UNIVERSITÀ DEGLI STUDI DI ROMA
"TOR VERGATA"

FACOLTÀ DI INGEGNERIA

Tesi di Laurea Specialistica in Ingegneria Informatica

Filtri particellari per la localizzazione e la navigazione di robot mobili in ambienti non noti privi di landmark

Relatore
Ing. F. Martinelli

Laureando
Fabrizio Romanelli

Anno Accademico 2004/2005

Indice

Ai miei genitori e
a coloro che mi hanno sostenuto in questi anni.

Capitolo 1

Introduzione

Nel contesto della robotica mobile e dell'automazione il problema dello *SLAM*[1] riveste un interesse sempre crescente: infatti la capacità di localizzarsi e di creare contemporaneamente la mappa di un ambiente sconosciuto, unita a quella di evitare gli ostacoli e di pianificare l'esplorazione, rende un robot mobile completamente autonomo. Il problema dello *SLAM* risulta tutt'altro che semplice, in quanto il robot ha il compito di stimare correttamente la propria posizione e al tempo stesso deve essere in grado di costruire una mappa dell'ambiente che lo circonda il meno possibile affetta da errori; in associazione allo *SLAM* , il robot per essere autonomo necessita di un algoritmo di navigazione che, oltre a pianificare l'esplorazione in modo tale da visitare luoghi sconosciuti all'interno di un ambiente, permetta di evitare ostacoli presenti sul suo percorso (algoritmo di *obstacle avoidance*) mentre si muove nell'ambiente.

Negli ultimi anni il filtro di Kalman esteso (*EKF*) ha rappresentato lo strumento più valido per l'approccio al problema dello *SLAM* . Come è noto, gli algoritmi *EKF-based SLAM* hanno due difetti, la complessità di tipo quadratico e la possibilità di effettuare associazioni di dati errate, che li rendono spesso difficilmente applicabili in ambienti reali. L'approccio allo *SLAM problem* descritto in questa tesi è chiamato *FastSLAM* e permette di fattorizzare lo *SLAM pos-*

[1] *Simultaneous Localization and Mapping.*

terior[2] in $N + 1$ stimatori ricorsivi, uno stimatore relativo al percorso del robot ed N stimatori indipendenti relativi agli N landmark condizionati alla stima del percorso del robot; il *FastSLAM* è una particolarizzazione del filtro particellare *Rao-Blackwellized* [5]. Oltre al vantaggio connesso alla dimensione fissa del filtro particellare, e alla snellezza computazionale rispetto ad un filtro di Kalman esteso classico, il *FastSLAM* ha l'ulteriore vantaggio che le decisioni per l'associazione di dati, data la natura particellare dell'algoritmo, possono essere prese diversamente per ogni particella[3] (*per-particle basis*).

Al *FastSLAM* è stato inoltre affiancato un algoritmo di *ICP*[4] *scan-matching*, tecnica basata sul confronto di due scansioni laser successive per stimare lo spostamento del robot; l'introduzione di questo algoritmo è stata necessaria per ridurre gli errori di tipo sistematico presenti sull'odometria.

Infine per rendere autonomo il robot, accanto ad un algoritmo di esplorazione, è stato implementato un algoritmo di *obstacle avoidance* necessario ad evitare impatti con gli ostacoli presenti sul percorso; l'algoritmo, analizzando il vettore rappresentativo dell'ostacolo, il cui modulo rappresenta quanto è vicino al robot, e la cui fase rappresenta l'angolazione tra il robot e l'ostacolo, fornisce dei valori necessari a correggere la rotta all'algoritmo di controllo, il quale sfrutta una legge proporzionale.

Prima di implementare il *FastSLAM* e l'algoritmo di *obstacle avoidance* sul robot mobile *XR4000* della famiglia *Nomad*, su questo è stato effettuato un lavoro di ripristino delle funzionalità dei motori, delle batterie[5] e dei sensori presenti (sensori di prossimità sonar e infrarosso, bussola digitale, sensore laser *long range*, telecamera digitale e scheda di comunicazione radio *ethernet*). Inoltre sono stati apportati dei miglioramenti: il vecchio processore *Intel Pentium II* 450 *MHz* è

[2] Il *posterior*, o probabilità a posteriori, è la probabilità condizionata di un evento, in cui si tenga conto di dati empirici.

[3] Nel *FastSLAM* ogni particella rappresenta una possibile posizione x, y, θ del robot, alla quale è associata inoltre la posizione dei landmark visibili dalla particella.

