robot, mentre le seconde sono le informazioni restituite da dispositivi di prossimità.

3.2.1 I sensori di prossimità

Nel capitolo 5° sono riportate in modo più completo le informazioni riguardo Nomad 200, il robot utilizzato per sperimentare il sistema di localizzazione. Le informazioni sensoriali elaborate sono quelle fornite dai suoi 16 dispositivi sonar ultrasonici *Polaroid* e i suoi 16 sensori a infrarossi.

I sonar forniscono la misura della distanza dagli oggetti basandosi sul tempo impiegato da un impulso ultrasonico per arrivare ad un ostacolo e ritornare al dispositivo che lo ha emesso. Il problema che nasce quando si utilizza una fonte sensoriale di questo tipo è la sua rumorosità [7][8]:

- I sensori ultrasonici non hanno una buona direzionalità. Questo limita l’accuratezza nella determinazione della posizione spaziale dell’oggetto rilevato a 10-50 cm, anche dipendentemente dalla distanza dall’ostacolo e dall’inclinazione della superficie colpita rispetto al fronte dell’onda sonora.

- Le misure delle distanze rilevate dai sensori ultrasonici sono seriamente corrotte da riflessioni.

- ”..l’uso dei sonar come rilevatori di distanze rappresenta talvolta il peggior caso per la localizzazione mediante utilizzo di distanze come dati.”(M. Drumheller. Mobile robot localization using sonar. Marzo 1987))

L’intervallo di sparo, ovvero il tempo che intercorre tra due emissioni consecutive di un impulso da parte dello stesso dispositivo, può variare da un minimo di 4 ms ad un massimo di 1 s; il valore minimo consente di individuare ostacoli ad una distanza massima di circa 70 cm. In realtà non verrà usata la frequenza di sparo massima, per cui è possibile misurare distanze superiori al metro.

Se i sensori sonar non sono molto affidabili per distanze elevate, diventano del tutto inaffidabili per piccole distanze, in quanto le rilevazioni vengono corrotte da

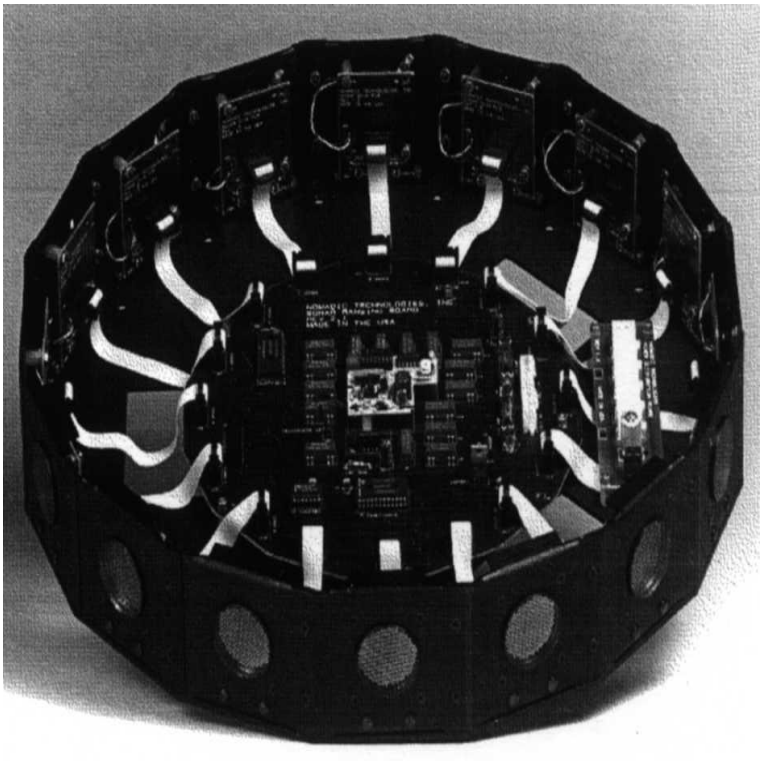

Figura 3.3: Sensori ultrasonici utilizzati su Nomad 200.

fenomeni di risonanza.
I dispositivi a radiazione infrarossa non hanno un raggio d'azione elevato come quello dei sensori ultrasonici, ma sono più affidabili sulle distanze limitate. Il loro funzionamento è simile a quello dei sonar: si basano anch'essi sul *time of flight* dell'impulso a radiazione infrarossa da essi emanato.
L'utilizzo combinato di queste due fonti sensoriali è il mezzo per ottenere non solo la distanza da un oggetto, ma anche la direzione in cui si trova rispetto al robot. Considerando infatti che sia nel caso dei sonar, sia in quello dei sensori a radiazioni infrarosse, l'angolo che separa gli impulsi provenienti da dispositivi adiacenti è di 22.5°, risulta possibile calcolare l'inclinazione del segmento congiungente il robot con l'ostacolo, rispetto al sistema di riferimento del robot stesso. 22,5° è anche il valore dell'errore massimo teorico che si può compiere nel calcolo dell'orientamento del *link* robot-ostacolo.

I sensori di prossimità sono quelli che forniscono al sistema di localizzazione

le variabili osservabili: il problema sta nel trasformare 16 distanze in qualcosa di importabile da un un modello matematico.

3.2.2 Odometria

L'odometria rappresenta quel'insieme di informazioni provenienti dagli organi dedicati al movimento del robot. Velocità di traslazione, velocità di rotazione, stallo dei motori e stato di carica delle batterie sono le informazioni interne di base per un robot mobile. Nel costruire un modello dinamico nell'ambito della localizzazione di solito si utilizzano le due velocità appena citate come ingressi del sistema. La sua parte controllabile e quella osservabile saranno ricavabili da odometria e sensori.

Gli unici input provenienti dal robot che il software utilizzerà saranno velocità di traslazione e rotazione, nonchè le misure dei sonar e degli infrarossi. I 32 dati provenienti dai sensori di prossimità verranno tradotti in due dati: distanza dall'ostacolo e direzione in cui si trova l'ostacolo ripetto al robot.

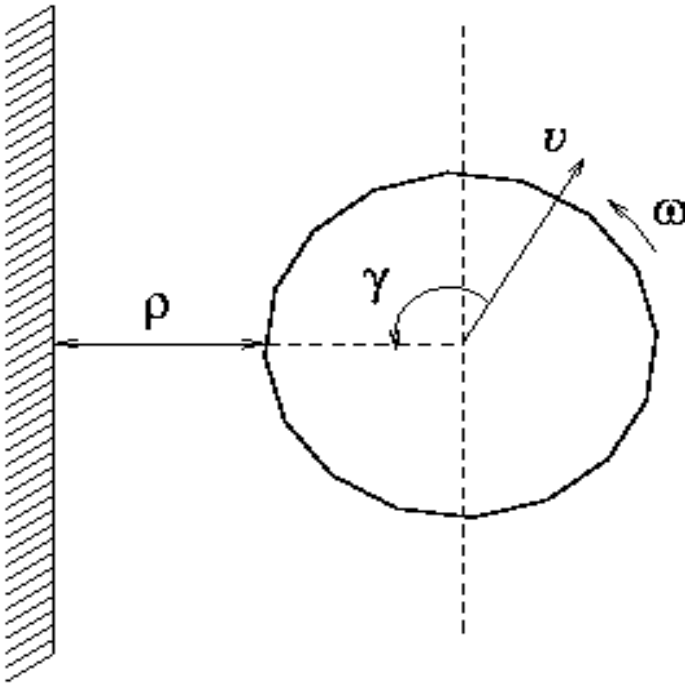

Figura 3.4: Grandezze utilizzate dal sistema di localizzazione: velocità di traslazione ν, velocità di rotazione ω, distanza dall'ostacolo ρ e direzione in cui si trova l'ostacolo rispetto al robot γ.

3.3 Il modello di sistema

Identificare il modello del sistema implica definire la relazione fra l'evoluzione dello stato ed i controlli, come già sottolineato nell'equazione 2.1. Lo stato del robot è identificabile univocamente mediante la tripla $\{x, y, \theta\}$, dove x e y sono la posizione del robot su un sistema di riferimento cartesiano, mentre θ è il suo orientamento.

Utilizzando semplici principi di cinematica non è difficile trovare 3 relazioni che descrivano l'evoluzione nel tempo di queste tre grandezze in un caso ideale:

$$\begin{aligned} x(k) &= x(k-1) + \nu(k-1) * cos(\theta(k-1))\delta t \\ y(k) &= y(k-1) + \nu(k-1) * sen(\theta(k-1))\delta t \\ \theta(k) &= \theta(k-1) + \omega(k-1) * \delta t \end{aligned} \tag{3.1}$$

Il problema è che le velocità traslazionale e rotazionale ν e ω sono affette da numerose imprecisioni dovute, ad esempio, all'attrito irregolare delle ruote sul pavimento, a imprecisioni nelle misurazioni o ad asperità incontrate durante il moto. Inoltre un modello di questo tipo suppone che le grandezze in gioco rimangano costanti per l'intervallo δt. Questo non significa che il modello sia errato, ma va corretto aggiungendo un termine di rumore e delle ipotesi:

- L'intervallo di tempo δt deve essere abbastanza piccolo da rendere trascurabile la differenza $|\theta(k) - \theta(k-1)|$ ai fini dell'aggiornamento delle variabili di stato $x(k)$ e $y(k)$
- Gli elementi di disturbo devono essere quantificabili e non troppo variabili nello spazio degli stati.
- Eventuali errori sistematici che affliggono l'odometria del robot devono essere conosciuti.

Se non si tenesse conto del rumore che affligge il sistema, dopo poche iterazioni i risultati ritornati dalle equazioni sarebbero estremamente incongruenti rispetto al

caso reale. Se indichiamo con

$$A(k) = A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$B(k) = \begin{pmatrix} cos(\theta(k-1))\delta t & 0 \\ sen(\theta(k-1))\delta t & 0 \\ 0 & \delta t \end{pmatrix}$$

il modello di sistema risulta descritto dalla seguente relazione

$$X(k) \; = \; A\, X(k-1) + B(k)U(k-1) + N(k) \tag{3.2}$$

con $X^T(k) = (x(k), y(k), \theta(k))$ e $U^T(k) = (\nu(k), \omega(k))$. Il rumore N sarà supposto gaussiano bianco con media μ e varianza σ.

Il modello di sistema così costruito è lineare e affetto da incertezza gaussiana. Quest'ultima ipotesi sul rumore è determinata dal fatto che utilizzando un filtro bayesiano, i rumori vengano supposti sempre gaussiani.

3.4 Il modello percettivo

Il modello percettivo o modello delle osservazioni lega la parte osservabile del sistema al suo stato. Mentre inscrivere l'evoluzione dello stato del robot in un modello lineare con rumore gaussiano è stato semplice e quasi obbligato, ben più difficile è trovare un'equazione che leghi le osservazioni alla posizione e all'orientamento del robot.

Indichiamo con $Z(k)$ la coppia $(\rho(k), \gamma(k))$, rispettivamente la distanza e la direzione in cui si trova l'oggetto più vicino al robot, rispetto al robot stesso (fig.3.4). Per trovare l'espressione analitica del modello percettivo sarebbe necessario conoscere una funzione $h : \Re^3 \rightarrow \Re^2$ che permetta di stabilire dove sarebbe l'oggetto più vicino e quanto disterebbe se il robot fosse in una determinata posizione, con un determinato orientamento.

Il fatto che, però, l'insieme degli stati sia chiuso e limitato, permette di creare

una struttura dati derivata dall'*occupancy map* che contenga i valori, calcolati mediante algoritmo, assunti da $h(X) \; \forall \; X \; \subset$ *free space*. L'idea è quella di creare altre due mappe, rappresentabili come due matrici della stessa dimensione della prima. In questo caso ogni cella non assumerà unicamente lo stato libero-occupato, poichè ad essa sarà associato un valore che indica, nel caso di una mappa, il valore di $\rho(x, y)$, e, nel caso dell'altra, il valore di $\gamma(x, y)$.

La creazione di queste due mappe può essere effettuata totalmente *off-line* e consultata dall'algoritmo di localizzazione ogni volta che sia necessario. Per convenzione gli angoli saranno compresi tra $-\pi$ e π.

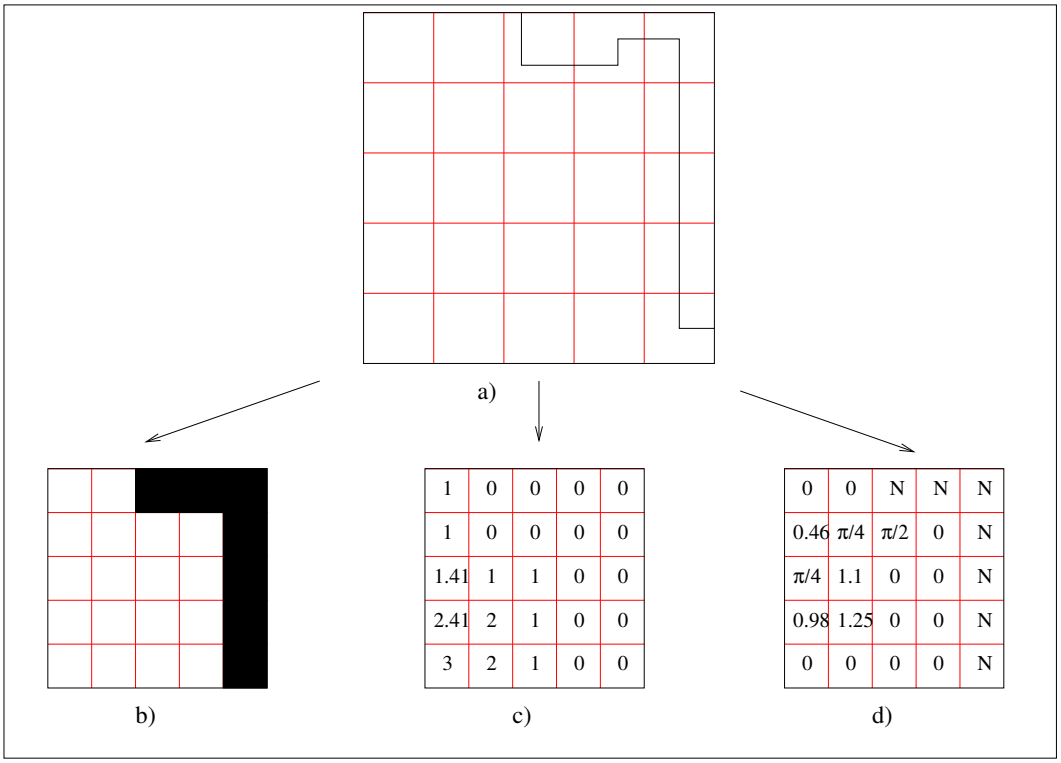

Figura 3.5: a)Stralcio di planimetria b)*Occupancy grid* c)Mappa delle distanze d)Mappa angolare.

3.5 Bootstrap filter

Dopo aver costruito i modelli di sistema e percettivo è possibile realizzare il filtro particellare adatto all'applicazione. Nel capitolo precedente l'argomento è sta-

to ampiamente introdotto. Non è stata, però, specificata la forma della densità di probabilità proposta $q_t = Prop(x(k)) = Q(x_k|x_{k-1}, z_k, H_{k-1}, I_k)$.

La funzione densità $Q(...)$ è quella che contraddistingue la potenza e la precisione dei filtri particellari: da essa, infatti, dipende la varianza dei pesi delle particelle, come già visto nell'equazione 2.25.

La soluzione ottima al problema sarebbe trovare una distribuzione proposta di questo tipo

$$Prop(x_k^{(i)}) = Q(x_k|x_{k-1^{(i)}}, z_k, H_{k-1}, I_k) = P(x_k|{x_{k-1}}^{(i)}, z_k, H_{k-1}, I_k) \quad (3.3)$$

Operativamente la distribuzione proposta permette di prevedere lo stato successivo partendo dallo stato attuale e di individuare la traiettoria descritta da una particella nello spazio degli stati. Avere a disposizione una stima che tiene conto non solo dei controlli ma anche delle osservazioni, aumenterebbe molto le probabilità di successo. Una funzione $Prop(x_k^{(i)})$ di questo tipo è tanto potente quanto difficile da creare: trovare una funzione matematica che intersechi parte osservabile e controllabile di un sistema è assai complesso.

La soluzione, come per il modello percettivo, potrebbe essere algoritmica. In questo caso, però, l'elaborazione non potrebbe essere *off-line*, in quanto le stime di posizione sono uno degli *step* del ciclo di localizzazione, per cui devono essere effettuate frequentemente e in tempo reale.

La scelta che è stata fatta è quella di un filtro particellare di tipo *bootstrap*. Questi filtri utilizzano una *proposal density* di questo tipo:

$$Prop(x_k^{(i)}) \;=\; Q(x_k|{x_{k-1}}^{(i)}, z_k, H_{k-1}, I_k) \;=\; P(x_k|x_{k-1}^{(i)}, H_{k-1}) \;=\; Prior(x_k^{(i)}) \quad (3.4)$$

Dalla equazione 2.25 si ottiene

$$w(x_k^{(i)}) \;=\; \frac{Post(x_k^{(i)})}{Prop(x_k^{(i)})} \;=\; \alpha w({x_{k-1}}^{(i)})\; P(z_k|x_k, I_k) \quad (3.5)$$

La funzione $Prior(x_k^{(i)})$ è definita dal modello di sistema: il rumore gaussiano bianco fa sì che l'equazione 3.2 definisca statisticamente l'evoluzione dello stato.

Il *bootstrap filter* senz'altro ha il difetto di non includere le osservazioni nel-

la densità proposta, e quindi di perdere un buon *feedback* nelle stime. Vi è però il vantaggio che sia la predizione dello stato che il pesatura delle particelle sono molto semplificati. Questo implica una maggior reattività e velocità nel compiere i passi del ciclo di localizzazione. Non si deve, infatti, dimenticare che l'algoritmo costruito dovrà essere eseguito da un robot che avrà potenza computazionale limitata.

Per quanto riguarda l'assegnazione dei pesi, la quantità $P(z_k|x_k, I_k)$ è ricavabile, analogamente alla $P(x_k|x_{k-1}^{(i)}, H_{k-1})$, dal modello percettivo. Questi concetti verranno ripresi e approfonditi nelle sezioni successive.

3.6 Algoritmo di localizzazione

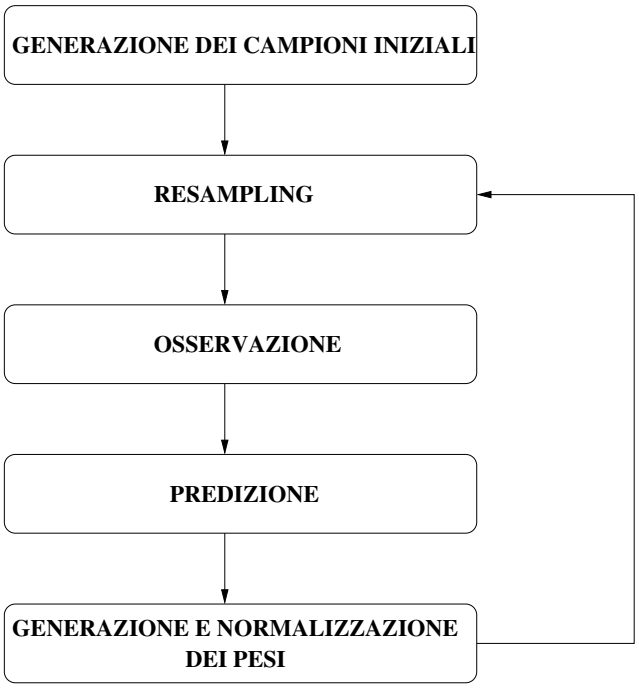

Figura 3.6: Schema dell'algoritmo.

In figura 3.6 è stato rappresentato il ciclo di localizzazione basato su filtro particellare *bootstrap*. Si suppone che le mappe *occupancy grid*, delle distanze e degli angoli siano già state create.

Di seguito verranno descritte tutte le fasi dell'algoritmo, che verranno realizzate successivamente sul sistema costruito.

3.6.1 Generazione dei campioni iniziali

La fase iniziale del ciclo è basata sulla prima ipotesi a cui fanno riferimento i filtri bayesiani: la conoscenza della $P(X(0))$: è necessario conoscere la forma della distribuzione di probabilità iniziale dello stato.

Se la localizzazione è di tipo globale, allora la posizione del robot è totalmente sconosciuta. In questo caso la $P(X(0))$ sarà una densità di probabilità uniforme. Campionandola le particelle ottenute si dovrebbero distribuire uniformemente sul *free space*. Nel caso, invece, sia conosciuta la posizione iniziale del robot, è conveniente utilizzare come $P(X(0))$ una funzione gaussiana centrata nel punto in cui il robot è posizionato e con una varianza dipendente dall'errore da cui è affetta la misurazione. Anche se si suppone che non ci siano errori di misura (cosa questa assai improbabile), non conviene generare tutte le particelle iniziali nello stesso punto del *free space*. Una delle condizioni che, infatti, permettono un buon funzionamento dell'algoritmo, è il mantenimento della diversità.

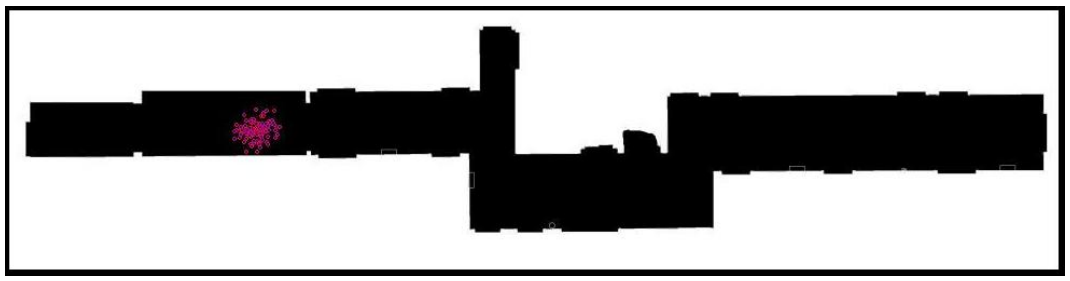

Figura 3.7: Campioni iniziali nel caso di P(X(0)) gaussiana.

3.6.2 Osservazione

La fase di osservazione è quella in cui il robot carica i valori provenienti dai suoi sensori e fornisce la variabile $Z(k)$, come già mostrato in precedenza. Per osservazione s'intende la misura della distanza tra il robot e l'oggetto più vicino e dell'an-

golo compreso tra l'asse X del sistema di riferimento cartesiano della planimetria e la congiungente il robot con l'ostacolo (fig.3.4).

Si definiscano d_i e l_i con $0 \leq i \ < 16$ le misure di distanza rilevate rispettivamente dai sensori ultrasonici e dai dispositivi a radiazione infrarossa.
Sia

$$Min_{sonar} \ = \ min\{d_0...d_n\} \ con \ n \ = \ 15$$

la distanza dall'oggetto più vicino rilevata dai sonar.
Sia

$$Min_{ired} \ = \ min\{l_0...l_n\} \ con \ n \ = \ 15$$

la distanza dall'oggetto più vicino rilevata dagli infrarossi.

Ipotizzando che entrambi i valori siano attendibili, ovvero che siano compresi nell'intervallo delle distanze rilevabili in modo affidabile dai dispositivi, si ottiene che

$$\rho \ = \ min\{Min_{sonar}, Min_{ired}\} \tag{3.6}$$

Considerando che ogni sensore di prossimità è separato da quello adiacente da un angolo di $\pi/8$ rad, sia i l'indice del sensore che rileva la distanza minore, allora il secondo elemento del vettore osservazione sarà ricavato dalla seguente espressione:

$$\gamma \ = \ \begin{cases} i * \frac{\pi}{8}, \ i \leq 8 \\ i * \frac{\pi}{8}, \ i > 8 \end{cases} \tag{3.7}$$

La distinzione tra i primi nove sensori e gli altri è dovuta al fatto che gli angoli devono essere compresi tra $-\pi$ e π: i sensori sulla parte sinistra del robot (indice tra 0 e 8) spaziano angoli tra 0 e π, mentre quelli sulla parte destra (indice tra 9 e 15) spaziano angoli tra $-\pi$ e 0. Come sarà evidenziato trattando lo *step* del pesatura delle particelle, il vettore $z_k \ = \ (\rho, \gamma)$ verrà utilizzato per il calcolo di $P(z_k|x_k)$ con l'aiuto del modello percettivo a mappe.

3.6.3 Predizione

Come era stato anticipato nel capitolo precedente, i filtri particellari, analogamente a tutti i filtri ricorsivi, hanno la proprietà di aggiornarsi con frequenza costante. Que-

sto significa che due stati stimati consecutivamente saranno separati da un intervallo di tempo fisso.

Durante la fase di predizione l'algoritmo tenta di stimare lo stato corrente partendo dallo stato e dai controlli effettuati nell'istante di tempo discreto precedente:

$$x_k \sim P(x_k|x_{k-1}, H_{k-1})$$

Il simbolo "$\sim$" va inteso come "campionato da". Nel *bootstrap filter* è il modello di sistema a generare le predizioni.Questa fase è totalmente *dead reckoning*, ovvero basata unicamente sull'odometria. E' fondamentale individuare eventuali errori sistematici (ad esempio se il robot ha una velocità superiore a quella che i suoi sensori odometrici rilevano, oppure se navigando tende a orientarsi verso un lato, etc.) e modellizzarli attraverso il rumore gaussiano che è presente nelle equazioni di moto del robot.

Nella fase di predizione ogni particella del filtro si sposta concordemente alla $P(x_k|x_{k-1}, H_{k-1})$, influenzata dal rumore gaussiano che funge da generatore di diversità: particelle nella stessa posizione potranno assumere due posizioni diverse nell'istante di tempo successivo (*multihypothesis tracking*).

Se indichiamo con $x_{est}^{(i)}$ la stima di stato per la particella i-esima:

$$x_{est}^{(i)} = x_{k-1}^{(i)} + B(k)U(K-1) + \eta \tag{3.8}$$

con $\eta = (\eta_x, \eta_y, \eta_\theta)$ vettore contenente la media del rumore gaussiano sulle tre componenti dello stato, allora si ottiene

$$P(x_k|x_{k-1}^{(i)}, H_{k-1}) \;=\; P(x_k|x_{est^{(i)}}) \;=\; \frac{1}{\sqrt{2\pi\sigma}}\, e^{\dfrac{-\,(x_k - x_{est}^{(i)})^2}{2\sigma^2}} \tag{3.9}$$

con $\sigma = (\sigma_x, \sigma_y, \sigma_\theta)$.

In figura 3.8 viene mostrata la stima della posizione che assumerà una particella che si muove con velocità ν e orientamento θ. La particella i-esima tenderà a spostarsi verso il centro delle curve di livello circolari della funzione 3.9.

Si è soliti dividere il rumore sull'odometria in due componenti: l'errore traslazionale e quello rotazionale. Moltiplicando il primo per il coseno e per il seno

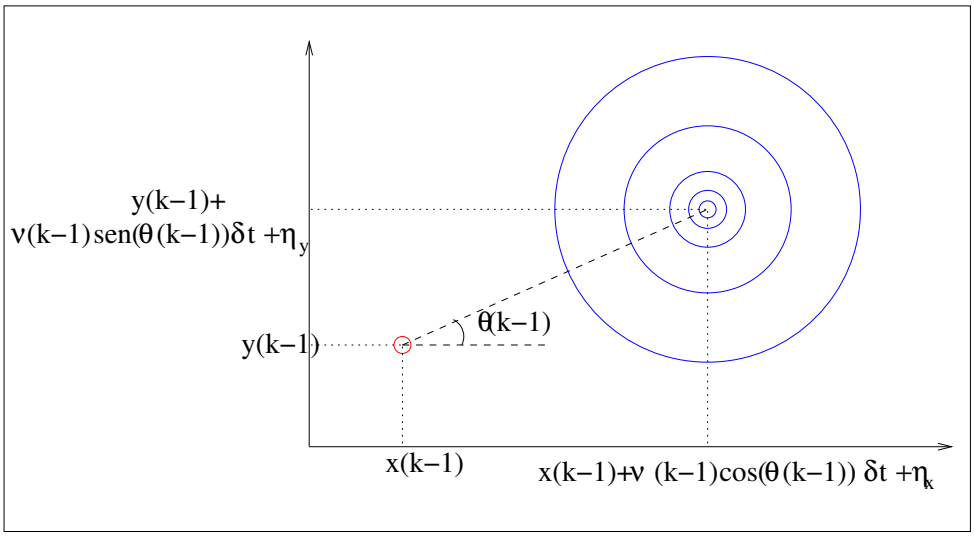

Figura 3.8: Fase di predizione: in blu sono rappresentate le curve di livello della $p(x_k|x_{k-1}, H_{k-1})$.

dell'orientamento, si ottengono rispettivamente il rumore sulla componente x dello stato e sulla componente y. L'errore di stima dell'orientamento è dato dal rumore rotazionale. Poste queste premesse, viene presentato l'algoritmo di predizione nel listato 2.1.

3.6.4 Generazione e normalizzazione dei pesi

Il peso di una particella indica quanto questa rappresenta una stima attendibile dello stato. Il suo valore dipende da quello che il robot percepisce: se le sue osservazioni sono coerenti con la posizione del campione allora il peso sarà alto, viceversa scenderà seguendo il profilo di una gaussiana generata sul sottospazio osservabile.

$$w(x_k^{(i)}) \;=\; \frac{Post(x_k^{(i)})}{Prop(x_k^{(i)})} \;=\; \alpha w({x_{k-1}}^{(i)})\; P(z_k|x_k^{(i)}, I_k)$$

Esprimiamo il modello percettivo mediante la funzione

$$d(k) = z(k) - z_{est}(k) \tag{3.10}$$

```
for j=1 to num_particelle do
   for i=1 to K do
      Err_trasl=random(M_trs* d_l, var_trs*d_l);
      Err_rot=random(M_rot*d_l, var_rot*d_l);
      theta[j]=theta[j] + Err_trasl;
      x[j]=x[j]+(d_l+Err_trasl*cos(theta[j]);
      y[j]=y[j]+(d_l+Err_trasl*sin(theta[j]);
      Err_rot=random(M_rot*d_l, var_rot*d_l);
      theta[j]=theta[j]+Err_rot;
   end for
   Particella[j]=(x[j],y[j],theta[j]);
end for
Return Particella;
```

Listato 3.1: Algoritmo di predizione dello stato

dove $z(k)$ contiene le osservazioni all'istante k e $z_{est}(k)$ la stima effettuata partendo dalla predizione effettuata basandosi sul modello di sistema. Indichiamo con $z_{est}^{(i)} = (\rho_{est}^{(i)}, \gamma_{est}^{(i)})$ la stima delle osservazioni per la particella i-esima. Identifichiamo con M_{dist} la mappa delle distanze e con M_{ang} la mappa angolare allora

$$z_{est}^{(i)} \sim z_{est(k)} \tag{3.11}$$

$$\rho_{est}^{(i)} = M_{dist}(x^{(i)}, y^{(i)}) \tag{3.12}$$

$$\gamma_{map}^{(i)} = M_{ang}(x^{(i)}, y^{(i)}) + \theta^{(i)} \tag{3.13}$$

con $x^{(i)}, y^{(i)}, \theta^{(i)}$ coordinate di stato del campione i-esimo. La stima dell'angolo non è corretta se non si tiene conto dell'orientamento del robot. Per questo motivo nel calcolare γ_{map} si è sommato $\theta^{(i)}$. Gli angoli, però, devono essere compresi tra $-\pi$ e π. Per ottenere una stima corretta deve essere compiuto un ulteriore calcolo:

$$\gamma_{est}^{(i)} = \begin{cases} \gamma_{map}^{(i)} \ , & se -\pi < \gamma_{map}^{(i)} < \pi \\ -2\pi + \gamma_{map}^{(i)} \ , & se\ \pi < \gamma_{map} < 2\pi \\ 2\pi + \gamma_{map}^{(i)} \ , & se\ -\pi < \gamma_{map} < -2\pi \end{cases} \tag{3.14}$$

Sia

$$d_k^{(i)} \sim d(k) = z_k - z_{est}^{(i)} \tag{3.15}$$

la differenza tra osservazione stimata e reale per il campione i-esimo. Il caso ottimo si avrebbe se questa quantità fosse uguale a 0. Come il modello di sistema, però, anche il modello percettivo contiene rumore che dipende da diversi fattori:

- Sia i sonar che gli infrarossi introducono un errore angolare che può arrivare a 22,5°, angolo compreso tra due dispositivi adiacenti rispetto al centro del robot.
- Gli impulsi ultrasonici emessi dai sonar sono affetti da riflessioni e risonanza.
- Quando il tempo di volo degli impulsi diventa elevato la precisione diminuisce. Anche i sonar hanno una capacità limitata in termini di distanza.
- Oggetti scuri causano errori di misurazione da parte dei dispositivi a radiazione infrarossa.

Tutti questi fattori devono essere presi in considerazione per determinare la media e la varianza del rumore supposto gaussiano che affligge le misure. L'espressione del peso della particella i-esima diviene

$$w(x_k^{(i)}) = \alpha w({x_{k-1}}^{(i)}) \frac{1}{\sqrt{2\pi\sigma_\rho}} e^{\frac{-(\rho_k - \rho_{est}^{(i)})^2}{2\sigma_\rho^2}} \frac{1}{\sqrt{2\pi\sigma_\gamma}} e^{\frac{-(\gamma_k - \gamma_{est}^{(i)})^2}{2\sigma_\gamma^2}} = \alpha w({x_{k-1}}^{(i)}) P(d_k^{(i)} = 0) \quad (3.16)$$

con $\alpha = \dfrac{1}{\sum_{i=1}^{N} w(x_k^{(i)})}$.

La densità di probabilità $P(d_k^{(i)} = 0) = P(z_k^{(i)}|P(z_k|x_k^{(i)}, I_k)$ è, dunque, approssimata a una gaussiana a media nulla e varianza $\sigma_d = (\sigma_\rho, \sigma_\gamma)$ equivalente a quella del rumore. La stima dello stato del robot viene effettuata mediante media pesata delle coordinate di stato dei campioni:

$$X(k) = \sum_{i=1}^{N} \frac{x_k^{(i)} w(x_k^{(i)})}{\sum_{i=1}^{N} w(x_k^{(i)})} \quad (3.17)$$

3.6.5 Resampling

Il ricampionamento è un'operazione necessaria per la buona riuscita dell'algoritmo di localizzazione principalmente per due motivi:

1. Ad ogni ciclo il peso del campione viene moltiplicato per il valore che aveva l'istante precedente. Dato che i valori in gioco sono compresi tra 0 e 1, $w(x_k^{(i)}) \sim 10^{-500}$ in poche iterazioni. Nemmeno un tipo come il double di C++ è adatto per trattare numeri così piccoli.

2. Le particelle "improbabili" non devono assorbire potenza computazionale.

Come già sottolineato in precedenza, l'idea chiave del *resampling* è quella di eliminare le particelle più leggere rimpiazzandole con altre. Quest'ultime dovranno essere generate dai campioni più pesanti e, quindi, con probabilità di successo maggiore. L'algoritmo di ricampionamento potrebbe avere una forma di questo tipo:

```
w_min = soglia di peso
for i:=0 to numero_particelle
  if w(particella[i])<w_min begin
    elimina particella[i]
    raddoppia la particella più pesante
  end
```

Listato 3.2: Resampling

Questo metodo di ricampionamento ha il difetto principale di non mantenere una certa diversità e di essere troppo deterministico. Inoltre ha un'elevata complessità computazionale perché la lista delle particelle deve essere sempre esplorata tutta.

Nel capitolo precedente è stato illustrato l'algoritmo di *importance resampling* Monte Carlo. Qui viene presentato un metodo simile, ma più efficiente, che verrà utilizzato per costruire N campioni con un algoritmo che presenta una complessità computazionale del tipo $O(N)$. [4].

La notazione pow(a,b) presente nel listato 3.3 riprende un comando della libreria matematica C e sta per a^b.

```
Costruisci la distribuzione cumulativa(CDF)
Prendi la radice n-esima di u[N] tale che
u[N]=pow(u[N],1/N)
Preleva N campioni u[i] (1 <i <= N) da
una densità uniforme U[0,1]
u[N]=pow(u[N],1/N)
for i=N-1 to 1 do
  Riscala il campione: u[i]=pow(u[i],1.0/i)*u[i]+1
end for
Esplora la pdf cumulativa
for i:=1 a N
  j:=0
  while u[i] > CDF(x[j]) do
    j++
  end while
  Aggiungi x[j] alla lista dei campioni.
end for
```

Listato 3.3: Ordered resampling con metodo Monte Carlo.

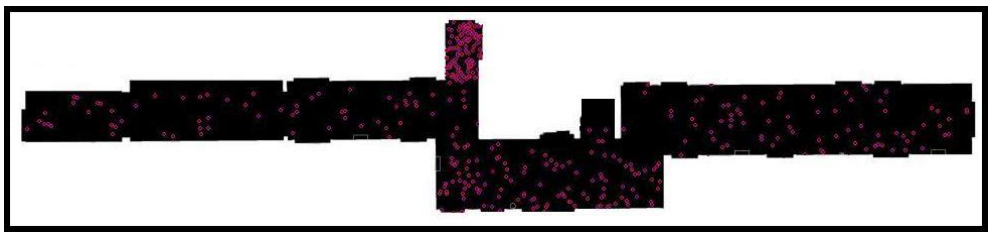

Figura 3.9: Particelle generate su tutto il *free space*.

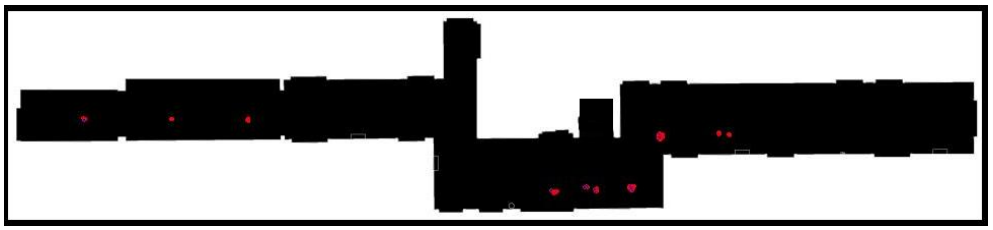

Figura 3.10: Effetto del ricampionamento sulla distribuzione spaziale delle particelle.

Capitolo 4

Strumenti disponibili

4.1 Librerie e strumenti

I filtri probabilistici sono ormai uno strumento affermato per lo sviluppo di applicazioni ingengneristiche. Nell'ambito della stima dello stato sono stati costruiti numerosi programmi che realizzano filtri bayesiani.
Nello sviluppo di un sistema di localizzazione è conveniente utilizzare strumenti software sufficientemente testati che garantiscano la piena efficacia e la robustezza del sistema, oltre a ridurre i tempi di sviluppo.
Non meno importante è l'architettura di controllo del robot. Essa funge da intermediario tra il localizzatore e il sistema, fornendo i controlli e le misurazioni dei dispositivi sensoriali.

Nei prossimi paragrafi verranno presentate le librerie utilizzate per costruire il programma di localizzazione.

4.2 OROCOS

Il progetto OROCOS è nato con l'intento di produrre un'architettura *Open Source* che fornisca una base per lo sviluppo di applicazioni valide sia per robot mobili che per robot manipolatori [9]. Il linguaggio utilizzato nella realizzazione è il *C++*, che ha favorito l'uso di librerie portabili come POSIX e ACE[10] permettendo il

mantenimento di un alto grado di indipendenza dalla piattaforma di esecuzione. La maggior parte degli schemi di comunicazione è stata realizzata in CORBA per permettere l'interoperabilità tra componenti eterogenei.
Il progetto OROCOS è tuttora incompleto, ma ha dato un notevole contributo alla filosofia dell'*Open Source*, che consente un'evoluzione progressiva del software attraverso l'apporto di persone che non sono direttamente legate al progetto, grazie anche alla sua modularità. Le principali caratteristiche di comunicazione del framework OROCOS sono [11][12][8]:

- Pattern predefiniti di comunicazione per garantire l'interazione tra i vari componenti e per separare la realizzazione interna di un componente dal suo comportamento esterno.
- Pattern di comunicazione che forniscono una chiara descrizione dei metodi offerti dall'interfaccia dei componenti.
- Le comunicazioni sono basate su CORBA utilizzando *IDL* per descrivere i dati utenti negli oggetti di comunicazione; questo permette indipendenza dalle piattaforme e dalla locazione.
- Si utilizzano oggetti *communication*, invece di metodi remoti di invocazione, riducendo così al minimmo il traffico, inoltre è possibile utilizzare un'arbitraria struttura dati come *STL* senza essere costretti a utilizzare sempre IDL.

Le principali caratteristiche del framework OROCOS sono:

- i moduli sono le unità che caratterizzano le applicazioni sviluppate con OROCOS. Essi sono tecnicamente realizzati come processi, possono essere distribuiti su piattaforme differenti ed eventualmente attivi su più *thread*. L'approccio è *object-oriented* e, quindi, permette di accedere ai moduli mediante chiamata a metodo e di nascondere la struttura del network agli utenti.
  I meccanismi di sincronizzazione e di aggancio riguardanti la struttura *multi-thread* sono già integrati all'interno dei pattern di comunicazione.

- *Reattività*: la ricezione dei messaggi è gestita in modo indipendente. Questo permette l'eliminazione dei ritardi di ricezione. Non c'è *deadlock*, quindi ogni modulo può ricevere contemporaneamente i propri messaggi.
- *Topologia*: la comunicazione è basata su messaggi in un ambiente *peer-to-peer*. La comunicazione tra moduli avviene via *socket* e in modo decentralizzato, riducendo i ritardi.
- *Primitive*: tutte le comunicazioni tra moduli sono supportate da una serie di primitive; il set dei pattern deve essere piccolo e facilmente utilizzabile tenendo conto delle necessità degli utenti quali la semplicità d'uso e la chiarezza della struttura.
- *Threading*: Tutte le primitive di comunicazione sono *thread safe*: i *thread* multipli possono accedere allo stesso server contemporaneamente senza bisogno di ulteriori sincronizzazioni. Questo riduce i disagi prodotti dalle regole di sincronizzazione e semplifica la struttura interna del modulo.

Per concludere la descrizione del progetto OROCOS è rilevante anche elencare i principi fondamentale su cui esso si fonda:

- *Scambio di messaggi*: chiamate a procedura remota o a memoria condivisa sono evitate.
- *Richieste/risposte asincrone*: l'invocazione del metodo e il suo completamento sono separati dalla sua esecuzione.
- *Peer-to-peer network*: l'utilizzo di un server centrale non è accettabile in applicazioni robotiche.
- *Gestione delle eccezioni*: non vengono utilizzate nei pattern di comunicazione. Il motivo di questa scelta è la difficoltà nel gestire le eccezioni su sistemi piccoli o *real time*.

Pattern di comunicazione

Nei programmi di controllo di un sistema basato su sensori e motori, la complessità può essere limitata solo mediante l'utilizzo di una serie di componenti discreti.