[4] Iterative Closest Point.

[5] Le batterie sono state aggiornate con batterie nuove e queste sono state adattate per essere rese compatibili con l'*XR4000*.

stato aggiornato con un processore *Intel Pentium III* 500 *MHz*, la RAM è stata portata da 128 *MB* a 384 *MB*. Infine il vecchio sistema operativo *Linux Red Hat 5.3* con Kernel *2.0.35* è stato aggiornato al sistema operativo *Linux Red Hat 7.3* con Kernel *2.4.18*, con un conseguente aumento delle prestazioni e della stabilità del sistema.

Gli algoritmi di navigazione, *obstacle avoidance* e *FastSLAM* sono stati implementati in linguaggio *C* su piattaforma Linux.

Capitolo 2

Il problema dello *SLAM*

Il problema dello *SLAM* consiste nella localizzazione e nella costruzione simultanea della mappa di un ambiente non noto a *priori* in cui si muove un robot mobile. Il robot effettua delle osservazioni degli oggetti nell'ambiente e della sua posizione, entrambe affette da rumore. Lo scopo dello *SLAM* è quello di costruire la mappa dell'ambiente esplorato ed il percorso del robot.

Se la mappa dell'ambiente fosse nota *a priori*, il problema della stima della posizione del robot si ridurrebbe ad un problema di localizzazione. Analogamente la costruzione della mappa di un ambiente, se fosse nota con esattezza la posizione del robot ad ogni istante, sarebbe di facile soluzione. Nello *SLAM* invece la creazione della mappa e la localizzazione devono essere effettuati simultaneamente dal robot, in modo tale che nessuna conoscenza *a priori* dell'ambiente e della posizione del robot sia necessaria: è facile intuire che lo *SLAM* quindi rappresenta la base per rendere autonomo un robot mobile.

Lo *SLAM* ha trovato largo impiego in applicazioni in cui non sono disponibili misure accurate della posizione del robot o in cui è preferibile l'utilizzo di robot mobili per l'esplorazione di luoghi in generale troppo distanti o troppo pericolosi per l'uomo: in queste condizioni è evidente che è necessario rendere completamente autonomo il robot, integrando oltre agli algoritmi di pianificazione del

moto e di esplorazione anche un valido algoritmo di localizzazione e costruzione di mappe simultanea.

La relazione che intercorre tra la localizzazione e la costruzione della mappa è una diretta conseguenza di come gli errori dovuti al movimento del robot si ripercuotono sugli errori propri dei sensori. Durante il movimento, la stima della posizione del robot è affetta da errori odometrici: la posizione degli oggetti presenti in prossimità al robot è quindi affetta oltre che da errori di misura anche dagli errori causati dal movimento del robot. La differenza sostanziale tra le due tipologie di errori è che la prima è di natura probabilistica e la seconda sistematica oltre che probabilistica. È evidente che la stima della posizione del robot è necessaria per poter ottenere una stima della mappa dell'ambiente.

In letteratura sono presenti algoritmi di tipo genetico per risolvere il problema dello *SLAM*, come in [11] e [16], oltre che tecniche basate sul filtro di Kalman esteso classico combinato ad altri metodi, come riportato in [20] ed infine altre tipologie di *SLAM* come quello che utilizza un filtro locale a sottomappe, riportato in [25].

Lo *SLAM posterior* Nello *SLAM* un robot esegue il controllo e le osservazioni dell'ambiente, entrambe affette da rumore; ogni controllo e ogni osservazione, se ad entrambi vengono associati i relativi modelli di rumore, possono essere considerati come vincoli probabilistici nel senso che ogni controllo vincola in senso probabilistico due posizioni successive del robot, mentre ogni osservazione vincola le posizioni relative del robot e degli oggetti nell'ambiente. Con il crescere dei vincoli probabilistici le nuove osservazioni possono essere utilizzate non solo per aggiornare la posizione e la mappa costruita fino all'istante corrente, ma anche per correggere la mappa costruita negli istanti precedenti.

Inizialmente i vincoli probabilistici possono essere molto incerti, ma osservando ripetutamente gli stessi oggetti nella mappa, questi diventano sempre più rigidi; se le osservazioni e i controlli fossero infiniti, le posizioni degli oggetti della mappa sarebbero completamente correlati. In questo contesto lo scopo dello

SLAM è quello di fare una stima della mappa e della posizione del robot, date tutte le osservazioni ed i controlli.