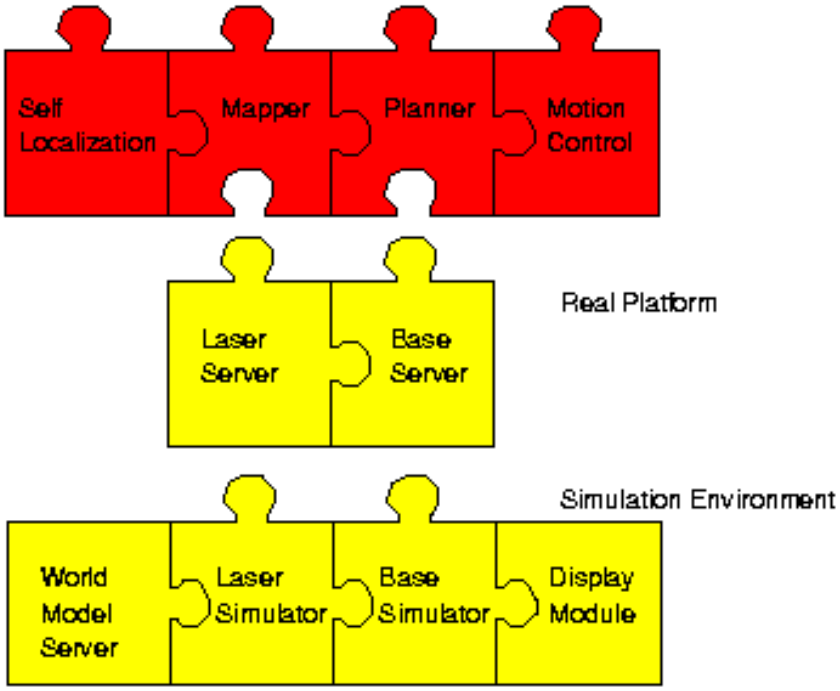

Figura 4.1: Le interfacce standardizzate dei componenti ne facilitano il rimontaggio.

La comunicazione tra quest'ultimi viene gestita realizzando un'interfaccia che permetta una chiara distinzione fra il suo comportamento esterno e la sua realizzazione interna.

I dati vengono trasmessi dagli utilizzatori come *communication objects*. Per accedere ai servizi i pattern di comunicazione forniscono dei metodi che sono identici per ogni componente, questo agevola la comprensione dei servizi forniti dai vari componenti e il loro utilizzo.
In figura 4.1 viene mostrato come le interfacce standardizzate dei componenti facilitino l'accesso ai servizi offerti e il rimontaggio dei componenti: ogni servizio di un componente è basato sulla totale specificazione dei pattern, i quali forniscono metodi di accesso testati e chiaramente strutturati.

La semantica predefinita dei pattern aiuta il costruttore del componente e quello dell'applicazione a creare e utilizzare i componenti distribuiti senza tener conto di dove vengano utilizzati.

Struttura dei pattern di comunicazione

- *communication objects*: caratterizzano i pattern di comunicazione e contengono sia la struttura dati da trasmettere, sia i metodi d'accesso.
- *Servizio*: è il risultato dell'unione tra un pattern di comunicazione e un *communication object*. Il primo definisce i metodi di accesso al servizio, mentre il secondo il contenuto da trasmettere.
- *Server-Client/Producer-Consumer/Master-Slave*: ogni pattern di comunicazione connette due terminali. A seconda del tipo di interazione, il pattern utilizzato prenderà il nome di *Server-Client/Producer-Consumer/Master-Slave*, in modo da distinguere i ruoli delle parti in gioco.

4.3 Orocos::Smartsoft

Smartsoft è un framework software realizzato nell'ambito del progetto OROCOS. Sistemi che prevedono l'utilizzo di sensori e motori possono essere semplificati mediante *template* per i più comuni pattern di comunicazione. L'interfaccia di ogni componente permette una chiara distinzione tra il suo comportamento esterno e la sua realizzazione interna, elemento fondamentale per sistemi con numerosi componenti e sviluppati da diversi progettisti.

Smartsoft è distribuito in diverse realizzazioni che differiscono tra loro per i meccanismi di comunicazione. La versione basata sul *middleware* CORBA è chiamata *Orocos::Smartsoft*. Di seguito ne vengono evidenziati i pattern [13].
La figura 4.2 mostra come l'interfaccia dei componenti sia formata da pattern di comunicazione standardizzati che consentono di individuare facilmente l'interazione tra i componenti e permettono l'accesso a servizi su più piattaforme in ambienti basati su *thread*(fig. 4.3).

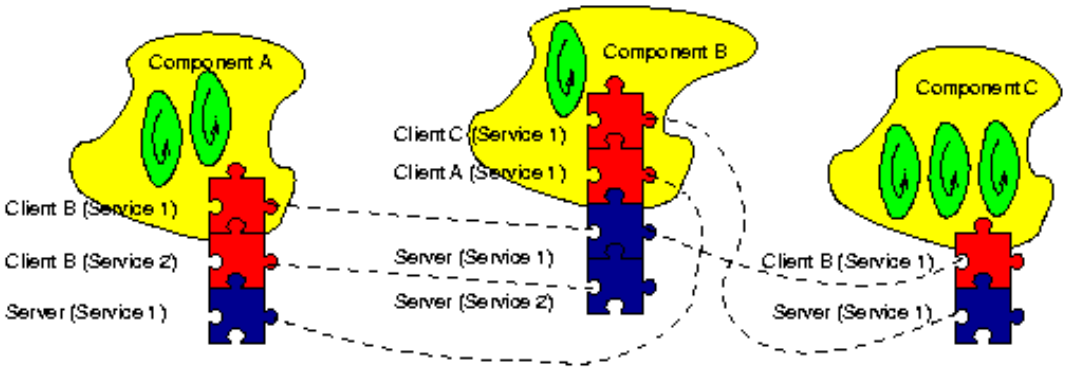

Figura 4.2: Schema di interazione tra componenti in *Orocos::Smartsoft*

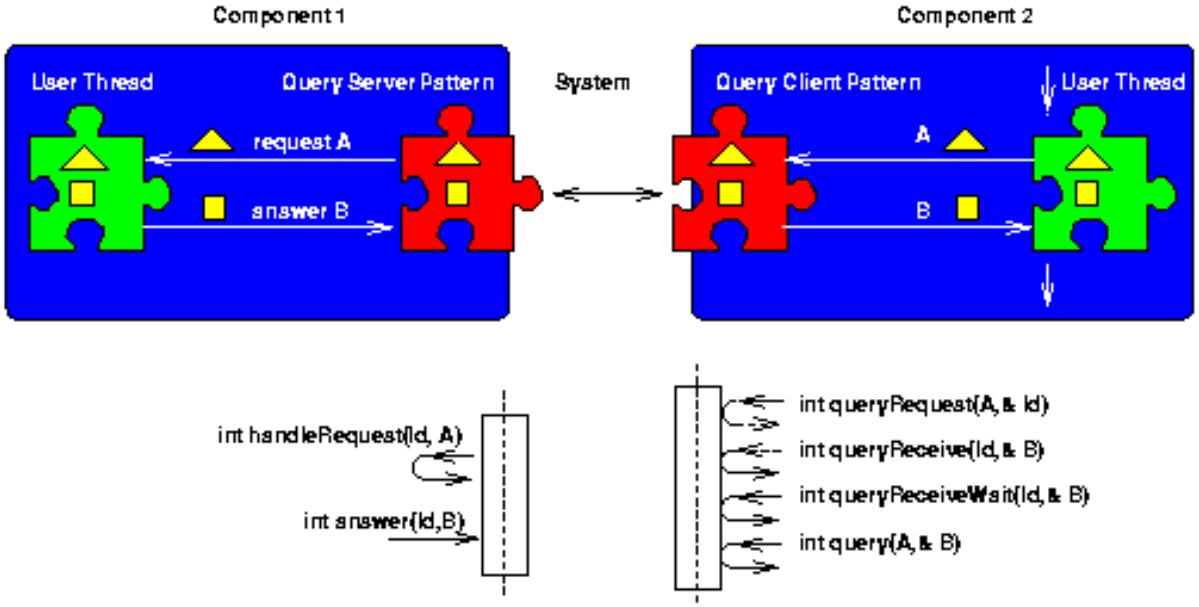

Figura 4.3: Utilizzo dei pattern di comunicazione

Pattern	**Descrizione**
Send	Trasferisce dati dal client al server senza la ricezione di una conferma dal server. Rappresenta una comunicazione *monodirezionale* utile per inviare comandi e settare configurazioni.
Query	Il client effettua una richiesta al server ed attende la risposta. Rappresenta una comunicazione *bidirezionale* tra un solo client ed un solo server. Può essere utile quando un'informazione viene utilizzata ad una frequenza molto bassa rispetto alla velocità con cui viene prodotta: è più ragionevole che il client la richieda mediante una query piuttosto che venga prodotta una quantità eccessiva di aggiornamenti non necessari.
Autoupdate	Uno o più client si registrano presso un server richiedendo un'informazione che viene inviata non appena nuovi dati sono disponibili. Rappresenta una comunicazione di tipo *push*. È possibile ridurre il traffico, se il client richiede l'informazione ad una frequenza minore rispetto a quella con cui viene prodotta, mediante una richiesta che prevede l'invio dei dati soltanto ad ogni ennesimo aggiornamento (*autoupdate timed*).
Event	Il server individua un evento avvenuto sullo stato del sistema ed informa in modo asincrono il client che ne assicura la gestione. Gli eventi sono utilizzati principalmente per notificare modifiche nello stato che sono rilevanti per coordinare i task in esecuzione.
State	Fornisce la possiblità di gestire lo stato interno di un componente, al fine di proteggere le risorse richieste evitando che le attività vengano interrotte all'interno di sezioni critiche
Configuration	Supporta una relazione di tipo *Master-Slave* tra i moduli che consente l'attivazione selettiva delle loro attività. Il modulo slave, quando un'attività viene disattivata, deve impedire che il proprio stato interno possa diventare inconsistente, deve proteggere l'esecuzione delle sezioni critiche e gestire la terminazione delle comunicazioni pendenti con gli altri moduli.

Tabella 4.1: Pattern di comunicazione identificati in *OROCOS*.

4.4 Libreria BFL

Nell'ambito della stima dello stato sono eseguiti numerosi progetti di ricerca a livello internazionale. Tali progetti hanno portato alla costruzione di diversi pacchetti che realizzano l'approccio bayesiano: librerie per il *Kalman Filtering*, per il filtraggio basato su griglia, su metodo Monte Carlo, su reti bayesiane o su modelli di Markov sono esempi che dimostrano il livello di studio di questo problema.
Bayesian Filtering Library [14] è una libreria C++ realizzata da *Klaas Gadeyne*[15] nell'ambito del progetto di ricerca su robot manipolatori autonomi nel *Department of Mechanical Engineering* alla *Katholieke Universiteit Leuven Nederlands*[16]. Le caratteristiche principali della libreria sono[14]:

- E' costruito su librerie matematiche, ma comunica con esse attraverso un *wrapper* eliminando la dipendenza da una libreria specifica.
- E' basato sul concetto di funzione densità di probabilità e permette di distinguerla dalla sua rappresentazione (per esempio a campioni o analitica)
- Il modello di sistema e percettivo sono indipendenti dal filtro utilizzato.

BFL, come OROCOS, è un progetto *Open Source* che può essere letto, ridistribuito e modificato da altri programmatori. L'idea chiave seguita nella costruzione di questo framework è stata quella di fornire un insieme di classi astratte fortemente adeguabili alle esigenze del progettista software.
Sia per le caratteristiche intrinseche sopra citate, sia perché ritenuto uno strumento più maturo ed affidabile, nel lavoro di tesi BFL è stato utilizzato per realizzare un filtro particellare gestito con il metodo Monte Carlo.
Come mostrato in figura 4.4, le densità di probabilità sono racchiuse in una classe *Pdf* ereditata da *MCPdf*. Quest'ultima contiene una lista di oggetti istanze della classe *WeightedSample*, che rappresenta i campioni pesati.
Il filtro *Boostrap* è costruito su una classe che eredita *Filter*. Attraverso i campi *Prior*, *Post* e *Proposal* si possono modellare le funzioni densità di probabilità descritte nel primo capitolo (figura 4.6). I modelli di sistema e delle misure sono anch'essi costruibili partendo dalle classi astratte *SystemModel* e *MeasurementModel*. In particolare il modello di sistema lineare è già realizzato in *LinearSystemModel*.

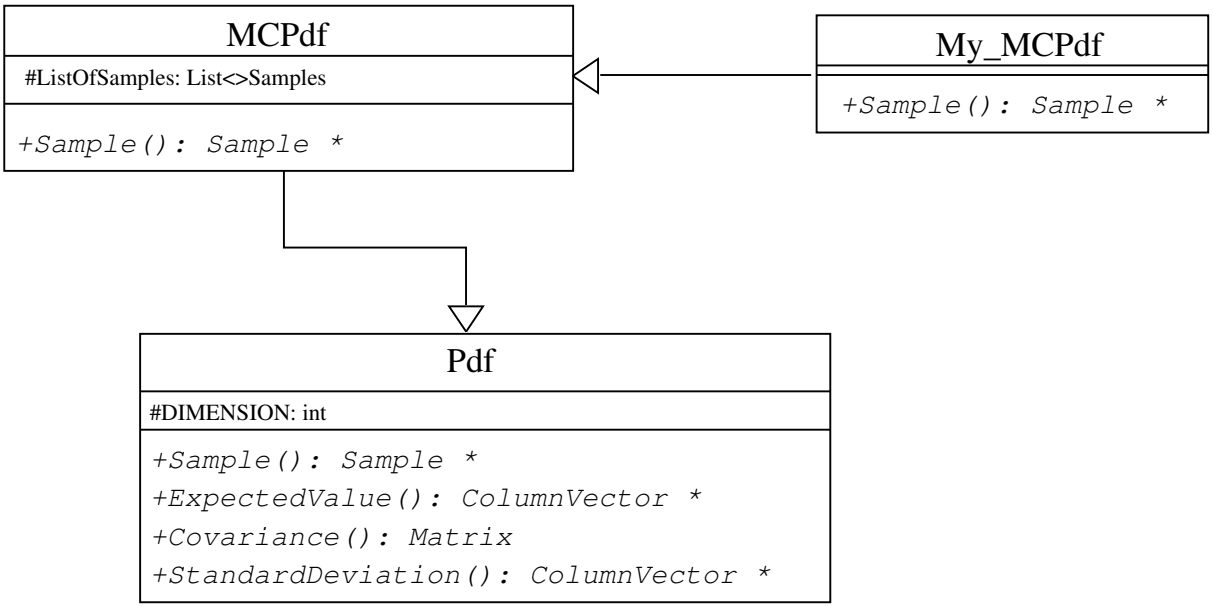

Figura 4.4: Realizzazione di una densità di probabilità campionata con metodo Monte Carlo

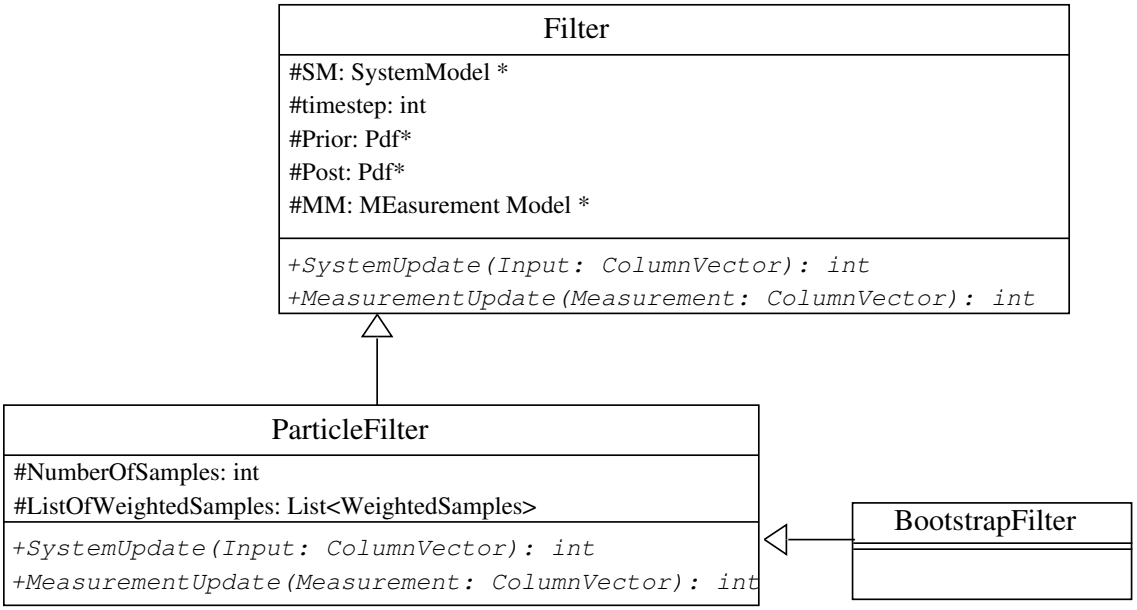

Figura 4.5: Realizzazione del filtro particellare bootstrap

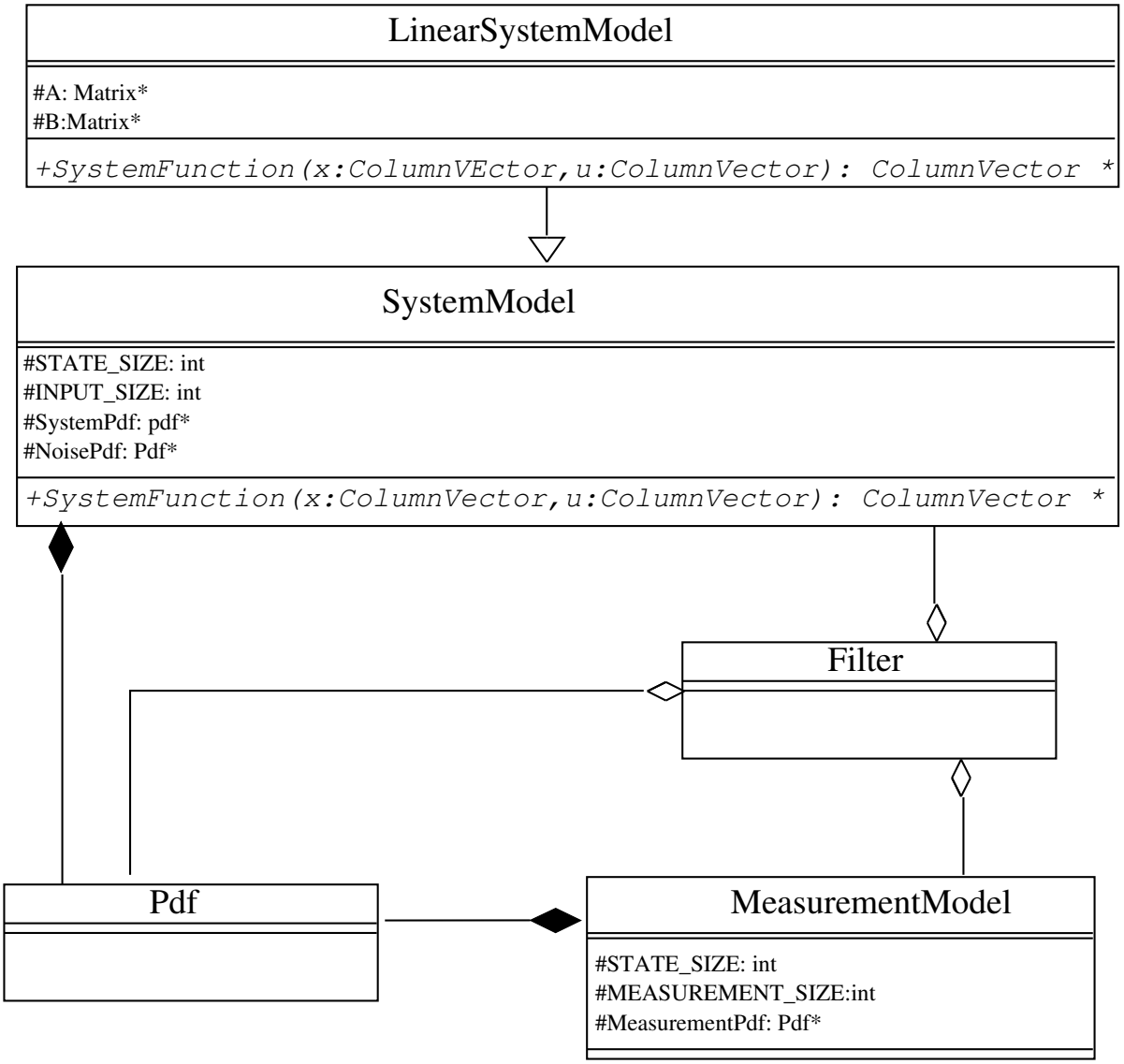

Figura 4.6: Realizzazione dei modelli di sistema e delle misurazioni

4.5 Strumenti per la simulazione: Player/Stage

Per costruire il sistema di localizzazione sono stati utilizzati due tool che permettono, una volta inseriti il modello del robot e la pianta dell'ambiente, di simulare il sistema. Queste due applicazioni sono il risultato di un lavoro di ricerca della *University of Southern California* e della *Simon Frazer University Vancouver*.

Player

Player è un *server* per il controllo dei sensori e degli attuatori di un robot mobile. Un programma *client* può connettersi a *Player* via *socket standard TCP*. La comunicazione si ottiene attraverso lo scambio di messaggi. *Player* è stato progettato per essere indipendente dal linguaggio di programmazione e dalla piattaforma, questo implica che il programma *client* può essere scritto in qualsiasi linguaggio che supporti la comunicazione sia *socket TCP*.

Il numero delle connessioni al server, compatibilmente alle capacità della rete, è illimitato [17].

C++ Client Library

Questa libreria è utilizzata per testare delle nuove caratteristiche del server e, anche se non perfetta, è altamente testata. E' costruita su un modello *service proxy* attraverso il quale il *client* può possedere oggetti locali che fungono da *proxy* per l'accesso ai servizi remoti.

Ci sono due tipi di *proxy*: uno particolarmente importante perché rivolto al server chiamato *PlayerClient*, e altri rivolti a specifici componenti del robot. Ognuno di essi è realizzato in una classe separata.