Le soluzioni *online* più adottate per risolvere il problema dello *SLAM* tendono a stimare una distribuzione probabilistica di *posterior* su tutte le possibili mappe e su tutte le possibili posizioni del robot, noti i dati dei sensori accumulati dal robot. Questa distribuzione, chiamata *SLAM posterior* è riportata nella (2.1)

$$p(s_t, \Theta \mid z^t, u^t, n^t) \tag{2.1}$$

dove con s_t si è indicata la posizione corrente del robot e con Θ la mappa. Il *posterior* è condizionato all'insieme di tutti i dati sensoriali z^t, i controlli u^t e l'associazione di dati n^t cioè la mappatura delle osservazioni z^t nelle *feature* contenute in Θ.

Per rappresentare la mappa Θ, può essere utilizzato un qualsiasi modello parametrizzato, ma in generale si assume che sia un *set* di *feature*: scegliendo la mappa in questo modo l'ambiente visto dal robot può essere rappresentato come un insieme di punti detti *landmark*. In una implementazione reale i *landmark* possono corrispondere alla posizione delle *feature* estratte dai sensori. Per rappresentare la mappa Θ nello *SLAM* qui trattato, le *feature* sono di tipo geometrico e sono rappresentate da rette estratte direttamente dai dati sensoriali oppure da angoli.

La stima del *posterior* è fondamentale in quanto, oltre a fornire la mappa e la posizione del robot più probabili, stima anche l'incertezza associata a queste quantità. I vantaggi legati alla stima del *posterior* sono essenzialmente due: considerando una distribuzione di possibili soluzioni, piuttosto che un'unica soluzione probabile, riduce la possibilità di errore anche in ambienti molto rumorosi. Il secondo vantaggio è rappresentato dal fatto che l'incertezza può essere utilizzata per valutare l'informazione relativa trasmessa da differenti parti della soluzione.

Utilizzando il filtro di Bayes è possibile calcolare[1] lo *SLAM posterior* al tempo

[1]Per una deduzione analitica si rimanda alla sezione 3.1.1.

t (riportato in (2.2)), dato il *posterior* al tempo $t-1$.

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta\, p(z_t | s_t, \Theta, n_t) \int p(s_t | s_{t-1}, u_t)\, p(s_{t-1}, \Theta | z^{t-1}, u^{t-1}, n^{t-1}) ds_{t-1} \quad (2.2)$$

In generale non è possibile risolvere in forma chiusa la (2.2), è comunque possibile risolverla assumendo una forma particolare per il *posterior*. Molte tecniche statistiche di stima, come il filtro di Kalman ed il filtro particellare, sono approssimazioni del filtro di *Bayes* nella sua forma generale.

2.1 Il filtro di Kalman esteso (*EKF*)

Per il problema generale della stima dello stato $x \in \mathbb{R}^n$ di un processo controllato tempo discreto governato da un'equazione stocastica lineare alle differenze il filtro di Kalman discreto[2] è lo stimatore ottimo nel caso lineare e gaussiano. Se il processo da stimare e la relazione che intercorre tra il processo e le misure sono non lineari, il filtro di Kalman discreto non è la soluzione ottima al problema della stima dello stato x: un filtro di Kalman che linearizza intorno alla media e alla covarianza è chiamato filtro di Kalman esteso (*EKF*[3]).

Se il processo è governato da una equazione stocastica alle differenze non lineare

$$x_t = f(x_{t-1}, u_{t-1}, w_{t-1}) \quad (2.3)$$

con misura $z \in \mathbb{R}^m$ cioè

$$z_t = h(x_t, v_t) \quad (2.4)$$

dove le variabili aleatorie w_t e v_t rappresentano rispettivamente il rumore del processo e della misura, la funzione non lineare f nella (2.3) mette in relazione lo stato al tempo precedente $t-1$ allo stato al tempo corrente t. Nella (2.3) u_{t-1} è il controllo applicato al tempo $t-1$ e w_{t-1} è il rumore di processo a

[2] Per una trattazione più esaustiva del filtro di Kalman esteso si rimanda a [24].

[3] Acronimo per *Extended Kalman Filter*.

media nulla (rumore bianco). La funzione non lineare h nella (2.4) rappresenta la relazione che intercorre tra lo stato x_t e la misura z_t, dove v_t è il rumore di misura.