L'utente prima crea un'istanza di *PlayerClient* e la utilizza per stabilire una connessione con il *server*; successivamente si creano dei *proxy device-specific* inizializzati utilizzando quello già esistente *PlayerClient*[18].

Stage

Stage è un prodotto che fa parte del *Project Player/Stage*, una raccolta di strumenti software realizzati nei lavori di ricerca su robot autonomi e sistemi a sensori intelligenti. *Stage* simula una popolazione di robot mobili, oggetti, sensori in un ambiente bidimensionale mappato in formato digitale. E' un software pensato come sistema multi-agente in grado di fornire modelli computazionalmente semplici dei dispositivi del robot cercando di simularli con un buon grado di fedeltà.

Gli oggetti simulati da *Stage* sono controllati attraverso *Player*. Il primo simula i dispositivi robotici e li rende disponibili al secondo, per cui un *client* che si collega non distingue operativamente il robot reale da quello simulato: le modalità di controllo sono le stesse.
Stage può simulare attuatori, sonar, laser, dispositivi di visione, odometria, *bumper* e altri dispositivi che di solito sono montati su un robot mobile[19].

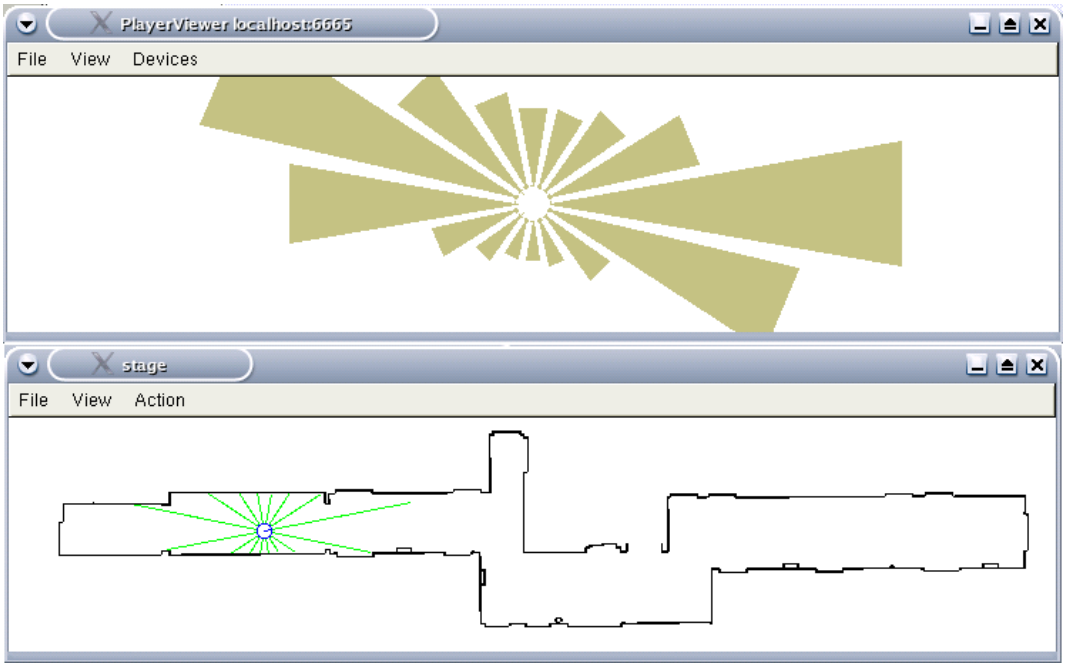

Figura 4.7: Player/Stage

4.6 Libreria Qt

Qt è un framework della *Trolltech* per applicazioni *GUI* in C++ multipiattaforma[20]. Esso fornisce tutte le funzionalità attualmente disponibili nell'ambito delle interfacce grafiche. L'utilizzo delle librerie Qt è di solito finalizzato all'interazione tra utente e applicazione. Se, ad esempio, viene selezionata un'icona di un menù, il programma deve visualizzare la finestra ad essa associata.
E' il *main event loop*, ovvero il ciclo principale dell'applicazione grafica, ad occuparsi del monitoraggio della finestra di sistema e di tutte le altre sorgenti.

Le classi di Qt sono spesso astratte, per cui lo sviluppatore le deve ereditare reimplementandone i metodi che ne caratterizzano le azioni.
Ad applicazioni GUI è associato un solo oggetto *QApplication*, indipendentemente dal numero di finestre create. *QApplication* è la classe che inizializza l'applicazione, setta i suoi parametri e governa il ciclo principale.

Il modello a oggetti del C++ standard ha una connotazione statica poco adatta allo sviluppo di applicazioni interattive. Qt fornisce un paradigma più flessibile chiamato *Qt Object Model*, che utilizza un modello di comunicazioni a segnali e *slot*. Un'interazione con l'interfaccia grafica produce un segnale al quale è associata una funzione eseguita mediante chiamata a metodo di un oggetto.
Nel lavoro di tesi Qt è stato utilizzato per la visualizzazione della mappa e della posizione del robot.

Capitolo 5

Il programma

5.1 Requisiti

L'obiettivo del lavoro di tesi è quello di costruire un programma che localizzi il robot mobile Nomad 200 all'interno di un ambiente mappato. Il localizzatore deve avere un certo numero di caratteristiche:

- Deve essere indipendente dall'algoritmo di moto eseguito sul robot.
- Non può essere troppo invasivo sulle risorse computazionali per evitare di influenzare i *behaviour* in esecuzione.
- La struttura del programma deve permettere l'aggiunta di nuovi input sensoriali, nel caso in futuro fossero disponibili.

Per permettere all'utente di osservare l'evoluzione del moto del robot il programma di localizzazione rende disponibile anche una finestra sul *display* in cui viene rappresentata la posizione sulla mappa metrica.

L'applicazione software è stata testata sia in simulazione che nel caso reale. Essa rappresenta un punto di partenza nell'ambito del problema della localizzazione che, come ogni lavoro di ricerca, potrà essere integrata e migliorata. L'utilizzo di fonti sensoriali come sonar e infrarossi ha permesso di ottenere buoni risultati nel *position tracking*. La localizzazione globale è, invece, difficilmente ottenibile con l'utilizzo esclusivo di queste fonti sensoriali. Test eseguiti hanno evidenziato come in simulazione a volte il robot riesca a localizzarsi senza conoscere la sua posizione iniziale,

cosa che si rivela molto più improbabile nel caso reale. Fonti sensoriali come laser o telecamere, comunque integrabili facilmente nel software, permetterebbero di innalzare molto le probabilità di successo anche nel caso di *global localization*.

5.2 La struttura

Il programma di localizzazione è stato interamente scritto in linguaggio C++. Questa scelta è stata motivata non solo dai vantaggi che un linguaggio *object-oriented* fornisce, ma anche dal fatto che le librerie utilizzate per interfacciarsi col robot e per realizzare il filtro particellare sono tutte scritte in C++. Questo linguaggio permette la costruzione di codici estremamente modulari e che, quindi, possono essere migliorati da altri programmatori.

La struttura del sistema è stata pensata tenendo conto della moltitudine di attività che un programma di questo tipo svolge, della elevata quantità di dati che deve scambiare con il robot nonché dell'interfaccia utente. Tutte le classi costruite sono racchiuse nel *namespace* LSOFT (*Localization SOFTware*). Il progetto realizzato è costituito da un file eseguibile e quattro librerie di cui viene di seguito fornita una breve descrizione:

1. *Perception*: legge i dati odometrici e sensoriali del robot e li rende elaborabili dal filtro particellare.

2. *Graph*: si occupa dell'elaborazione del file *.pnm* finalizzata alla costruzione dell'*occupancy grid* e della visualizzazione dei dati su monitor.

3. *Localize*: è la libreria dedicata all'elaborazione dei dati allotetici e idiotetici al fine di fornire la posizione del robot in un sistema di riferimento assoluto.

4. *Model*: contiene i modelli di sistema e di percezione.

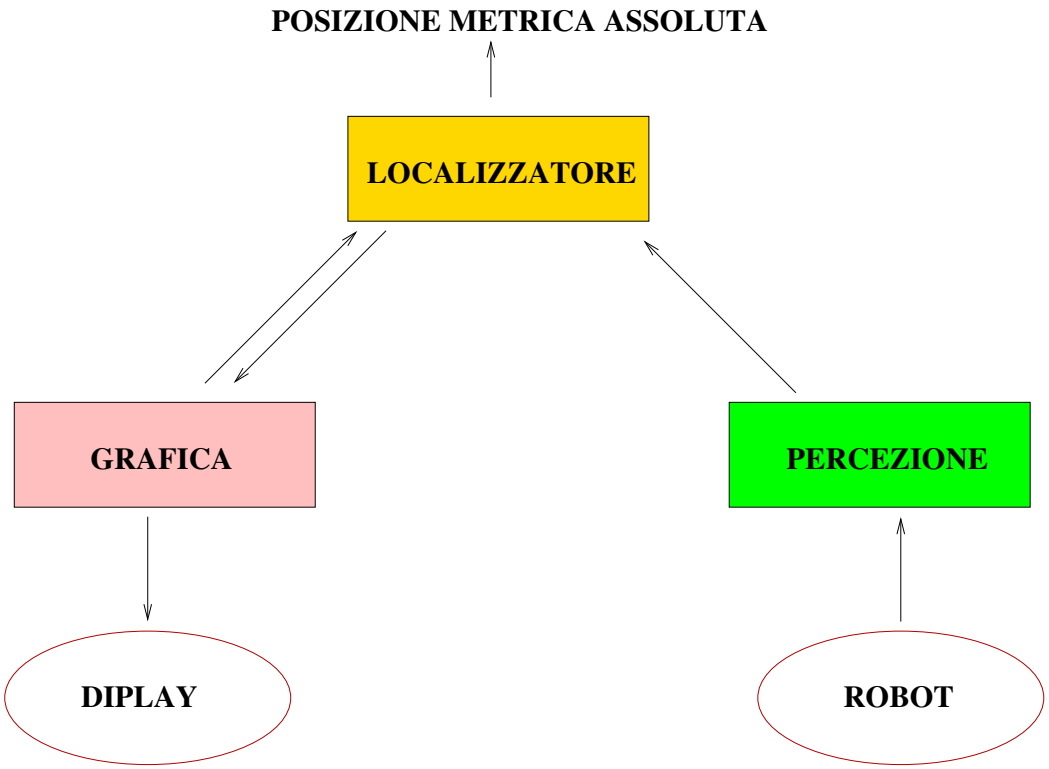

Figura 5.1: Struttura del progetto

5.3 Perception

Le classi appartenenti a questa libreria sono dedicate alla lettura dell'odometria, dei valori misurati dai sensori ultrasonici e dai dispositivi radiazione infrarossa.
Il server che gestisce la comunicazione con il robot viene interrogato periodicamente per acquisire i valori necessari al ciclo di localizzazione. Mentre i valori dei sonar possono essere interrogati unicamente nel momento in cui si deve attribuire un peso ai campioni del filtro particellare, è fondamentale accedere ai valori odometrici con una frequenza molto elevata: la fase di predizione nell'algoritmo di localizzazione è basata sulle velocità traslazionale e rotazionale del robot. Se questi valori non sono corretti le particelle possono seguire traiettorie totalmente errate.
Il problema si può manifestare nel caso l'algoritmo di moto modifichi i valori delle velocità del robot così rapidamente da non permettere al localizzatore di leggere tutte le loro variazioni. Eseguire la lettura dei valori solo una volta all'inizio di ogni ciclo potrebbe comportare dei rischi di errata interpretazione dei dati come è mo-

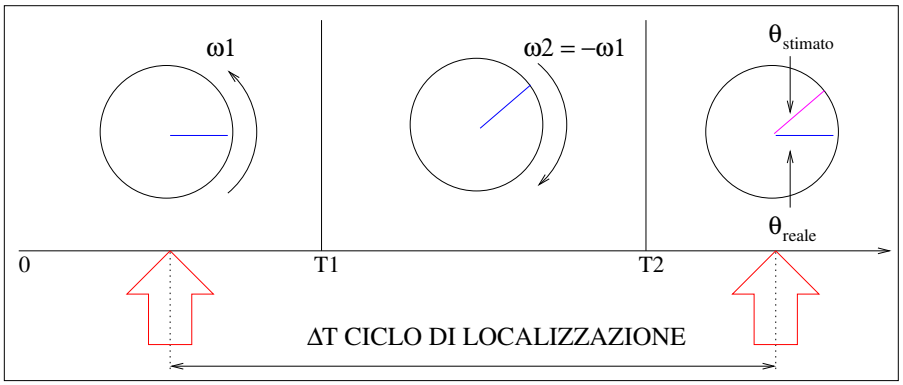

Figura 5.2: Caso di perdita di un controllo sul robot. Le frecce rosse indicano i momenti in cui vengono letti i valori odometrici.

strato in figura 5.2.

Il server non può in ogni caso fornire l'evoluzione continua delle variabili odometriche, ma per ridurre gli errori è necessario sfruttare la massima frequenza alla quale è permesso accedervi. L'idea è quella di creare dei *thread* dedicati alla lettura dei controlli e dei valori sensoriali. Al ciclo di localizzazione verrà fornito un dato maggiormente affidabile ricavato dalla lettura di più valori nell'intervallo di tempo occupato da un ciclo di localizzazione.

La libreria di percezione è stata realizzata con classi leggermente diverse a seconda che dovessero utilizzare il sever del robot reale o quello del simulatore. Nel listato 4.1 viene mostrato il meccanismo attraverso il quale il *thread* di percezione aggiorna i valori odometrici. I dati allotetici non necessitano di continuo aggiornamento perché di essi vengono utilizzati solo quelli più recenti. Il meccanismo di trasformazione dei 16 valori sensoriali nel vettore osservazione è già stato illustrato nel 3° capitolo.

5.3.1 Lettura dell'odometria e dei sensori in simulazione

La libreria *PlayerClient*(par. 4.5) fornisce delle classi che permettono la connessione al simulatore *Stage*. Specificando il nome dell'*host* e la porta a cui è connesso il robot simulato è possibile creare dei *proxy* attraverso i quali leggere i valori odometrici e sensoriali. In figura è mostrato il meccanismo d'interazione tra questa classe

e *Player*.

```
u = 0 vettore contenente i controlli
z = 0 vettore contenente le osservazioni;
num_letture = 0;
repeat
  num_letture = num_letture +1;
  u = (u + valore restituito dal server)/num_letture;
until (ciclo di localizzazione richiede u e z)
acquisisci z;
```

Listato 5.1: Algoritmo di acquisizione dei controlli dei dati sensoriali

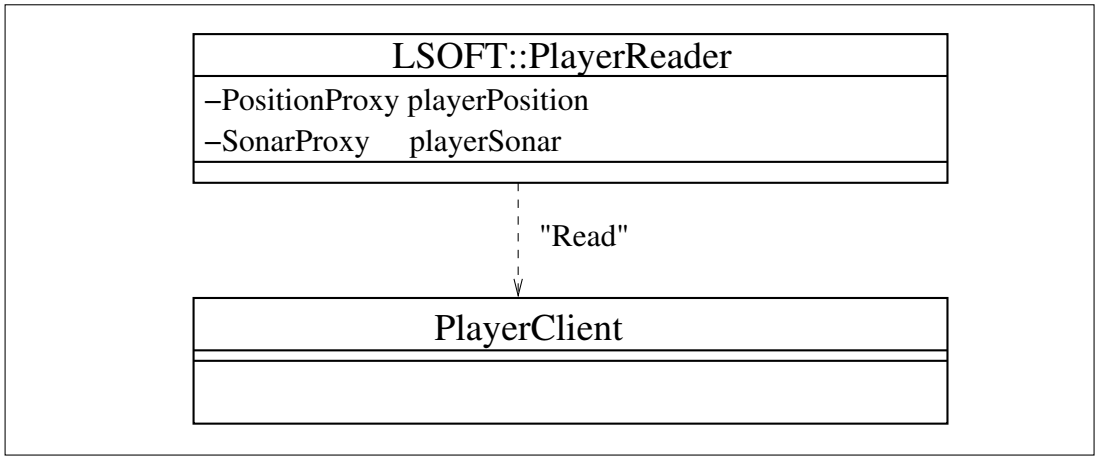

Figura 5.3: Lettura dei dati odometrici e sensoriali mediante chiamata del metodo Read della classe PlayerClient.

5.3.2 Lettura dei dati utilizzando il server del robot reale

Nel caso di interazione con il robot reale è la classe *LSOFT::NomadReader* a connettersi con il server Smartsoft e a leggere i valori necessari al ciclo di localizzazione. I controlli e i dati sensoriali sono forniti da tre *proxy*, istanze della classe *CHS::PushTimedClient*. Quest'ultima è fornita dal framework *Orocos::Smartsoft* e permette ad un cliente del servizio di richiedere aggiornamenti dal *server* a intervalli prestabiliti. Questo avviene grazie a un meccanismo di sottoscrizione nel quale si

specifica la frequenza con la quale il *server* deve inviare i dati richiesti. In questo caso è stato deciso di sottoscriversi alla frequenza massima a tutte e tre le sorgenti d'informazione. Nel caso dell'odometria questa scelta è stata dettata dalla necessità di monitorare in modo accurato il movimento del robot; nel caso dei dati sensoriali, invece, si è voluto ottenere il dato più aggiornato possibile.
L'interazione tra le classi è mostrata in figura 5.4: le istanze della classe *CHS::PushTimedClient* fungono da *proxy* inviando i dati descriventi la velocità traslazionale e rotazionale del robot, nonché i valori ritornati da sonar e infrarossi. Il metodo *getNearestObject* di *LSOFT::NomadReader* ritorna il vettore di due elementi contenente la misurazione.

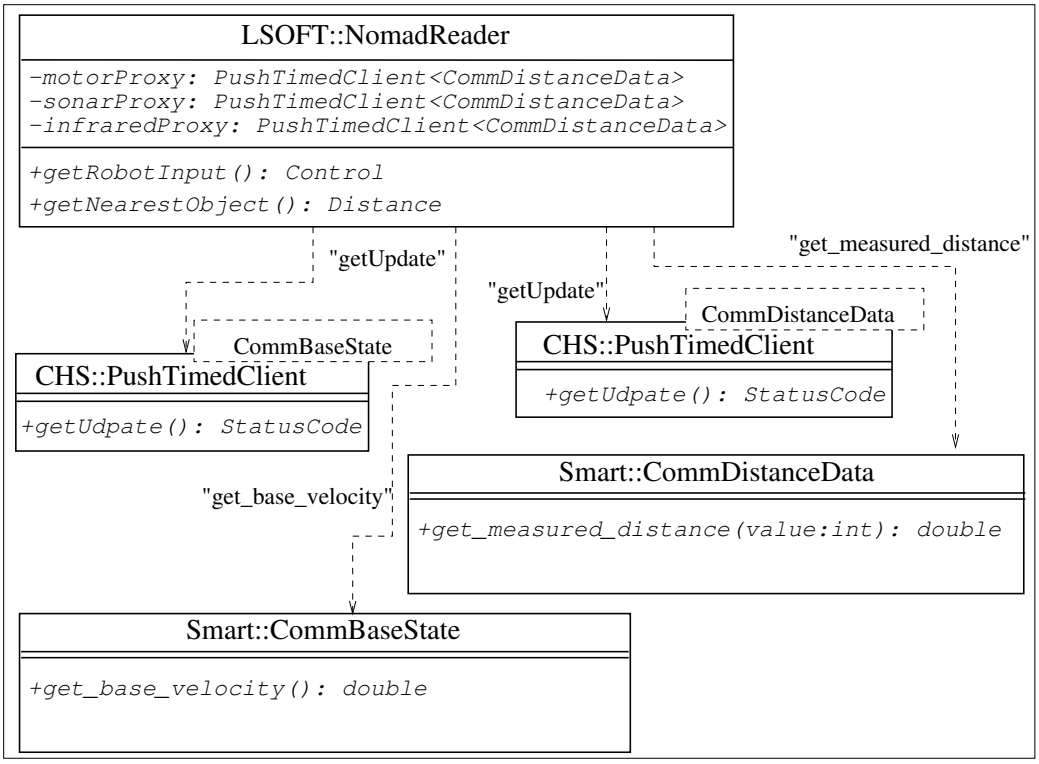

Figura 5.4: Lettura dei dati odometrici e sensoriali utilizzando i *proxy* di Orocos::Smartsoft.

5.4 Graph

Nella libreria *Graph* sono implementate le classi che gestiscono la visualizzazione, importano il file *.pnm*, costruiscono la mappa e la memorizzano. Il *desktop manager kde*, attualmente il più diffuso in ambienti *linux*, rende disponibile le librerie *Qt* per lo sviluppo di applicazioni grafiche. La visualizzazione su finestre con sbarre di scorrimento, lo studio pixel per pixel di immagini memoriazzate su file, è possibile ereditando classi astratte di *Qt*.

5.4.1 La classe Map

Tutte le mappe utilizzate dal sistema di localizzazione hanno la stessa dimensione, ma contengono dati con caratteristiche diverse. Per questo motivo la classe che la gestisce è organizzata a *template* in modo che sia possibile specificare il tipo di dati che essa contiene. La costruzione parte dallo studio del file *pnm*. La classe *QPixmap* permette di leggere il colore dei pixel dell'immagine. Il *free space* è rappresentato in nero, mentre al resto dello spazio è assegnato il bianco.
La risoluzione della mappa definirà il numero di pixel che bisogna leggere per poter asssegnare un valore ad una cella. Come è evidenziato in figura 5.5 tutte le celle so-

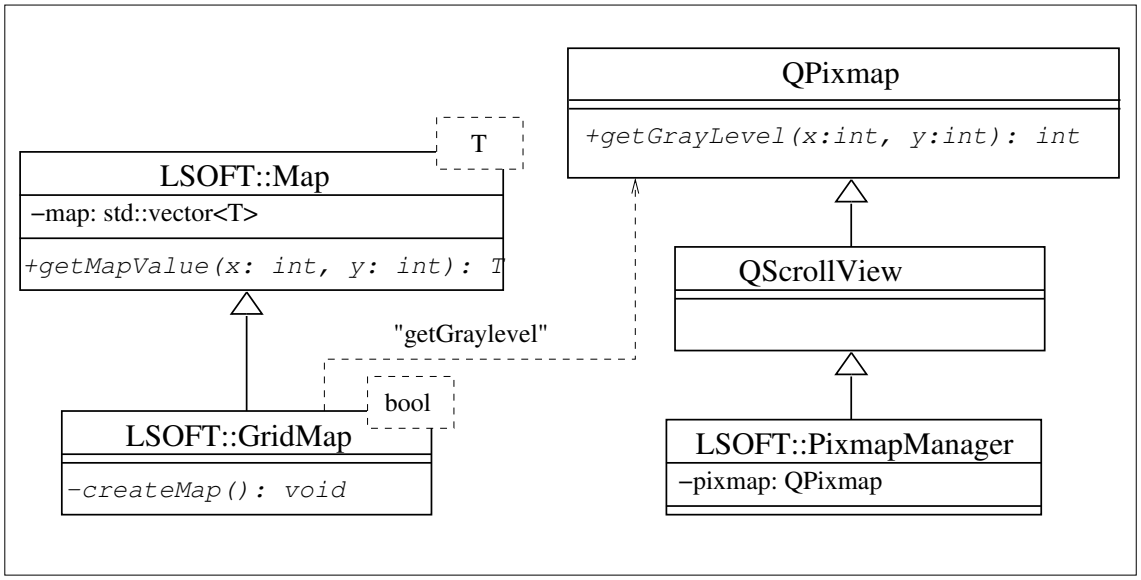

Figura 5.5: Creazione delle mappe.

no memorizzate in una classe della libreria standard C++: *std::vector*. Questa scelta è motivata dall'alto livello di ottimizzazione di questa classe nella gestione delle strutture dati [21].
Il file che viene elaborato deve essere di estensione *pnm* e rispecchiare il più accuratamente possibile la planimetria dell'ambiente.

5.4.2 Visualizzazione

Le librerie *Qt* utilizzano la classe *QApplication* per gestire la visualizzazione su schermo e l'interazione tra utente e oggetti grafici. Alla chiamata del metodo *QApplication::show()* comincia il ciclo principale di Qt che causa l'impossibilità di interagire con gli oggetti creati se non attraverso l'utilizzo di segnali inviati all'applicazione. I segnali sono di solito lanciati in seguito a un'interazione dell'utente con l'applicazione grafica.

Quello della visualizzazione è un problema molto importante anche e soprattutto in fase di *debug*. Alcuni metodi della classe *LSOFT::PixmapManager* caricano la posizione stimata dal ciclo di localizzazione e la rappresentano sulla mappa metrica. Questa operazione avviene ogni mezzo secondo attraverso l'invio di un segnale all'applicazione. La classe *QTimer* ha dei metodi che permettono di eseguire l'aggiornamento del display con frequenza stabilita dal programmatore. In fase di simulazione viene eseguita nell'output grafico anche la rappresentazione delle particelle sulla mappa per poter studiare il comportamento delle densità di probabilità.

L'operazione di *logging* sulla posizione del robot è computazionalmente piuttosto pesante, per cui può togliere efficienza al ciclo di localizzazione. Mentre in fase di simulazione questo problema è trascurabile, la sua rilevanza aumenta se si esegue l'applicazione sul robot.
In figura 5.6 è mostrato il meccanismo di aggiornamento del display. L'oggetto *LSOFT::PixmapManager* quando viene costruito lancia il metodo *QTimer::start(int)* che permette di avviare il temporizzatore. Il visualizzatore verrà risvegliato con la frequenza desiderata.

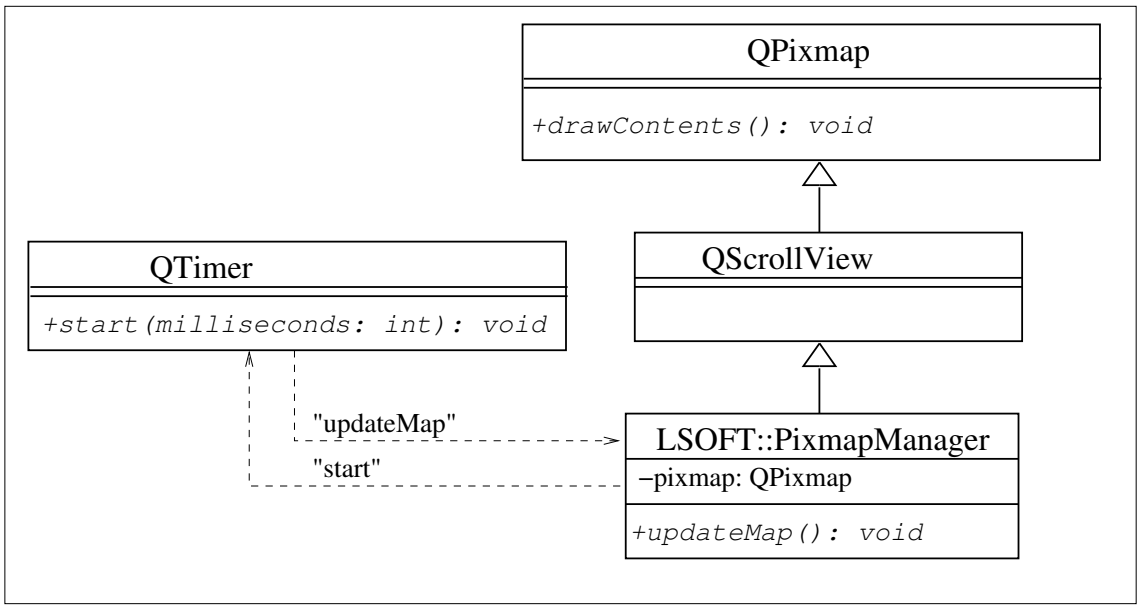

Figura 5.6: Aggiornamento dell'interfaccia grafica

Figura 5.7: Esempio di visualizzazione in fase di simulazione: in alto a sinistra è indicata la posizione del robot; sulla prima mappa sono rappresentate le particelle, mentre sulla seconda il *logging* sulla posizione stimata.

5.5 Model

Per aumentare la modularità del software e permetterne un eventuale miglioramento è stato deciso di inscrivere le classi che realizzano i modelli di sistema e percettivo in una libreria distinta da quella che contiene il filtro particellare. In effetti le librerie BFL permettono di costruire vere e proprie scatole chiuse utilizzabili indifferentemente anche cambiando in modo radicale, per esempio, le equazioni descriventi il modello cinematico del robot.

Il modello di sistema è stato costruito come oggetto *LinearAnalyticSystemModelGaussianUncertainty* di BFL. Questa classe permette di inserire le matrici A e B dell'equazione 3.2, come istanze della classe *BFL::Matrix*. La gestione di questi oggetti matematici è affidata alla libreria *Newmat*. BFL comunica con essa mediante un *wrapper*.
Il rumore è modellizzato come gaussiano con caratteristiche studiate sperimentalmente. Esso è fondamentale anche in fase di simulazione, non solo perché, di fatto, l'accesso alle variabili di controllo non è continuo e, quindi, causa del disturbo, ma anche perché maggior grado di diversità si introduce all'interno delle previsioni del moto, più le ipotesi percorreranno cammini diversi permettendo al sistema di correggersi nel caso di stima errata. Media e varianza del disturbo sulle coordinate di stato si inseriscono specificando il vettore delle medie e la matrice delle covarianze. Non bisogna dimenticare, inoltre, che i filtri particellari modellano come gaussiane variabili che, in realtà,non lo sono. Ecco perché le caratteristiche del rumore rappresentano un'interfaccia per regolare il sistema e non necessariamente devono essere calcolate in modo rigoroso a priori. Il valore che assumono è quello che fa convergere meglio la stima dello stato con lo stato reale e può essere ottenuto iterativamente. Usando *Stage* non ci sono problemi legati all'attrito delle ruote col pavimento o alla meccanica del motore, ma l'accesso ai controlli inviati dal server *Player* ha una frequenza che varia a seconda della potenza computazionale disponibile. Non è conveniente cercare di stimare le caratteristiche del disturbo a priori: solamente in fase di sperimentazione, partendo da valori ragionevoli, si possono quantificare in modo ottimo tutti i parametri.

Il modello delle percezioni non è ottenibile come semplice istanza di una classe della libreria BFL. L'utilizzo delle mappe delle distanze e angolari non è stato

previsto: si è trattato dunque di ereditare la classe *AnaliticMeasurementModelGaussianUncertainty* per costruire il modello. Il problema principale si è manifestato nel realizzare una classe che potesse gestire la densità di probabilità $P(z_k|x_{k)}$. L'equazione 3.16 evidenzia le grandezze in gioco al momento della pesatura di una particella.
Il calcolo della densità di probabilità condizionata è effettuato da un metodo di *MeasureConditionalPdf*, classe figlia di quella astratta *BFL::ConditionalGaussian*. Nella descrizione del ciclo di localizzazione verrà chiarito meglio l'utilizzo di queste classi.
In figura 5.8 viene descritta la struttura della libreria *libModel*.

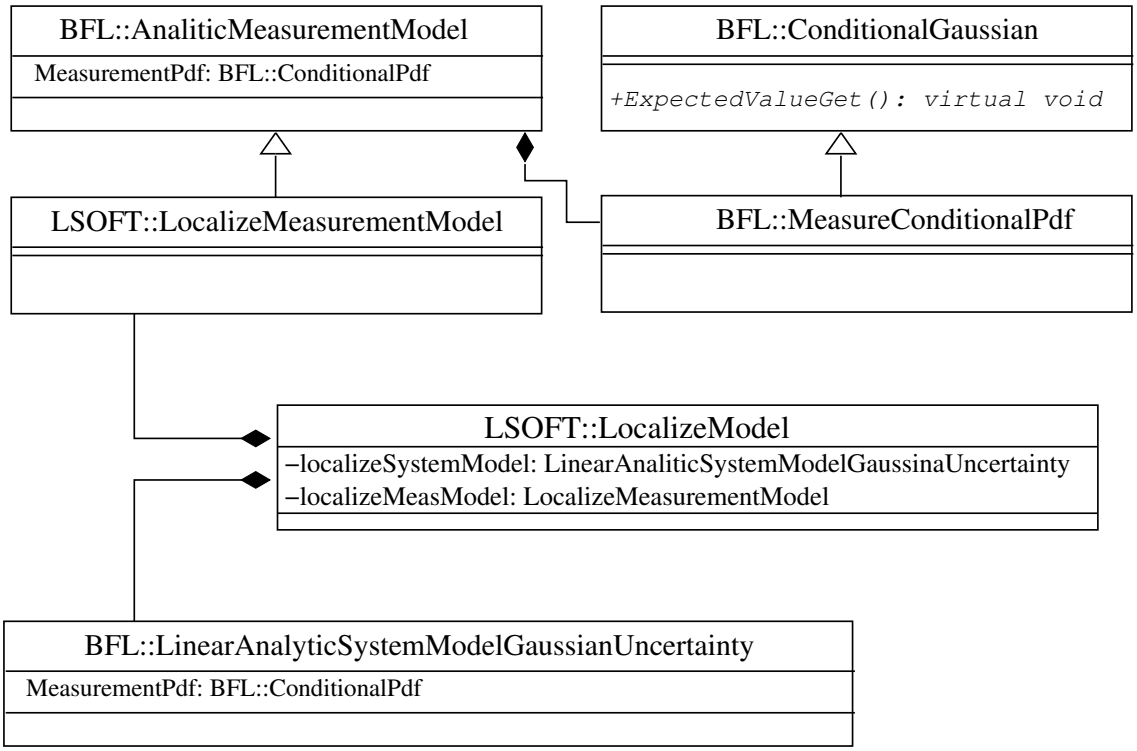

Figura 5.8: Struttura della libreria contenente i modelli di sistema e delle percezioni.

5.6 Localize

La libreria dedicata alla localizzazione contiene le classi che realizzano il filtro particellare e quelle che regolano il *threading* nel sistema. Molte di esse ereditano dalla libreria astratta BFL.
Il cuore della libreria è la classe *Localizer*. Essa inizializza il filtro, costruisce le mappe ed effettua il ciclo di localizzazione.
Come già sottolineato in precedenza, le strutture dati anche piuttosto complesse sono state costruite per minimizzare l'utilizzo di potenza computazionale durante il ciclo di localizzazione. Le mappe vengono create quando viene istanziata *Localizer* e non subiscono modifiche per tutto il tempo di utilizzo dell'applicazione. I passi del ciclo di localizzazione vengono eseguiti dal metodo *Localizer::localizeRun()*.

```
class Localizer
{
  public:
    Localizer(int argc, char **argv, double map_resolution)
   ~Localizer();
    int localizeRun();
    double getX();
    double getY();
    double getTheta();
  private:
    GridMap           occupancyMap;
    Map<double>       distanceMap;
    Map<double>       thetaMap;
    void createLocalizeMaps(Map<long> & xOffsetMap
                          , Map<long> & yOffsetMap);
    LocalizeParticleFilter *localizeMCFilter;
    BFL::LinearAnalyticSystemModelGaussianUncertainty
                                    *localizeSystemModel;
    LocalizeMeasurementModel *localizeMeasModel;
    LocalizeMCPdf *priorDensity;
    LocalizeConditionalPdf *proposalDensity;
    LocalizeModel     *nomadModel;
};
```

Listato 5.2: Estratto delle parti significative di Localizer.h.

5.6.1 Creazione e gestione dei campioni

Il numero di particelle è il primo parametro che è necessario stabilire per avviare il ciclo di localizzazione. Molti campioni assorbirebbero troppa potenza computazionale, viceversa un numero esiguo di particelle non permetterebbe una stima adeguata della posizione.
La classe *BFL::WeightedSample* incapsula i campi e i metodi sufficienti a gestire un campione della densità di probabilità che si vuole modellare.
La classe *LSOFT::localizeMCPdf* contiene i metodi per la generazione di particelle su tutto il *free space*, necessari nel caso si voglia tentare una *global localzation*. In caso, invece, di *position tracking*, in BFL sono disponibili metodi per il campionamento di funzioni densità di probabilità gaussiana per l'inizializzazione della posizione.
Se si indica con $X(\eta, \sigma)$ la funzione $P(X(0))$ allora

$$\eta_x = x_0; \quad \eta_y = y_0; \quad \eta_\theta = \theta_0 \tag{5.1}$$

dove x_0, y_0 e θ_0 sono le coordinate di stato iniziali e

$$\sigma_x = x_0; \quad \sigma_y = y_0; \quad \sigma_\theta = \theta_0 \tag{5.2}$$

esprimono la varianza dell'errore che si compie nel misurarle. Non è necessaria un'elevata precisione nella fase di inizializzazione: la retroazione compiuta dal ciclo di localizzazione dovrebbe essere sufficiente a far convergere le particelle nello stato corretto.
La struttura dati che contiene le N istanze di *WeightedSample* è un oggetto *std::list*.
Il campo *priorDensity* punta un oggetto che contiene, alla chiamata del costruttore di *Localizer*, la densità iniziale di probabilità da campionare.

5.6.2 Predizione e pesatura dei campioni

Il filtro *bootstrap* effettua la fase di predizione utilizzando unicamente la $P(x_k|x_{k-1}, u_{k-1})$.
La classe che modellizza questa densità di probabilità è istanziata nell'oggetto *localizeSystemModel*, campo di *Localizer*.

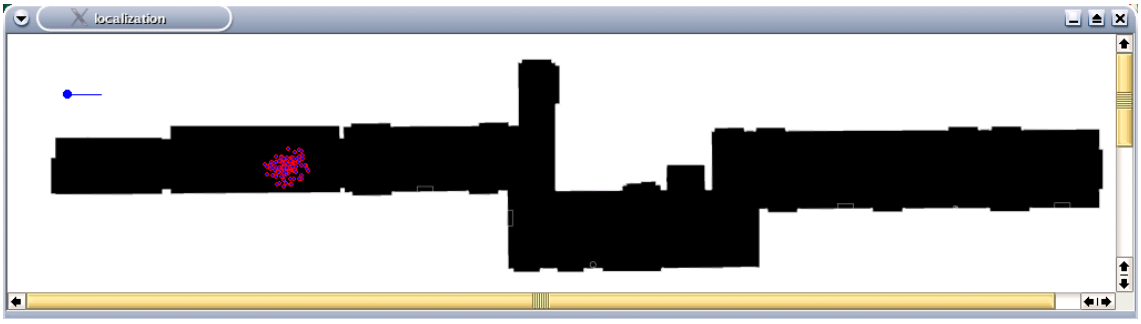

Figura 5.9: Inizializzazione con campioni generati su una densità $P(X(0))$ gaussiana.

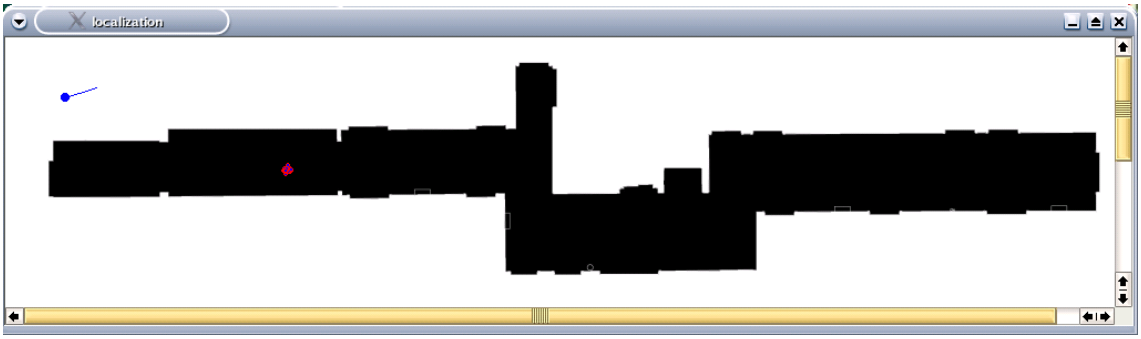

Figura 5.10: Effetto del ciclo di localizzazione sulla distribuzione iniziale di campioni.

La chiamata a *BFL::LinearAnaliticSystemModelGaussianUNcertainty::Update()* aggiorna il sistema spostando le particelle secondo il moto del robot. Questa fase è computazionalmente piuttosto onerosa perché la lista dei campioni pesati deve essere totalmente visitata. I valori dei controlli sono letti da una variabile globale che gestisce la comunicazione con la *thread* di percezione.
Finita la fase *dead reckoning* vengono consultati i valori sensoriali e pesate le particelle secondo il modello delle percezioni contenuto nell'oggetto puntato da *localizeMeasModel*(listato 4.2).

5.6.3 Escape resampling

Nella prima fase di sperimentazione lo studio dell'andamento della distribuzione delle particelle ha evidenziato una forte tendenza della densità $P(x(k) = x_k)$ a con-

vergere in una zona estremamente limitata nello spazio. La perdità di diversità non è in generale un elemento negativo, perché spesso è sintomo del fatto che i campioni seguono molto bene l'andamento del robot. Ma a volte i campioni possono essere ingannati da ambienti particolarmente ripetitivi che non permettono di distinguere alcuni luoghi solo in base a dati di distanza. Il sistema di localizzazione deve essere in grado di capire se sta perdendo la stima corretta dello stato e cercare di correggersi.
Il robot si sta perdendo quando le sue percezioni divergono rispetto a quelle che dovrebbe avere se la stima del suo stato fosse corrretta. L'idea quindi è quella di confrontare i valori delle mappe delle distanze e di quella angolare alle coordinate cartesiane stimate con gli elementi del vettore osservazione.
La classe *BFL::BootStrapFilter* non prevede una possibilità di questo tipo, ma ef-

```
z : vettore delle osservazioni
distanceMap : mappa distanze
thetaMap: mappa angolare
x,y : posizione stimata
if (z[1]-distanceMap[x,y]>SOGLIA[1] and
            z[2]-thetaMap[x,y]>SOGLIA[2])
  call escapeResample;
else
  call standardResample;
```

Listato 5.3: Scelta del tipo di ricampionamento.

fettua un normale *Importance Resampling*. Conseguentemente si è resa necessaria la realizzazione di una classe *LSOFT::LocalizeParticleFilter* che eredita da *BootstrapFilter* e realizza il metodo *escapeResample*.
Chiaramente un'operazione di questo tipo non può essere frequente, per cui i valori che fungono da soglia nella decisione del tipo di ricampionamento da effettuare devono essere fissati ragionevolmente. Si deve considerare, inoltre, che un *escape* può essere deleterio se effettuato quando il robot non è perso.
Il metodo realizzato ricampiona con algoritmo standard una parte delle particelle e rigenera le altre su una gaussiana centrata nella posizione stimata mediante le particelle già create. Questo implica che un robot totalmente perso non può rilocalizzarsi

con un meccanismo di questo tipo, ma scostamenti non troppo elevati possono essere corretti piuttosto rapidamente. Variando il fattore k presente nell'algoritmo 5.6.3

```
somma_x:= 0;
for i:=1 to num_campioni/k do
begin
  campiona x[i] dalla lista delle particelle;
  somma_x:= somma_x + x[i];
end;
x_medio=somma_x/(num_campioni/k);
X(x_medio,sigma) :funzione gaussiana
for i:=num_campioni/k to num_campioni do
  campiona x[i] da X;
```

Listato 5.4: *Escape Resample*

è possibile decidere il grado di *escape* del ricampionamento: più basso è il valore che assume, maggiori saranno i campioni generati dalla funzione gaussiana.