Le variabili aleatorie v_t e w_t non sono note ad ogni istante, è possibile comunque approssimare sia il vettore di stato che quello di misura senza considerare le due grandezze

$$\tilde{x}_t = f(\hat{x}_{t-1}, u_{t-1}, 0) \tag{2.5}$$

e

$$\tilde{z}_t = h(\tilde{x}_t, 0) \tag{2.6}$$

dove $\hat{x}_t$ è una stima *a posteriori* dello stato. È da notare comunque un difetto del filtro di Kalman, cioè che le distribuzioni delle varie variabili aleatorie presenti nelle equazioni (2.3) e (2.4) non sono più normali, per via delle rispettive trasformazioni non lineari: l'*EKF* è quindi uno stimatore dello stato *ad hoc* che approssima l'ottimo dato dalla regola di *Bayes* con la linearizzazione.

Linearizzando le (2.5) e (2.6) si ottiene

$$x_t \approx \tilde{x}_t + A(x_{t-1} - \hat{x}_{t-1}) + W w_{t-1} \tag{2.7}$$

$$z_t \approx \tilde{z}_t + H(x_t - \tilde{x}_t) + V v_t \tag{2.8}$$

dove[4]

- x_t e z_t sono i vettori di stato e di misura attuali,
- $\tilde{x}_t$ e $\tilde{z}_t$ sono i vettori di stato e di misura approssimati dalle (2.5) e (2.6),
- $\hat{x}_t$ è una stima dello stato *a posteriori* all'istante t, dato il vettore delle misure z_t,
- le variabili aleatorie w_t e v_t rappresentano rispettivamente i rumori del processo e di misura,

[4]Nelle matrici Jacobiane è stato soppresso il pedice t, per snellezza di notazione, ma è evidente che le matrici in questione sono differenti per ogni istante t e devono essere quindi ricalcolate ad ogni passo.

- A è la matrice Jacobiana delle derivate parziali di f rispetto a x, cioè:

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_{t-1}, u_{t-1}, 0),$$

- W è la matrice Jacobiana delle derivate parziali di f rispetto a w, cioè:

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}(\hat{x}_{t-1}, u_{t-1}, 0),$$

- H è la matrice Jacobiana delle derivate parziali di h rispetto a x, cioè:

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\tilde{x}_t, 0),$$

- V è la matrice Jacobiana delle derivate parziali di h rispetto a v, cioè:

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}(\tilde{x}_t, 0).$$

È possibile definire l'errore di predizione ed il residuo della misura come

$$\tilde{e}_{x_t} \equiv x_t - \tilde{x}_t \tag{2.9}$$

e

$$\tilde{e}_{z_t} \equiv z_t - \tilde{z}_t \tag{2.10}$$

Le (2.7) e (2.8), utilizzando la (2.9) e la (2.10) possono essere riscritte come

$$\tilde{e}_{x_t} \approx A(x_{t-1} - \hat{x}_{t-1}) + \epsilon_t \tag{2.11}$$

e

$$\tilde{e}_{z_t} \approx H\tilde{e}_{x_t} + \eta_t \tag{2.12}$$

dove ϵ_t e η_t rappresentano nuove variabili aleatorie indipendenti a media nulla e con matrici di covarianza rispettivamente $W\,Q\,W^T$ e $V\,R\,V^T$, con Q matrice di covarianza del rumore di processo ed R matrice di covarianza del rumore di misura.

Chiamando con $\hat{e}_t$ la stima dell'errore di predizione $\tilde{e}_{x_t}$ dato dalla (2.11), questa quantità può essere utilizzata per ottenere una stima dello stato *a posteriori* per il processo non lineare originario utilizzando la (2.13)

$$\hat{x}_t = \tilde{x}_t + \hat{e}_t \tag{2.13}$$

Le variabili aleatorie nella (2.11) e nella (2.12) hanno le seguenti distribuzioni di probabilità:

$$p(\tilde{e}_{x_t}) \sim N(0, E([\tilde{e}_{x_t}\tilde{e}_{x_t}^T]))$$

$$p(\epsilon_t) \sim N(0, W\,Q_t\,W^T)$$

$$p(\eta_t) \sim N(0, V\,R_t\,V^T)$$

L'equazione del filtro di Kalman utilizzata per ottenere una stima di $\hat{e}_t$, considerando le precedenti approssimazioni ed essendo nullo il valore predetto di $\hat{e}_t$, è quella riportata in (2.14), dove K_t rappresenta il guadagno del filtro di Kalman[5]

$$\hat{e}_t = K_t\tilde{e}_{z_t} \tag{2.14}$$

Sostituendo la (2.14) nella (2.13) ed utilizzando la (2.12) si ottiene

$$\hat{x}_t = \tilde{x}_t + K_t\tilde{e}_{z_t} = \tilde{x}_t + K_t(z_t - \tilde{z}_t) \tag{2.15}$$

che può essere utilizzata per l'aggiornamento della misura nel filtro di Kalman esteso.