5.7 Il threading

Smartsoft fornisce delle primitive per la gestione dei *thread* che permettono di programmare utilizzando un paradigma a oggetti anche quando si vogliono gestire più *task* nell'applicazione. Il *threading* POSIX è totalmente concepito in una logica funzionale per cui integrarlo in un software C++ può casusare numerose difficoltà che portano spesso all'utilizzo di soluzioni innaturali in una logica *object oriented*. Pur non eliminando totalmente questi problemi, Smartsoft permette di realizzare la funzione eseguita dal *thread* nel metodo *svc(void)* della classe *CHS::SmartTask*. Questo permette di utilizzare una struttura a oggetti anche per realizzare delle *thread*. *Smartask::svc(void)* viene eseguito alla chiamata di *SmartTask::open()*.

Il *threading* è stato reso indispensabile dalla necessità di leggere i valori inviati dal server, localizzare e aggiornare l'interfaccia grafica contemporaneamente. Smartsoft fornisce anche dei meccanismi di prenotazione delle risorse e di gestione delle sezioni critiche, che ha permesso ai tre *task* di comunicare tra loro.

E' stato già sottolineato il fatto che i filtri particellari sono caratterizzati da una

frequenza di aggiornamento nota e fissa. In fase di progettazione è stato, quindi, stabilito l'intervallo di tempo δt occupato da un ciclo di localizzazione. Questo significa che *LSOFT::Localizer::localizeRun()* deve essere eseguita in un tempo minore di δt e che deve essere richiamata solamente nell'istante temporale corretto. Il *thread*, cioè, si addormenta per un certo periodo e si risveglia quando l'intervallo temporale assegnato a un ciclo di localizzazione si è esaurito. Al suo risveglio accede alla risorsa critica contenente i dati odometrici elaborati dai metodi delle classi *Reader* ed emette un *acknowledge* riazzerando la media pesata dei dati letti dal *thread* di percezione. Contemporaneamente l'applicazione grafica accede alla lista di campioni e alla stima della posizione in modo totalmente asincrono, rispettando un suo intervallo di *repaint*.
La classe che regola l'interazione tra localizzazione e percezione è *LSOFT::localizeTask*. Per gestire la periodicità delle chiamate a *localizeRun()* è creato anche un *thread* temporizzatore attraverso il metodo *svc()* della classe *LSOFT::localizeTimer*.
Nel listato 4.6 viene evidenziato come il metodo tenta di accedere ad una regio-

```
class localizeTask : public CHS::SmartTask
{
  public:
    localizeTask(Localizer *localize_manager);
    ~localizeTask();
    int svc();

    private:
    Localizer* localizationManager;
};
```

Listato 5.5: Dichiarazione della classe localizeTask

ne critica condizionale chiamando il *SmartConditionMutex::wait()*. E' la funzione *svc()*, realizzata in *localizeTimer*, a chiamare *SmartConditionMutex::signal()*.

```
int localizeTask::svc(void)
{
  firstInputCond.wait();

  while (1)
    {
      robotInputCond.wait();
      contMutex.acquire();
      stepCont = 0;
      contMutex.release();
      localizationManager->localizeRun();
    }
  }
```

Listato 5.6: Metodo *svc()* per la gestione del *loop* di localizzazione.

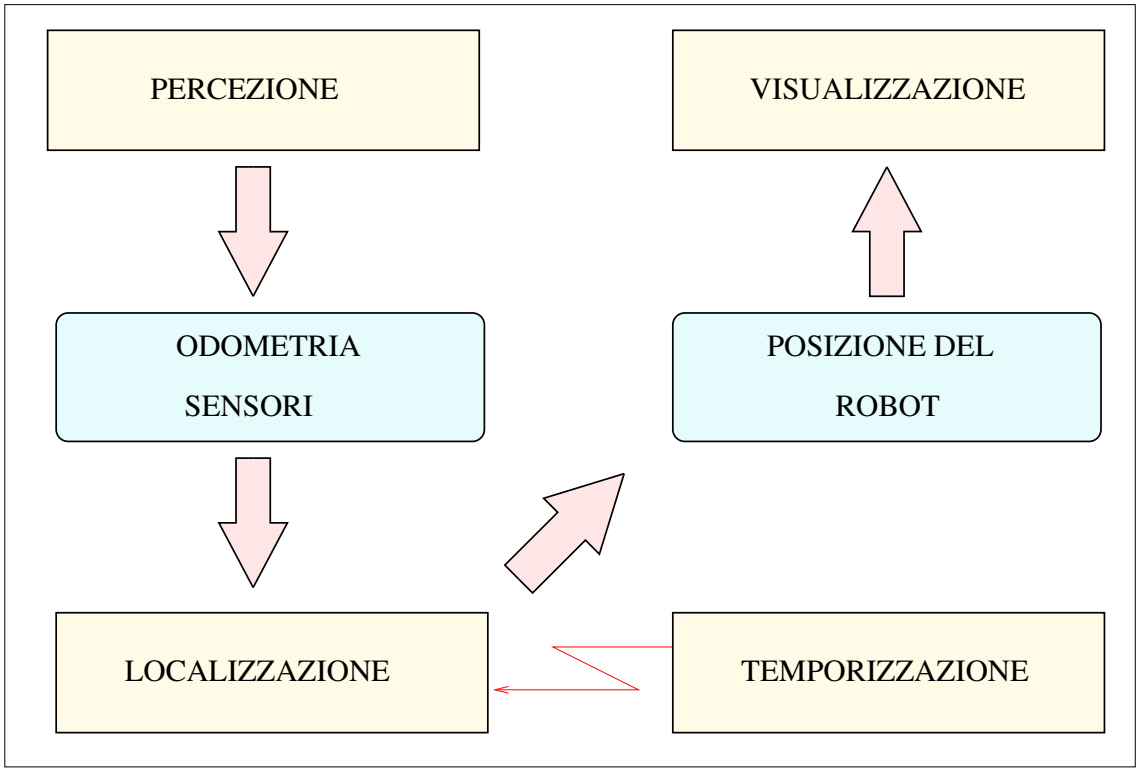

Figura 5.11: Interazione tra i *thread* nel sistema di localizzazione

5.8 Client-Server

Il server Smartsoft rende disponibili a intervalli regolari i dati provenienti dal robot ed il sistema di localizzazione è il *client* che si sottoscrive per poterli leggere ed elaborare. In realtà di *client* ne vengono creati tre: uno per ogni tipo di dato che si vuole ottenere dal robot.
A seconda di quale dispositivo si voglia conoscere lo stato è necessario collegarsi a un servizio diverso specificandone il nome all'atto della costruzione del *client*. Nel listato 4.7 vengono riportate le linee di codice relative alla creazione degli oggetti che saranno responsabili di fornire i dati provenienti dal robot.

```
CHS::SmartThreadManager *threadManager =
                            CHS::SmartThreadManager::instance();

CHS::SmartComponent *component;

component=new CHS::SmartComponent("localization",argc,argv);

//lettore dei controlli
CHS::PushTimedClient<Smart::CommBaseState>
nomadInputPushTimedClient(component,"smartNomadServer"
                                          ,"basestate");
//lettore dei sonar
CHS::PushTimedClient<Smart::CommDistanceData>
sonarPushTimedClient(component,"smartNomadServer",
                                       "distanceson");
//lettore degli infrarossi
CHS::PushTimedClient<Smart::CommDistanceData>
infraredPushTimedClient(component,"smartNomadServer",
                                         "distanceir");
```

Listato 5.7: creazione dei *client* responsabili della lettura dei valori odometrici e sensoriali. La stringa immessa come terzo parametro del costruttore indica il nome del servizio che si utilizza.

Un sistema di localizzazione viene utilizzato per monitorare la posizione del robot durante il suo movimento. Questo implica che parallelamente alla sua esecuzione verranno lanciati uno o più processi per controllare il moto del robot. L'obiettivo è quello di permettere a questi processi di assumere delle decisioni proprio in base

alla stima di stato effettuata dal localizzatore. Per questo motivo anche il sistema di localizzazione deve agire da server creando un servizio che pubblichi coordinate spaziali e orientamento del robot.
Utilizzando la classe *Smart::PushNewestServer* è possibile creare un *server* che renda disponibile il valore più recente della posizione. I *behaviour* di moto potranno sottoscriversi per creare applicazioni guidate dalle coordinate spaziali del robot.

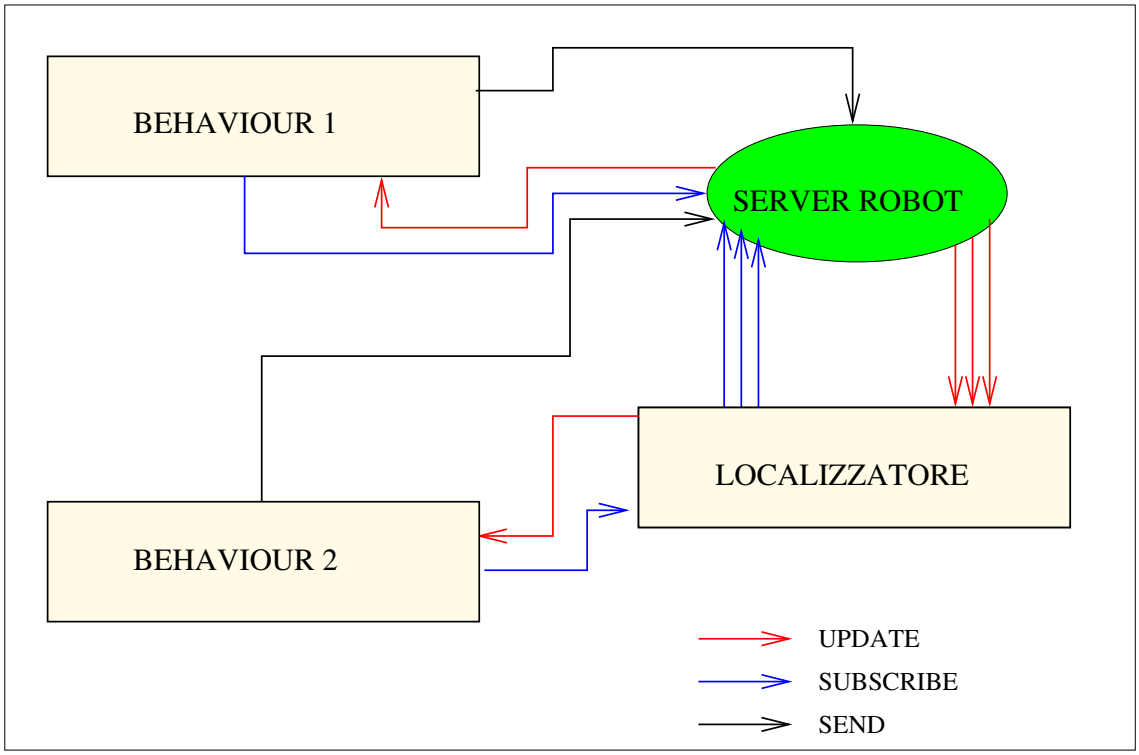

Figura 5.12: Esempio di approccio a *behaviours* che utilizza il sistema di localizzazione. I controlli del *behaviour* 1 sono guidati dai dati del robot, mentre quelli del *behaviour* 2 sono guidati dai dati inviati dal localizzatore.

Capitolo 6

Sperimentazione

6.1 Nomad 200

In generale un robot mobile può essere schematizzato in quattro parti fondamentali: sensori, attuatori, controllore e memoria. I sensori forniscono informazione riguardo

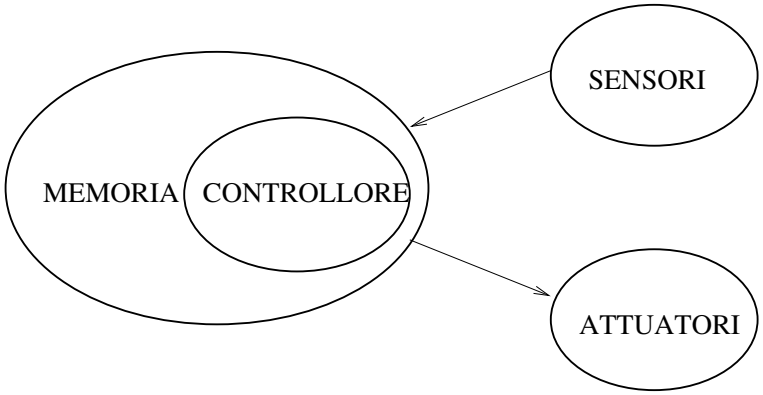

Figura 6.1: Architettura generale di un robot mobile.

l'ambiente circostante, mentre gli attuatori consentono il movimento del robot. Nel caso di robot mobili, gli attuatori più diffusi sono i motori in corrente continua. Il sistema di controllo è il settore più importante per la ricerca nel campo della robotica perché permette al progettista di definire i comportamenti del robot in funzione delle sue percezioni sensoriali [12].
La memoria è utilizzata per immagazzinare sia i programmi di controllo, che i valori forniti dai sensori.

Il robot utilizzato in questa tesi durante la fase sperimentale è *Nomad 200* [22], un robot mobile prodotto dalla società statunitense *Nomadic Technologies Inc.* della prima metà degli anni '90.
E' un robot di medie dimensioni che presenta una buona dotazione di sensori e attuatori adatti ad un utilizzo in ambienti chiusi e strutturati. Il robot contiene un'unità autonoma di percezione, elaborazione e movimento in un ingombro complessivo relativamente ridotto, avente forma ciclindrica di $50\ cm$ di diametro e $80\ cm$ di altezza (fig. 6.2).

Figura 6.2: Nomad 200

6.1.1 Movimento

Il robot è sostenuto da tre ruote di identica dimensione fissate in posizione simmetrica alla base della struttura (fig. 6.3). Il moto traslazionale è prodotto dal movimento solidale delle ruote, a cui viene trasmessa l'energia di un unico motore elettrico mediante un sistema di cinghie. La velocità rotazionale viene regolata da un secondo motore che ruota contemporaneamente l'asse di ogni ruota in modo sincrono. La parte superiore del robot(torretta) può ruotare in modo completamente indipendente rispetto alla base mediante un terzo motore posto nella parte inferiore. Le velocità

traslazionale e rotazionale massime raggiungibili sono rispettivamente $50\ cm/s$ e $45°/s$.

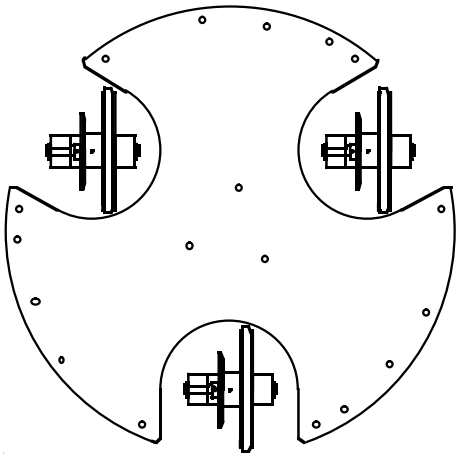

Figura 6.3: Schema delle ruote che sorreggono il robot Nomad 200.

6.1.2 Sensorialità propriocettiva

Odometria

L'elettronica di controllo dei servomeccanismi fornisce un insieme di primitive fondamentali per il calcolo delle grandezze odometriche: il sistema di elaborazione centrale contenuto all'interno del Nomad si avvale delle informazioni provenienti dagli organi di movimento per produrre una stima in tempo reale delle velocità istantanee di traslazione e rotazione del robot.

Contemporaneamente alle misure odometriche il robot fornisce alcune informazioni riguardanti il proprio stato di funzionamento, quali la verifica di eventuali condizioni di stallo dei motori e lo stato di carica delle batterie.

Bussola

Sulla sommità del robot è posizionata una bussola elettronica *KVH* C100 che rappresenta un sensore assoluto di orientamento rispetto al nord magnetico. Essa misura elettronicamente l'angolo formato da un piccolo nucleo magnetico posto in fluttuazione libera all'interno del suo contenitore risppetto rispetto alla sua posizione di riposo. Un microprocessore dedicato interpreta i dati provenienti dall'ago magnetico e produce una stima dell'angolazione della torretta del Nomad.

Tuttavia la correttezza della misura è fortemete influenzata dalla presenza di campi magnetici nelle vicinanze del robot.

6.1.3 Sensorialità eterocettiva

Sensori di contatto

Lo sviluppo geometrico di Nomad 200 è realizzato in modo da proteggere quanto più possibile le componenti più delicate dalle eventuali collisioni che possono avvenire con gli ostacoli incontrati durante il movimento. Per questo motivo la base del robot presenta un diametro maggiore rispetto alla torretta ed è rivestita da due anelli sporgenti di gomma antiurto, visibili nel dettaglio riportato in figura 6.4. All'interno dello strato di gomma sono inseriti 20 microinterruttori (10 per ogni anello) che individuano la presenza di un oggetto che si trovi a contatto col robot. Questi sensori sono detti *di pressione* o *bumper* e possono essere utilizzati per effettuare operazioni come lo spostamento di oggetti (*object pushing*), oltre che per rilevare urti.

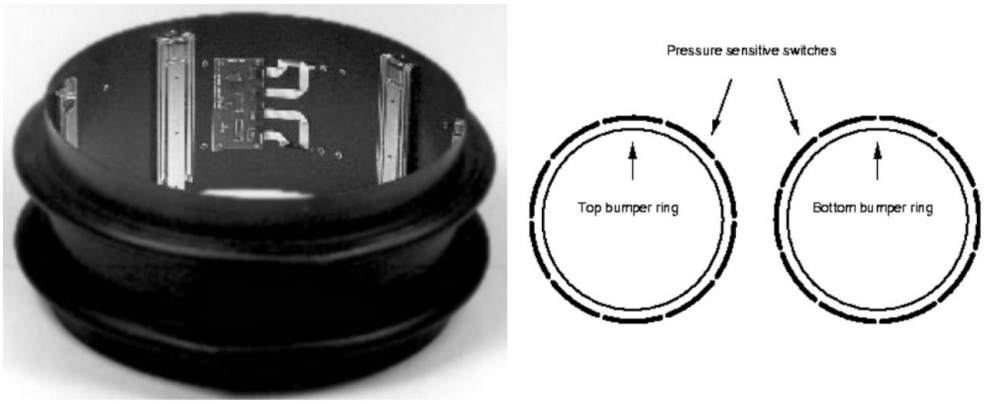

Figura 6.4: Sensori di pressione del Nomad 200.

Sensori di prossimità

Nella sommità della torretta è alloggiato un sistema sensoriale ad ultrasuoni (Sensus 200) in grado di rilevare la presenza di ostacoli a distanza nello spazio di lavoro.Esso si basa su un totale di 16 sensori sonar ultrasonici *Polaroid* controllati dalla scheda dedicata *Polaroid 6500*; il modulo *Sensus 200*, illustrato in figura 6.5, consente di misurare correttamente distanze nell'intervallo $30 \div 600\ cm$ con una precisione stimata pari all'1%.

Il *beam pattern* riportato in figura 6.5 (realizzato con frequenza 49.4 kHz) mostra

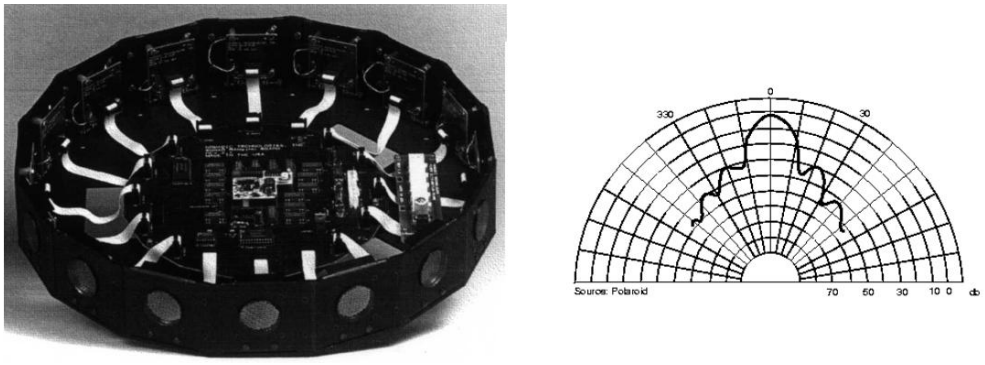

Figura 6.5: Sensori ultrasonici del Nomad 200.

come un singolo trasduttore sonar sia in grado di rilevare efficacemente oggetti che si trovano entro un cono massimo di circa 25°: i sensori, che risultano sfasati tra loro di 22.5°, riescono quindi a spaziare l'intero perimetro del robot.

I sonar utilizzati sfruttano la misura del *tempo di volo* di un treno d'impulsi per identificare la distanza da un ostacolo. Lo stesso trasduttore elettrostatico produce il segnale e ne riceve l'eco di ritorno. Questo rende impossibile la rilevazione di oggetti molto vicini a causa di fenomeni di risonanza.
Il circuito stampato a cui sono connessi i 16 sonar realizza un *multiplexing* che consente di collegare alternativamente ogni sonar con l'unico circuito di comando Polaroid posto al centro della scheda: è possibile abilitare selettivamente, via software, i singoli sensori e decidere l'ordine progressivo di sparo che viene ripetuto ciclicamente dall'hardware dedicato. Assieme al *firing pattern* è possibile specificare anche la durata del ciclo di ricezione che viene eseguito prima della commutazione del circuito di comando al sonar successivo: i valori bassi di tale intervallo consentono di eseguire un numero maggiore di misure nell'unità di tempo, ma l'utilizzo di tempi dilatati permette di rilevare echi provenienti da oggetti distanti, che, altrimenti, non ritornerebbero alla sorgente prima dell'invio dell'impulso successivo. L'intervallo di sparo può variare da un minimo di 4ms ad un massimo di 1s; il valore minimo consente di individuare ostacoli ad una distanza di circa $70\ cm$.

L'utilizzo di un solo circuito di comando per l'intero anello di sonar impedisce di attivare contemporaneamente più di un sensore alla volta; questa soluzione però, oltre a consentire un notevole risparmio di hardware, è di fatto implicita nel-

l'uso di sensori di questo tipo, che, soffrendo fortemente di problemi di interferenza reciproca, sono generalmente inadatti ad un uso simultaneo.