Utilizzando come apice il segno meno per indicare che la quantità è la stima *a priori* all'istante t data la conoscenza del processo prima dell'istante t, le equazioni del filtro di Kalman esteso sono riportate nella tabella 2.1 e nella 2.2. Le equazioni in tabella 2.1 proiettano le stime dello stato e della covarianza dall'istante di tempo precedente $t-1$ all'istante corrente t. Come si può notare dalla tabella 2.2,

[5] Si rimanda alla tabella 2.2.

$\hat{x}_t^- = f(\hat{x}_{t-1}, u_{t-1}, 0)$
$P_t^- = A_t\, P_{t-1}\, A_t^T + W_t\, Q_{t-1}\, W_t^T$

Tabella 2.1: Equazioni di aggiornamento del filtro di Kalman esteso.

$K_t = P_t^- H_t^T (H_t\, P_t^-\, H_t^T + V_t\, R_t\, V_t^T)^{-1}$
$\hat{x}_t = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-, 0))$
$P_t = (I - K_t\, H_t) P_t^-$

Tabella 2.2: Equazioni di aggiornamento delle misure del filtro di Kalman esteso.

le equazioni di aggiornamento delle misure[6] sono utilizzate per correggere le stime dello stato e della covarianza con il vettore delle misure z_t.

Nonostante la rilevante influenza del filtro di Kalman esteso, tale da renderlo l'approccio predominante nella risoluzione dei problemi di *SLAM*, esso soffre di due problemi che lo rendono difficilmente applicabile in ambienti di grandi dimensioni oppure in ambienti reali: la complessità quadratica e la sensibilità al fallimento nell'associazione dei dati.

Complessità quadratica dell'*EKF* Nel filtro di Kalman esteso sia il tempo di calcolo che la memoria richiesta aumentano quadraticamente con il numero di landmark presenti nella mappa[7]. L'incertezza dello *SLAM posterior* è rappresentata come una matrice di covarianza contenente le correlazioni tra tutte le possibili coppie delle variabili di stato. È provato che la memoria richiesta per immagazzinare questa matrice di covarianza cresce come N^2 (con N numero totale di landmark presenti nella mappa). Inoltre ogni osservazione incorporata nell'*EKF* deve necessariamente influenzare tutte le altre variabili di stato: per

[6] Nel filtro di Kalman esteso un ruolo importante è giocato dalla matrice Jacobiana H_t nell'equazione per il calcolo del guadagno dell'*EKF* K_t, in quanto questa serve per amplificare o propagare solo la componente rilevante dell'informazione data dalle misure.

[7] Questo rappresenta il motivo per il quale l'applicazione dell'*EKF* per la risoluzione dello *SLAM* è limitato all'impiego in ambienti in cui siano presenti solo poche centinaia di landmark. In ambienti reali in cui sono presenti milioni di *feature* è evidente che il problema dello *SLAM* non può essere risolto con il filtro di Kalman esteso.

includere una osservazione sensoriale il filtro di Kalman esteso deve effettuare una operazione su ogni elemento della matrice di covarianza che richiede un tempo quadratico[8].

Associazione di dati a ipotesi singola Il problema dello *SLAM* è in generale formulato considerando l'associazione di dati conosciuta[9]: questo significa assumere che ad ogni osservazione effettuata dal robot è legata una variabile n_t che specifica quale landmark ha generato l'osservazione sensoriale[10]. Nel filtro di Kalman esteso l'approccio standard all'associazione di dati è quello di assegnare ogni osservazione ad un landmark utilizzando un criterio di massima verosimiglianza[11].

Dato che l'*EKF* non ha la capacità di rappresentare l'incertezza relativa all'associazione dei dati, incorporare nel filtro una associazione di dati sbagliata porterebbe ad effetti indesiderati e se vi fossero parecchie associazioni di dati errate, il filtro divergerebbe. La corretta associazione dei dati non avviene immediatamente, al primo passo, anzi in generale è necessario effettuare molte osservazioni per identificare il corretto *mapping* tra letture sensoriali e landmark.

[8] In pratica è evidente che non è conveniente implementare l'*EKF* in forma generale, risulta piuttosto vantaggioso implementare approssimazioni del filtro di Kalman che permettano di ottenere costi computazionali molto ridotti.

[9] Per associazione di dati si intende il *mapping* tra le osservazioni ed i landmark; l'associazione di dati conosciuta (*known data association*) è quella per cui è noto il legame tra le osservazioni ed i landmark, mentre l'associazione di dati non conosciuta (*unknown data association*) è quella per cui non è noto a priori il *mapping* tra osservazioni e landmark.

[10] Nel mondo reale le associazioni tra le osservazioni ed i landmark sono delle variabili non ricavabili direttamente, ma devono essere definite in modo da determinare la posizione del robot e quella dei landmark.

[11] Ogni osservazione è assegnata al landmark che con maggiore probabilità l'ha generata; se la probabilità dell'osservazione è troppo bassa, allora verrà aggiunto un nuovo landmark al filtro.

2.2 Filtri particellari per lo *SLAM*

In letteratura sono presenti vari algoritmi approssimati di *SLAM EKF* che permettono di ridurre la mappa completa in un insieme più ridotto di sotto-mappe, in modo tale da ridurre di molto la complessità computazionale, rispetto all'*EKF* classico; questa classe di algoritmi soffre comunque del problema dell'associazione dei dati.

Nel problema dello *SLAM* la correlazione tra gli elementi della mappa è dovuta solo all'incertezza sulla posizione del robot: se quindi fosse noto con esattezza il percorso del robot, la posizione dei landmark potrebbe essere stimata indipendentemente. Se si considera il problema dello *SLAM* come una rete dinamica di *Bayes*, è possibile verificare l'esattezza della precedente affermazione. Considerando la posizione del robot s_t al tempo t come una funzione probabilistica dipendente dalla posizione all'istante precedente s_{t-1} e dal controllo u_t e considerando l'osservazione z_t al tempo t come dipendente dalla posizione s_t e dal landmark osservato θ_{n_t} lo *SLAM posterior* può essere riscritto come in (2.16)

$$p(s^t, \Theta | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_{n=1}^{N} p(\theta_n | s^t, z^t, u^t, n^t) \tag{2.16}$$

dove il primo termine a secondo membro della (2.16) rappresenta il *path posterior* ed il secondo termine gli stimatori della posizione dei landmark[12]. Il *posterior* così fattorizzato può essere efficientemente approssimato utilizzando un filtro particellare, dove ogni singola particella rappresenta un percorso campione del robot; ad ogni particella sono integrati N stimatori dei landmark indipendenti (implementati come *EKF*), uno per ogni landmark presente nella mappa. In totale ci saranno quindi $N \cdot M$ filtri di Kalman, uno per ogni *feature* della mappa, per tutte le M particelle presenti nel filtro particellare. Questa particolare

[12] Questo significa che lo *SLAM posterior* può essere decomposto come un prodotto di $N+1$ stimatori ricorsivi: uno per la stima del percorso del robot ed N stimatori della posizione dei landmark, tutti condizionati alla stima del percorso. Questa fattorizzazione, che è esatta e non approssimata, è stata presentata da Murphy nel 1999 [14].

soluzione al problema dello *SLAM* prende il nome di *FastSLAM* [13] ed è una particolarizzazione del filtro particellare *Rao-Blackwellized* [5].

Dato che il percorso del robot cresce con l'aumentare del tempo, utilizzare uno *SLAM posterior* come quello descritto in (2.16) può sembrare una scelta non buona, in quanto ci si può aspettare che la dimensione del filtro addetto alla stima del *posterior* cresca anch'esso col tempo. In realtà questo non accade, poiché le equazioni di aggiornamento dei landmark e i pesi associati alle particelle dipendono soltanto dall'ultima posizione del robot s_t, e non da tutte le precedenti posizioni[13].

Oltre al vantaggio connesso alla dimensione fissa del filtro particellare, e alla snellezza computazionale rispetto ad un filtro di Kalman esteso classico, il *fastSLAM* ha l'ulteriore vantaggio che le decisioni per l'associazione di dati, data la natura particellare dell'algoritmo, possono essere prese diversamente per ogni particella (*per-particle basis*). Questo significa che alle particelle che predicono la corretta associazione di dati, sarà assegnato un peso maggiore rispetto a quelle che predicono un'associazione di dati sbagliata, che eventualmente verranno rimosse[14].