Per misurare la distanza di oggetti particolarmente vicini al robot, la Nomadic ha messo a disposizione il modulo *Sensus 300*, inserito nella parte inferiore della torretta, che dispone di 16 emettitori/ricevitori di radiazione infrarossa in grado di rilevare ostacoli fino ad una distanza massima di circa $60\ cm$.
Ogni sensore è formato da due diodi LED emettitori ed un fotodiodo rilevatore: la distanza viene misurata a partire dall'intensità della radiazione riflessa. Questa viene modulata opportunamente dai diodi emettitori per ridurre l'effetto del rumore presente nell'ambiente.
L'intensità della radiazione misurata dal fotodiodo è funzione della distanza della

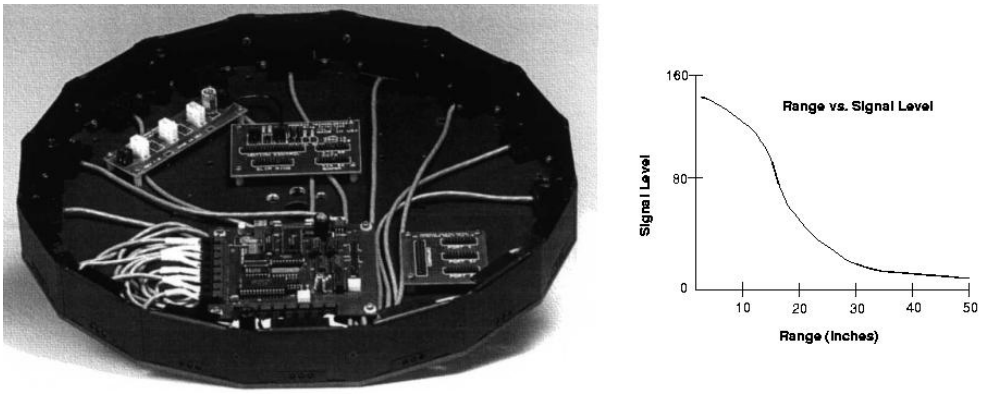

Figura 6.6: Sensori a emissione di radiazione infrarossa del Nomad 200.

superficie riflettente dell'ostacolo, come evidenzia il grafico di figura 6.6, ma dipende fortemente anche dal grado di illuminazione infrarossa dell'ambientee dal grado di riflettività delle superfici. Per rendere significativo il valore acquisito, ogni ciclo di lettura prevede l'effettuazione alternata di due misure consecutive con diodo di emissione acceso e spento. Questa strategia consente di produrre una misura differenziale che riduce l'effetto del rumore. Una fase di calibrazione dei sensori in funzione dell'ambiente di utilizzo resta comunque di particolare importanza.
La misura analogica dell'intensità di radiazione viene convertita in segnale digitale mediante un *VCO*(*Voltage Controlled Oscillator*) che genera un'onda quadra di frequenza proporzionale al valore misurato. Un contatore ad alta frequenza provvede alla misurazione del periodo di questa oscillazione al fine di determinare una stima

digitale del valore desiderato. La misura complessiva richiede un tempo pari a circa 2.5 ms.

6.2 Ambiente di sperimentazione e mappatura

L'ambiente di sperimentazione in cui è stato svolto il *testing* del sistema di localizzazione è il corridoio del piano terra del Dipartimento di Ingegneria dell'Informazione. La planimetria, che qui viene riproposta, evidenzia come l'ambiente sia in certi tratti molto ripetitivo, ma non manchino degli elementi che caratterizzano certi luoghi rispetto ad altri. Nonostante sia la mappa che la posizione vengano rappresentate metricamente, come nell'approccio *landmark based*, anche nell'approccio metrico elementi che producono discontinuità nelle osservazioni del robot permettono delle correzioni sulla distribuzione delle particelle.
La costruzione della mappa è avvenuta effettuando un rilievo metrico dell'ambiente. Utilizzando il programma *AutoCad 2004* i dati raccolti sono stati restituiti con una planimetria molto accurata. Il risultato ottenuto è stato esportato in formato *pnm*.

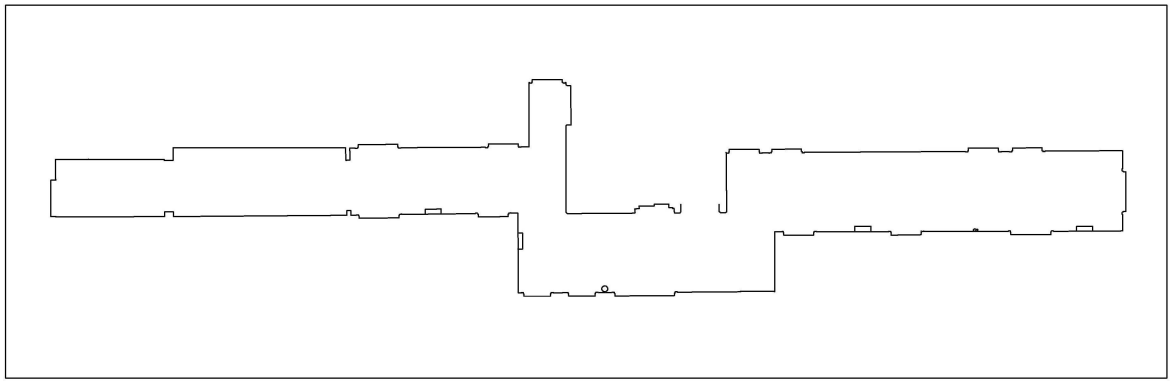

Figura 6.7: Planimetria dell'ambiente di sperimentazione.

La scala del disegno in formato elettronico è $1:100$, in cui un pixel rappresenta $1\ cm^2$. Anche la distinzione tra *free space* e spazio non navigabile è stata realizzata manualmente. Nel *free space* è stata inserita parte della rampa di scale: i gradini risultano pressochè invisibili per i sensori a emissione di radiazione infrarossa e per

quelli a emissione di impulsi ultrasonici perché, salendo, si allontanano dalla loro portata.

6.3 Simulazione

6.3.1 Modello del Nomad 200

```
define nom_auto_sonar sonar
(
  scount 16
  spose[0] [0.250000 0.000000 0.000000]
  spose[1] [0.230970 0.095671 22.500000]
  spose[2] [0.176777 0.176777 45.000000]
  spose[3] [0.095671 0.230970 67.500000]
  spose[4] [0.000000 0.250000 90.000000]
  spose[5] [-0.095671 0.230970 112.500000]
  spose[6] [-0.176777 0.176777 135.000000]
  spose[7] [-0.230970 0.095671 157.500000]
  spose[8] [-0.250000 0.000000 180.000000]
  spose[9] [-0.230970 -0.095671 202.500000]
  spose[10] [-0.176777 -0.176777 225.000000]
  spose[11] [-0.095671 -0.230970 247.500000]
  spose[12] [-0.000000 -0.250000 270.000000]
  spose[13] [0.095671 -0.230970 292.500000]
  spose[14] [0.176777 -0.176777 315.000000]
  spose[15] [0.230970 -0.095671 337.500000]
)
define nomad_auto2dx position
(
  shape "circle"
  size [0.500000 0.500000]
  offset [0.0 0.0]
  nom_auto_sonar()
  power()
)
```

Listato 6.1: Modello del Nomad 200 contenuto nel file *nomad.inc*.

Per poter costruire un ambiente di simulazione per il software *Stage* è necessario creare un file di estensione *world*. Esso contiene le informazioni per caricare la mappa e il riferimento a un altro file di estensione *inc* contenete il modello del robot.

Stage utilizza unicamente modelli bidimensionali per la rappresentazione degli oggetti. Il modello del robot viene costruito nel file *.inc* utilizzando coordinate spaziali in scala $1 : 1$. Nel listato 6.1 viene presentato *nomad.inc*, contenente il modello del Nomad 200 costruito.
Il robot è di forma cilindrica, per cui viene reso in 2D come una circonferenza di diametro $0.5\ m$. Sono anche specificate le posizioni dei 16 sonar sul perimetro del modello spaziale. Il file *world* contiene la porta a cui si connetterà con *Player*, oltre ai riferimenti alla mappa, al robot, alla posizione e all'orientazione iniziale di quest'ultimo. Il risultato ottenuto è mostrato in figura 6.8. Con la primitiva *power()* si specifica che il modello è dinamico, ovvero può ricevere dei comandi di velocità che, quindi, ne causano il moto.

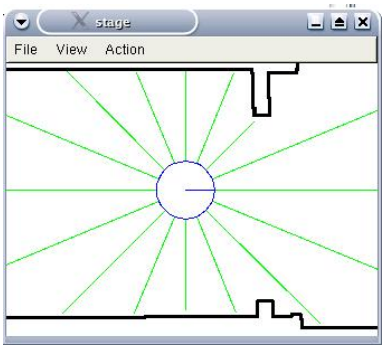

Figura 6.8: Visualizzazione del modello del Nomad 200 in ambiente di sperimentazione mediante *Stage*.

6.3.2 Prove sperimentali

Per poter sperimentare il localizzatore è stato preliminarmente costruito un programma che realizzasse un algoritmo di moto. Tale algoritmo, per quanto riguarda la parte simulativa, realizza una navigazione di tipo *obstacle avoidance*: ogni volta che il robot incontra un ostacolo varia la sua direzione al fine di avere davanti a sé

Rumore	**Media**	**Varianza**
x (cm)	0	3
y (cm)	0	1.5
θ (rad)	0	0.0001
distanza misurata (cm)	0	20
angolo misurato (rad)	0	0.35

Tabella 6.1: Media e deviazione standard degli errori che affliggono la stima delle coordinate di stato e le componenti del vettore osservazione nel caso di simulazione

spazio libero. In un ambiente sperimentale come quello utilizzato in questa tesi un algoritmo di questo tipo permette di esplorare tutto l'ambiente con frequenti cambi di direzione, i quali costituiscono un fattore critico per il test del localizzatore.
Sfruttando la libreria *PlayerClient* è stato possibile monitorare l'evoluzione dello stato reale del robot e confrontare il dato ottenuto con la stima effettuata dal localizzatore.
Potendo effettuare numerose prove, si è deciso di verificare le prestazioni del sistema a diverse velocità di navigazione, tutte inferiori a $20\ cm/s$. Un altro fattore di valutazione è stato il confronto dei risultati ottenuti eseguendo la localizzazione con la procedura di *escape resample*, illustrata nel precedente capitolo, e quelli ottenuti senza il suo utilizzo.

Numerose prove hanno permesso di stabilire le caratteristiche del rumore sul sistema e sulle percezioni in simulazione(tab. 6.1). Alla componente angolare del vettore di osservazione è stata assegnata una varianza molto maggiore, in proporzione, rispetto a quella data alla componenente di distanza, essendo i dispositivi sensoriali separati l'uno dall'altro da un angolo di 22.5°. In simulazione l'errore sull'odometria è piccolo, ma non nullo: l'accesso ai valori delle velocità rotazionale e traslazionale del robot avviene via *socket* e, quindi, in modo discontinuo e a intervalli irregolari, in relazione alla percentuale di utilizzo della potenza del calcolatore. L'elaborazione su 120 particelle si è dimostrata ottima per non impegnare troppe risorse, ma anche per non perdere variabilità nella distribuzione spaziale dei campioni. L'algoritmo di localizzazione ha mostrato una maggior precisone quando non supportato dalla procedura *escape*. Tale risultato, però, non ha sminuito l'utilità di questo ulteriore *feedback* poichè l'algoritmo base si è dimostrato meno robusto di quello coadiuvato da *escape*. Quando il moto del robot assume comportamenti

Velocità(cm/s)	**Escape**	**Err. medio (cm)**	**Dev. std (cm)**
10	no	16.07	10.42
10	si	17.23	12.62
15	no	19.5	16.33
15	si	22.44	19.87

Tabella 6.2: Risultati ottenuti mediante prove in ambiente di simulazione.

fortemente variabili il sistema di localizzazione può perdere la stima corretta. Su 10 prove eseguite a bassa velocità (minore o uguale a $15\ cm/s$) l'algoritmo base fallisce circa il 15% delle volte rispetto al 10% per la versione dotata di *ecape*, causando, in tali occasioni, la totale perdita della posizione del robot. Questi valori sono dati considerando prove sperimentali in cui il robot non abbia avuto comportamenti anomali. Per comportamenti anomali s'intendono moti vibratori, in cui la velocità di rotazione cambia ripetutamente e a piccoli intervalli di tempo il verso.
Naturalmente l'ambiente influisce sul tipo di traiettorie descritte dal robot: ci sono parti del'ambiente di sperimentazione che hanno messo in crisi il localizzatore più di altre: il piccolo corridoio rappresentato sulla parte superiore della planimetria ha sempre innescato moti anomali sul robot e solo con algoritmo dotato di *escape resampling* è stato possibile mantenere il robot localizzato in alcune prove.

In figura 6.9 viene mostrato un caso in cui il sistema di localizzazione ha evidenziato una buona robustezza ai moti oscillatori del robot, esplorando anche la parte critica dello spazio. Il veicolo parte dal lato sinistro della planimetria, entra nella parte più stretta dello spazio e ne esce ancora localizzato. Si può notare che in varie zone la traiettoria stimata si allontana anche di $45 - 50\ cm$ da quella corretta in seguito alle turbolenze create dal moto assunto dal robot nella porzione stretta del *free space*, ma è anche evidente che si riallinea allo stato corretto col procedere del moto. Nella parte finale il robot è allineato con un errore inferiore a $12\ cm$. L'algoritmo di moto utilizzato in simulazione è intrinsecamente portato a creare moti anomali: ogni volta che incontra degli ostacoli effettua dei controlli che causano brusche sterzate. Costruendo un algoritmo che cerchi di limitare le rapide escursioni della grandezze odometriche, le percentuali di fallimento del localizzatore sarebbero molto più contenute.
Si deve sottolineare, infine, che tutte le discontinuità osservabili nel tracciato di co-

lore giallo in figura 6.9 sono correzioni effettuate dall'*escape resample*, tutte andate a buon fine, con una corrispondente riduzione dell'errore tra i 10 e i 25 cm. Ad

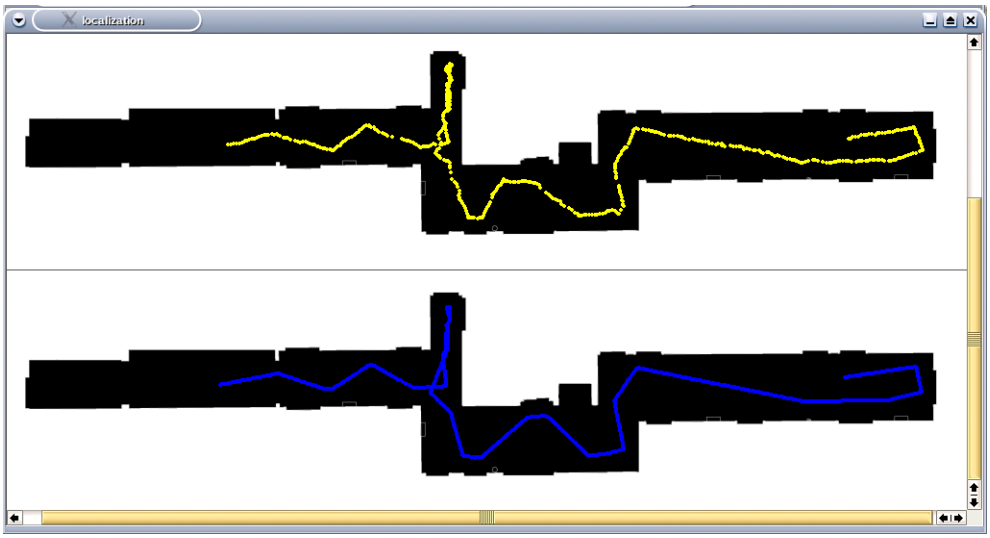

Figura 6.9: Visualizzazione di una navigazione simulata con accesso ad un ambiente ristretto. In blu è indicata la traiettoria reale del robot simulato, mentre in giallo è indicata l'evoluzione dello stato stimato.

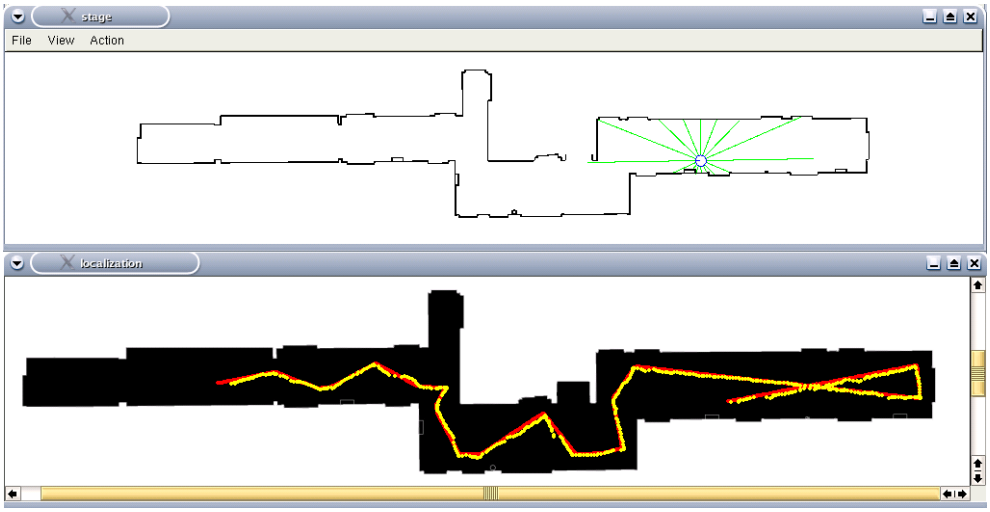

Figura 6.10: Visualizzazione di una navigazione simulata senza moto anomalo. Sulla prima finestra è mostrata l'interfaccia grafica che visualizza il robot nell'ambiente, mentre nella seconda sono indicate la traiettoria reale e quella stimata.

alte velocità le percentuali aumentano e l'*escape* non ha più un effetto positivo: la soluzione ideale è eseguirlo a basse velocità e, quindi, a bassa rumorosità sui valori percettivi.
Nella figura 6.10 viene mostrata una verifica sperimentale senza moti anomali, in cui appare evidente come la posizione stimata sia coincidente con quella effettiva nelle zone a bassa variabilità dell'orientamento e mantenga una stima affidabile anche nelle zone più critiche dello spazio.

6.4 Sperimentazione con Nomad 200

In ambiente reale le fonti di rumore aumentano fortemente rispetto al caso studiato in simulazione. Gli impulsi emessi dai dispositivi ultrasonici vengono in parte assorbiti e in parte riflessi dalle pareti dell'ambiente. Le porte di colore scuro, invece, assorbono la radiazione emessa dai dispositivi infrarossi impedendo di misurare la distanza che le separa dal robot. Il pavimento sul quale il veicolo naviga ha caratteristiche disuniformi di cui si è dovuto tenere conto nel momento in cui si sono determinate le caratteristche dei rumori sul modello del sistema. Infine, la meccanica stessa del *Nomad 200* introduce ulteriori elementi di disturbo.

6.4.1 Algoritmo di moto

La scelta dell'algoritmo di moto è stata fatta cercando di evitare fenomeni oscillatori o vibratori dell'orientazione del robot. Inizialmente si è voluto osservare qualitativamente l'andamento del sistema per stimare le caratteristiche del rumore, per cui l'algoritmo che comanda il robot era indipendente dai valori forniti dal localizzatore. *smartSimpleCentClient*[12] è il nome dell'eseguibile che cerca di comandare il robot in modo da farlo rimanere al centro del corridoio durante il suo moto. Le traiettorie descritte dal robot sono più dolci di quelle tracciate in simulazione, fatto che sicuramente ha rappresentato un vantaggio per la riuscita del *position tracking*.
Il programma si collega al server *Smartsoft* sottoscrivendosi sia ai valori misurati dai sonar che a quelli misurati dagli infrarossi. La velocità di navigazione varia da un massimo di $15\ cm/s$ a un minimo di $8\ cm/s$.

Rumore	**Media**	**Varianza**
x (cm)	0.0	5.0
y (cm)	0.0	2.0
θ (rad)	-0.0004	0.0001
distanza misurata (cm)	0	30
angolo misurato (cm)	0	0.35

Tabella 6.3: Media e deviazione standard degli errori che affliggono la stima delle coordinate di stato e le componenti del vettore osservazione nel caso reale

Dopo aver stimato le catteristiche del rumore, l'algoritmo è stato modificato al fine di valutare quantitamente la validità del localizzatore. La soluzione trovata è stata quella di far fermare il robot in zone prestabilite, ovvero a determinate altezze del corridoio(fig.6.11). Data la forma allungata dell'ambiente di sperimentazione, si è deciso di far fermare il veicolo in corrisponenza di determinate coordinate sull'asse delle ascisse, fino all'invio da parte dell'utente di un comando per far ripartire il robot. Questo tipo di operazione ha permesso di effettuare dei rilievi metrici per stabilire volta per volta l'esatta locazione del *Nomad 200* sulla mappa metrica.

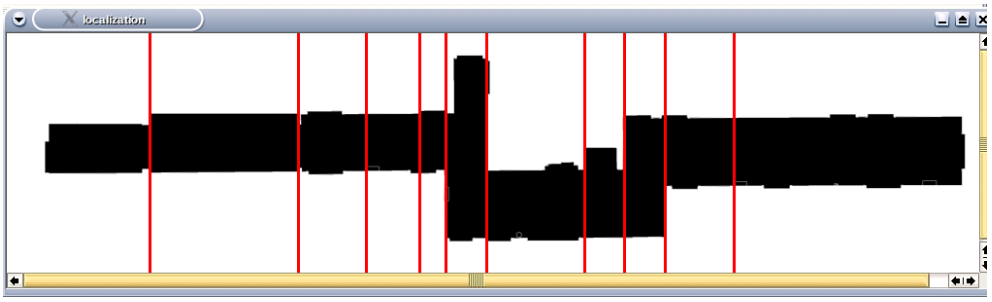

Figura 6.11: Le linee verticali indicano i punti di coordinate x in cui far fermare il robot.