Anche per l'aggiunta o la rimozione dei landmark il filtro particellare si mostra molto flessibile, in quanto possono sorgere situazioni in cui differenti particelle hanno un diverso numero di *feature*, il che se da una parte rende più complicata la costruzione della mappa, dall'altra permette al *fastSLAM* di rimuovere landmark spuri, se osservazioni successive suggeriscono che a quei landmark sono associate osservazioni in maniera errata.

[13] Questo implica che la dimensione del filtro particellare rimane invariata nel tempo.

[14] Con il crescere delle osservazioni le associazioni dei dati passate possono essere aggiustate in modo tale da rendere più corretto il *mapping* tra le osservazioni e le *feature*.

Capitolo 3

FastSLAM

Il problema dello *SLAM* deriva dal riscontro che quando il percorso effettuato da un robot, che effettua controlli e raccoglie osservazioni di *feature* nel mondo reale entrambi affetti da rumore, è sconosciuto, perché ad esempio non si hanno a disposizione strumenti (come il GPS) che permettono di conoscere con esattezza la posizione del robot, all'errore nel percorso effettuato dal robot sono correlati gli errori nella mappa: conseguenza di questo problema è che sia lo stato del robot che la mappa devono essere stimati simultaneamente.

La correlazione tra il percorso del robot e la mappa mette in luce un aspetto cruciale dello *SLAM* ossia la chiusura dei cicli[1]. L'importanza della chiusura dei cicli risiede nel fatto che all'inizio della sua esplorazione il robot non ha alcuna incertezza relativa alla sua posizione e quindi anche l'incertezza relativa ai landmark vicini alla posizione iniziale è nulla; infatti la posizione iniziale del robot è presa come origine del sistema di riferimento globale. Questo implica che appena il robot osserva nuovamente un landmark vicino alla posizione iniziale, e di cui quindi è molto sicuro, l'incertezza relativa alla stima della sua posizione attuale decresce in maniera significativa ed allo stesso modo anche l'incertezza relativa alle posizioni passate del robot nonché quella relativa ai landmark precedentemente osservati (che sono appunto legati alle posizioni passate).

[1] Chiudere il ciclo significa che posizione iniziale e finale del robot coincidono.

Il filtro di Kalman e l'*EKF* rappresentano distribuzioni di probabilità utilizzando un modello parametrizzato (una distribuzione multivariata di Gauss); i filtri particellari diversamente rappresentano distribuzioni che utilizzano un insieme finito di stati campione, detti *particelle*. Le regioni che hanno un'alta probabilità di contenere il robot contengono un numero molto elevato di particelle, mentre quelle a bassa probabilità ne contengono poche o nessuna. Dato un numero sufficiente di campioni i filtri particellari possono approssimare distribuzioni multi-modali anche molto complesse. L'applicazione dei filtri particellari nella robotica mobile ha trovato largo uso nella localizzazione, in cui è nota a priori la mappa ed il robot deve essere in grado di stimare la propria posizione[2].

La capacità di approssimare distribuzioni multi-modali e di includere modelli non lineari per le misure e per il movimento, rende i filtri particellari particolarmente robusti. In generale però questi particolari filtri sono utilizzati per risolvere problemi di dimensione ridotta, e quindi possono in prima battuta sembrare inutilizzabili nel contesto dello *SLAM*, che ha dimensioni molto grandi. Una particolare classe di filtri particellari, chiamati *Rao-Blackwellized* [5], permette di fattorizzare il problema dello *SLAM* in una sotto-classe di problemi di stima dei landmark, tutti indipendenti tra di loro e condizionati alla stima del percorso del robot.

3.1 Descrizione del problema

3.1.1 Lo *SLAM problem* come una catena di Markov

Assumendo che il robot si muove in un ambiente a due dimensioni, la posizione del robot è data dalle coordinate x ed y e dall'angolo di *heading*. Nel corso della trattazione la posizione del robot al tempo t sarà indicata con s_t, e con s^t sarà

[2] Il più comune esempio di localizzazione con l'utilizzo del filtro particellare è la *MCL* acronimo per *Monte Carlo Localization* [23], dove le particelle sono utilizzate per rappresentare la distribuzione delle possibili posizioni del robot relative alla mappa.

indicata la traiettoria completa fino al tempo t ad ogni passo (3.1).