6.4.2 Parametri rumorosi

L'esecuzione di numerose prove sperimentali ha permesso la stima del rumore sul sistema e sulle percezioni (Tabella 6.3). Aumentare il rumore di sistema implica anche una distribuzione spaziale dei campioni più ampia. Questo fa perdere precisione nella stima, ma permette di migliorare la robustezza dell'algoritmo.

Come evidenziato in figura 6.12, i parametri in tabella 6.3 hanno permesso di mantenere una buona stima della posizione del robot lungo il percorso su tutto il corridoio. I valori trovati non sono generici, ma specifici dell'ambiente in cui il robot deve navigare. Nel caso di questa tesi, lo spazio si sviluppa orizzontalmente; ne consegue che l'errore sulle x è maggiore perché il robot cambia anche di $21\ m$ la coordinata sulle ascisse della sua posizione, ma non più di $3.5\ m$ quella sulle ordinate. L'errore sull'orientazione è dovuto a una lieve tendenza da parte di *Nomad 200* ad orientarsi verso destra durante lunghi tratti di navigazione a velocità sostenuta.
L'errore di misura determina il peso delle particelle. La sua varianza determina l'apertura della gaussiana che descrive il disturbo. Il valore di $30cm$ non è corretto per descrivere la varianza dell'errore sulla distanza, ma permette di enfatizzare i valori particolarmente vicini a quello corretto. Per l'angolo non è stato possibile fare considerazioni analoghe perché la differenza tra il valore ottenuto dalla mappa angolare e quello misurato presentano spesso differenze rilevanti. Modellare una gaussiana troppo stretta anche per l'angolo, dunque, avrebbe abbassato l'importanza di questa misura perché troppo spesso avrebbe inciso debolmente nella determinazione del peso (eq. 3.16).

Figura 6.12: Visualizzazione di un *position tracking* eseguito sul robot reale.

6.4.3 Prove sperimentali

Sono state eseguite cinque verifiche sperimentali con l'algoritmo di moto che prevede l'arresto del robot in corrispondenza di valori prestabiliti delle ascisse. Grazie a queste prove si è potuta stabilire la reattività del sistema e l'errore compiuto dal localizzatore.
In tabella 5.4 sono indicate le coordinate sulle x in corrispondenza delle quali il

Campione	1	2	3	4	5	6	7	8	9	10
Ascissa	526	1073	1324	1522	1617	1769	2131	2276	2428	2682

Tabella 6.4: Posizioni di sosta del robot (acissa in cm), corrispondenti ai segmenti sulla mappa di figura 6.11.

robot viene fermato dal programma di moto *smartSimpleCentClient*, il quale setta la sua velocità a 0 finchè l'utente non preme un tasto per far ripartire il veicolo. Le misurazioni sono state eseguite cercando di non disturbare il localizzatore: anche quando il robot è fermo il sistema esegue il ciclo di localizzazione, per cui avvicinarsi ad esso può provocare forti variazioni percettive che peggiorano la stima dello stato.

Il campione 1 è la posizione iniziale. Il robot si trova in una posizione di ascissa molto maggiore rispetto a 526 prima dell'inizio del suo moto. La prima misurazione sul campo viene fatta rispetto alla coordinata stabilita nella tabella 5.4, ma non è rilevante per stabilire la reattività del sistema dato che il robot è fermo all'inizio della prova.

In tabella 5.5 sono indicate le posizioni reali a cui il robot si è fermato in ogni prova, seguite da quelle stimate dal localizzatore. In figura 6.13 sono state rappresentate sulla mappa le posizioni stimate dal localizzatore (colore giallo) e quelle reali (colore rosso). Risulta evidente che il robot si sia fermato sempre dopo l'ascissa prestabilita, sia per le latenze nella comunicazione tra i processi *client* e quelli *server*, che per i ritardi nell'esecuzione meccanica dei comandi inviati al robot. Un altro motivo può essere il tempo δt di esecuzione di un ciclo di localizzazione che, nel caso più sfortunato, può causare una fermata ritardata di circa $6\ cm$ con velocità di navigazione pari a $15\ cm/s$ e $\delta t = 0.4\ sec$.

Le prove eseguite hanno evidenziato un comportamento uniforme tenuto dal sistema di localizzazione: se il robot non effettua forti cambiamenti di direzione allora la stima si avvicina molto al valore corretto, viceversa, quando le traiettorie descritte sono caratterizzate da numerose curve, la stima diverge dalla posizione reale.

I risultati sono stati ottenuti senza l'utilizzo dell'*escape resample*. Tale scelta è dovuta al fatto che le situazioni in cui le percezioni risultano incongruenti con lo stato stimato e che, quindi, causerebbero un *escape*, spesso non sono dovute a una stima scorretta della posizione, ma a percezioni troppo corrotte dal rumore. Se le

Prova	1	2	3	4	5
$(x_1, y_1)_{real}$	626 , 428	631 , 448	615 , 413	620 , 433	623 , 436
$(x_2, y_2)_{real}$	1073 , 399	1111 , 399	1114 , 421	1075 , 401	1098 , 400
$(x_3, y_3)_{real}$	1329 , 421	1341 , 408	1344 , 385	1330 , 440	1334 , 445
$(x_4, y_4)_{real}$	1533 , 362	1530 , 393	1516 , 405	1530 , 400	1525 , 425
$(x_5, y_5)_{real}$	1637 , 409	1618 , 404	1642 , 405	1628 , 415	1621 , 400
$(x_6, y_6)_{real}$	1786 , 602	1794 , 607	1773 , 607	1780 , 605	1777 , 598
$(x_7, y_7)_{real}$	2181 , 598	2162 , 611	2180 , 613	2177 , 584	2166 , 610
$(x_8, y_8)_{real}$	2321 , 591	2288 , 613	2312 , 606	2302 , 580	2302 , 606
$(x_9, y_9)_{real}$	2454 , 479	2454 , 446	2454 , 451	2447 , 452	2479 , 465
$(x_{10}, y_{10})_{real}$	2687 , 429	2707 , 425	2707 , 425	2685 , 425	2703 , 427
$(x_1, y_1)_{estimated}$	634 , 441	611 , 430	612 , 433	629 , 438	633 , 442
$(x_2, y_2)_{estimated}$	1078 , 409	1091 , 401	1093 , 406	1089 , 412	1080 , 416
$(x_3, y_3)_{estimated}$	1338 , 451	1337 , 372	1334 , 392	1342 , 470	1355 , 460
$(x_4, y_4)_{estimated}$	1530 , 410	1530 , 385	1526 , 412	1544 , 423	1550 , 407
$(x_5, y_5)_{estimated}$	1632 , 406	1628 , 382	1627 , 402	1639 , 411	1621 , 410
$(x_6, y_6)_{estimated}$	1770 , 605	1781 , 594	1773 , 682	1775 , 600	1779 , 652
$(x_7, y_7)_{estimated}$	2142 , 605	2141 , 583	2131 , 604	2139 , 610	2141 , 615
$(x_8, y_8)_{estimated}$	2278 , 617	2282 , 590	2283 , 616	2280 , 615	2288 , 623
$(x_9, y_9)_{estimated}$	2437 , 402	2436 , 395	2443 , 388	2439 , 401	2455 , 392
$(x_{10}, y_{10})_{estimated}$	2688 , 435	2695 , 447	2685 , 433	2686 , 430	2690 , 444

Tabella 6.5: Coordinate reali espresse in cm e stimate sulle cinque verifiche sperimentali.

Prova	**Errore medio (cm)**	**Deviazione standard (cm)**
1	30.296	22.801
2	32.820	13.557
3	32.856	21.196
4	32.188	29.491
5	30.418	19.127
Media sul tot.	32.116	21.23

Tabella 6.6: Media e varianza dell'errore di localizzazione compiuto nelle cinque prove.

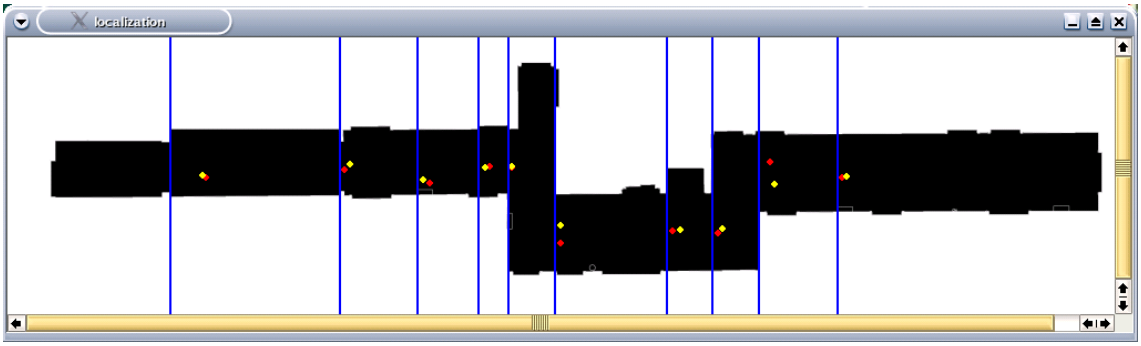

Figura 6.13: Prova sperimentale con errore avente media $30.3\ cm$ e deviazione standard $22.8\ cm$.

particelle si facessero divergere in una situazione di questo tipo i pesi a loro assegnati sarebbero errati, perché frutto di osservazioni disturbate. Verifiche sperimentali hanno dimostrato che il robot in corrispondenza dei punti 7 e 8, cioè nella zona più larga del corridoio, si è sempre perso utilizzando la procedura *escape resample*. Il fallimento di questo metodo di correzione vale, però, nel caso specifico di localizzazione mediante utilizzo di dispositivi a impulsi ultrasonici. Tentativi di correzione sono andati a buon fine quando il robot si è trovato vicino alle pareti e ha potuto, quindi, utilizzare una fonte sensoriale più precisa come gli infrarossi. Le prove in simulazione e tratti rilevanti di prove con *Nomad 200* hanno evidenziato prospettive interessanti che suggeriscono di non abbandonare lo studio di questo tipo di ricampionamento correttivo, soprattutto se si può disporre di dispositivi sensoriali ad elevata precisione come i laser scanner.

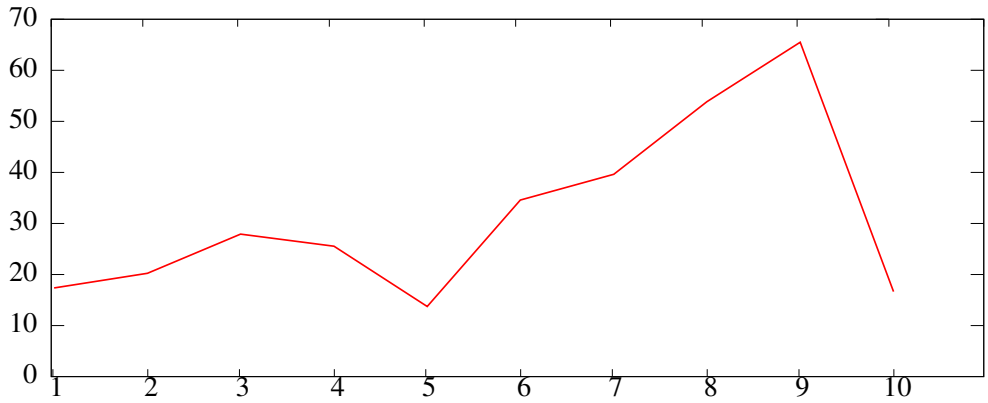

Figura 6.14: Evoluzione dell'errore in funzione del punto campionato.

Pur non utilizzando un meccanismo correttivo complementare all'assegnazione dei pesi alle particelle, il rumore sull'odometria e sulle percezioni ha aumentato la retroazione effettuata dal ciclo di localizzazione del filtro bayesiano. Maggiore è l'errore, infatti, e più i campioni selezionati con metodo Monte Carlo assumeranno stati diversi: quando l'errore è alto, i pesi dei campioni, che sono il fattore discriminante del ricampionamento, assumono valori poco differenti tra loro. Questo fatto diminuisce la predominanza di un numero ristretto di particelle e favorisce la sopravvivenza di un numero più elevato di particelle aventi stati diversi tra loro. La stima diviene più imprecisa, ma l'algoritmo aumenta la sua robustezza perché maggiore è l'errore, più forte diviene la retroazione.
In figura 6.14 viene evidenziato come nel tratto iniziale l'errore medio compiuto dal localizzatore sia sempre inferiore ai $30\ cm$, per poi aumentare nel tratto in cui i cambiamenti di direzione aumentano la loro ampiezza e lo spazio intorno al robot incrementa le proprie dimensioni. Questi due fattori disturbano entrambi sia i dati odometrici rilevati (fig.5.2), che le percezioni sensoriali (par.3.2.1). L'errore massimo è raggiunto in corrispondenza del 9° campionamento, al termine, cioè, del tratto critico dell'ambiente di sperimentazione. Se, infatti, frequenti mutamenti dell'orientazione causano errori nel rilevamento dei controlli, anche i dispositivi ultrasonici hanno difficoltà nel ricevere il segnale riflesso dall'ostacolo. Va evidenziato come il ritorno a un moto più uniforme permetta all'algoritmo di correggere la stima riportando l'errore, arrivato oltre $60\ cm$, sotto i $20\ cm$ (10° campione).

Capitolo 7

Conclusioni

Questa tesi si è proposta di realizzare un modulo di localizzazione e, in particolare, di *position tracking*, per un robot mobile su una mappa metrica. Tale funzionalità è, infatti, essenziale per la navigazione autonoma in ambiente parzialmente osservabile. In particolare è il presupposto per sviluppare algoritmi di pianificazione del moto. Nello svolgimento della tesi sono state privilegiate tecniche probabilistiche che consentono di affrontare il problema della stima dello stato partendo dai modelli descriventi la cinematica del robot e delle sue percezioni sensoriali. La costruzione di mappe che forniscono le componenti del vettore osservazione per ogni cella dell'*occupancy grid* ha permesso di risparmiare risorse computazionali e ha ridotto notevolmente il tempo necessario per ottenere le informazioni richieste dal ciclo di localizzazione.

Lo studio approfondito degli strumenti software già disponibili ha permesso di sviluppare il programma di localizzazione su librerie testate e, conseguentemente, caratterizzate da un alto livello di affidabilità. La scelta di un algoritmo probabilistico è risultata necessaria a causa della forte rumorosità dei dispositivi sensoriali in dotazione sul robot utilizzato in fase di sperimentazione. La scelta dell'utilizzo del filtro di tipo *bootstrap* è motivata sia dalla costruzione più agevole della probabilità $p(x_k|x_{k-1}, u_{k-1})$ rispetto alla $p(x_k|x_{k-1}, u_{k-1}, z_k)$ utilizzata da altri filtri particellari, che dal ridotto utilizzo delle risorse di calcolo nella fase di predizione del ciclo di localizzazione.

Il sistema realizzato ha dimostrato di poter eseguire un buon *position tracking*

utilizzando fonti sensoriali con alto livello di rumorosità. L'utilizzo di un modello a *thread* ha permesso la separazione della fase di comunicazione col robot e quella di elaborazione dei dati, migliorando di conseguenza la fase predittiva basata sull'odometria. Inizialmente non era stata prevista tale separazione e il robot si perdeva rapidamente non appena le variazioni del suo orientamento aumentavano in ampiezza e frequenza.
Durante la fase sperimentale l'algoritmo ha dimostrato buona capacità di correzione e dei comportamenti riconducibili agli studi fondati su basi biologiche già compiuti nell'ambito della robotica mobile in cui la visione dell'ambiente è condizionata dal tipo di azioni che si compiono. Questo implica che anche per un essere capace di comportamenti intelligenti è difficile avere una percezione chiara dell'ambiente nel caso in cui esso si muova rapidamente e con repentini cambi di direzione. Allo stesso modo, quando il *Nomad 200* ha compiuto traiettorie curvilinee, ha avuto difficoltà nel localizzarsi, ma quando l'algoritmo di moto ha imposto al robot di fermarsi, in pochi secondi la stima della posizione si è corretta perché il rumore sulle osservazioni è diminuito. Questo evento suggerisce possibili sviluppi che permettano all'algoritmo di moto di interagire col sistema di localizzazione, rallentando il moto per permettere al robot di "guardarsi intorno" e localizzarsi correttamente. Come è stato sottolineato nel precedente capitolo, l'errore di stima passa da $60\ cm$ a $20\ cm$ nel momento in cui il profilo delle velocità diventa più uniforme e l'assegnazione del peso alle particelle diviene più corretta perché basata su osservazioni meno errate.

Il problema della localizzazione globale è stato affrontato in fase di simulazione, ma con risultati piuttosto scarsi. Durante le verifiche sperimentali si nota che dopo la fase di generazione uniforme di particelle sul *free space*, il primo ricampionamento addensa le particelle nelle zone più probabili dello spazio. Quasi sempre uno degli addensamenti è nella posizione corretta, ma, senza nessun meccanismo di controllo, la maggior parte delle volte sono altre le particelle alle quali l'algoritmo di condensazione assegna il livello più alto di credibilità. Anche quando si esegue il *position tracking* col robot reale può capitare che, nel caso in cui la posizione sia inizializzata con coordinate troppo inesatte, le particelle generate su una funzione gaussiana al primo campionamento convergano nel punto sbagliato. Questo comportamento dell'algoritmo è dovuto al fatto che, mentre le mappe delle distanze sono costrui-

te su uno spazio ”quasi” continuo, i dispositivi sensoriali danno una visione molto più discreta dell'ambiente circostante. Eseguire la localizzazione globale necessita dunque di dispositivi sensoriali che diano una visione più continua dello spazio osservabile e la possibilità di conoscere il proprio orientamento mediante una bussola che, nel caso del *Nomad 200*, non ha un livello di affidabilità sufficiente.

Possibili sviluppi che consentano di arrivare ad una *global localization* sono l'utilizzo di ulteriori fonti di informazione che permettano di limitare il campo di ricerca, la realizzazione di meccanismi correttivi come *escape resample* oppure la creazione di mappe contenenti le percezioni reali del robot. In questo caso si potrebbe tentare un approccio *SLAM* (*Simultaneous Localization And Mapping*).

Dal punto di vista algoritmico, il ciclo di localizzazione potrebbe essere migliorato trovando l'espressione di $p(x_k|x_{k-1}, u_{k-1}, z_k)$. Poter campionare questa funzione densità di probabilità permetterebbe di includere le osservazioni nella fase di predizione e, nel caso il peso dovuto all'aumento dell'utilizzo delle risorse di calcolo non diventi eccessivo, la precisione e l'affidabilità della stima dello stato potrebbero aumentare.

Bibliografia

[1] Gregory Dudek and Michael Jenkin. *Computationale principles of mobile Robotics*. Cambridge University Press, 2000.

[2] David Fillat and Jean-Arcady Meyer. Map-based navigation in mobile robots. I.a review of localization strategies. *http://animatlab.lip6.fr/pages/FilliatPublicationsEn*, February 2003.

[3] David Fillat and Jean-Arcady Meyer. Map-based navigation in mobile robots. II.a review of localization strategies. *http://animatlab.lip6.fr/pages/FilliatPublicationsEn*, February 2003.

[4] The application of probabilistic techniques for state/parameter estimation of (dynamical) systems and pattern recognition problems. *people.mech.kuleuven.ac.be/~kgadeyne/papers/*, June 2004.

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

[6] Matthias Mühlic. Particle filters, an overview. In *Filter-Workshop Bucuresti,2003*.

[7] John J. Leonard. *Directed Sonar Sensing for Mobile Robot Navigation*. Hugh F. Durrant-Whyte, 1992.

[8] Francesco Monica. Progettazione di un'architettura modulare, aperta e in tempo reale per un robot mobile. Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Parma, 2002.

[9] Open RObot COntrol Software, OROCOS. `http://www.orocos.org`.

[10] Hughes Network System. *The Adapive Comunication Environment: "Ace", A Tutorial*, 2000.

[11] C. Schlegel. Communications Patterns for OROCOS. Hints, Remarks, Specifications. Technical report, Research Institute for Applied Knowledge Processing (FAW), February 2002.

[12] Alessio Ruffini. Realizzazione di un'architettura software riconfigurabile per un robot mobile di servizio. Tesi di Laurea in Ingegneria Informatica, Università degli Studi di Parma, 2004.

[13] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons, 2000.

[14] Klaas Gadeyne, Tine Lefebvre, and Herman Bruyninckx. An open software framework for bayesian state and parameter estimation. In *2001 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING. Valencia, Spain. September 3-7 (Monday-Friday), 2001*, 2001.

[15] Bayesian filtering library. `http://people.mech.kuleuven.ac.be/~kgadeyne/software/actsens/software/%filter/doc/html/`.

[16] Autonomous manipulation. `http://www.mech.kuleuven.ac.be/pma/research/manip/default_en.phtml`.

[17] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. Player, version 1.42c2 user manual. *`http://playerstage.sourceforge.net/`*, December 2003.

[18] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. Player c++ client library version 1.4 reference manual. *`http://playerstage.sourceforge.net/`*, December 2003.

[19] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. Stage version 1.3.3 user manual. *`http://playerstage.sourceforge.net/`*, December 2003.

[20] Qt reference documentation. *`http://doc.trolltech.com/3.0/index.html`*.

[21] Nicolai M. Josuttis. *The C++ Standard Library, A Tutorial and Reference*, volume 2. Addison-Wesley, August 2002.

[22] Nomadic Technologies Inc. *The Nomad 200 User Guide*, December 1993.

[23] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. *`http://www.cm.mcgill/yiannis/Pubblications/index.html`*, 2003.

[24] OROCOS at FAW. *`http://www1.faw.uni-ulm.de/orocos`*.

[25] Sebastian Thrun. Robotic mapping: A survey. *`http://robots.stanford.edu/papers/`*, February 2002.