$$s^t = \{s_1, s_2, \ldots, s_t\} \tag{3.1}$$

Le posizioni degli N landmark[3] nella mappa (che per semplicità verrà indicata con Θ) verranno indicate con $\{\theta_1, \theta_2, \ldots, \theta_N\}$. Durante la navigazione il robot accumula informazioni relative al movimento: queste informazioni possono essere generate osservando i sensori odometrici oppure i controlli effettuati sul robot. Per descrivere l'insieme dei controlli eseguiti dal robot si utilizza la notazione in (3.2)

$$u^t = \{u_1, u_2, \ldots, u_t\} \tag{3.2}$$

dove con u_i (con $i = 1, \ldots, t$) si è indicato il controllo al tempo i. Durante il proprio moto, il robot accumula anche osservazioni dell'ambiente, in particolare delle *feature* presenti, che sono descritte dal raggio e dall'angolo compreso tra il robot ed il landmark. L'insieme delle osservazioni è indicato con z^t (3.3).

$$z^t = \{z_1, z_2, \ldots, z_t\} \tag{3.3}$$

dove con z_i (con $i = 1, \ldots, t$) si è indicata l'osservazione al tempo i. Per quanto concerne l'associazione di dati, occorre definire una quantità[4] n_t che rappresenta l'identità del landmark corrispondente all'osservazione z_t. In accordo alla notazione precedentemente utilizzata, si può scrivere l'insieme delle associazioni di dati come in (3.4)

$$n^t = \{n_1, n_2, \ldots, n_t\} \tag{3.4}$$

Per semplicità si è assunto che ad ogni controllo u_t è associata una singola misura[5] z_t.

Come già detto precedentemente lo scopo dello *SLAM* è quello di stimare la posizione del robot s_t e la mappa Θ, date le osservazioni z^t ed i controlli u^t

[3]Per landmark si intende una caratteristica (*feature*) estratta direttamente dai sensori, come ad esempio una caratteristica geometrica (retta o punto).

[4]La notazione $n_2 = 5$ indica che all'istante 2 il robot osserva il landmark numero 5.

[5]Osservazioni multiple ad ogni passo possono essere processate in maniera sequenziale, ma per semplicità di notazione questo caso non verrà considerato.

affetti da rumore. In termini probabilistici questo si esprime con il cosiddetto *SLAM posterior*, la cui forma generale è riportata in (3.5).

$$p(s_t, \Theta | z^t, u^t) \tag{3.5}$$

Più semplicemente si può scrivere, se è noto l'insieme dell'associazione di dati n^t:

$$p(s_t, \Theta | z^t, u^t, n^t) \tag{3.6}$$

Dopo aver definito lo *SLAM posterior*, si può ricondurre il problema dello *SLAM* ad una catena di Markov. Infatti la posizione corrente del robot s_t può essere scritta come una funzione probabilistica della posizione del robot al tempo $t-1$ e del controllo applicato u_t; questa funzione prende il nome di *modello di movimento* perché descrive come il controllo guida il movimento del robot, ed inoltre questa funzione (riportata in (3.7)) descrive il modo in cui il rumore sul controllo introduce incertezza nella stima della posizione attuale del robot.

$$p(s_t | s_{t-1}, u_t) \tag{3.7}$$

Oltre al *modello di movimento* occorre considerare il *modello di misura*[6]. In particolare l'osservazione z_t è una funzione del landmark osservato (o analogamente dei landmark osservati) θ_{n_t} e della posizione attuale del robot s_t come è riportato in (3.8).

$$p(z_t | s_t, \Theta, n_t) \tag{3.8}$$

Utilizzando il *modello di movimento* ed il *modello di misura* è possibile calcolare ricorsivamente lo *SLAM posterior* al tempo t conoscendo il *posterior* al tempo $t-1$: questa implementazione ricorsiva prende il nome di filtro di Bayes.

Deduzione del filtro di Bayes Il *posterior* definito in (3.6) può essere riscritto utilizzando la regola di Bayes, come segue:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta \, p(z_t | s_t, \Theta, z^{t-1}, u^t, n^t) \, p(s_t, \Theta | z^{t-1}, u^t, n^t) \tag{3.9}$$

[6] Il *modello di misura* descrive la fisica ed il modello di errore del sensore del robot.

dove η è il denominatore della regola di Bayes. Considerando che z_t è funzione della posizione del robot s_t, della mappa Θ e dell'associazione di dati n_t, come descritto in (3.8), la (3.9) può essere riscritta come:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta\, p(z_t | s_t, \Theta, n_t)\, p(s_t, \Theta | z^{t-1}, u^t, n^t) \tag{3.10}$$

Applicando il teorema della probabilità totale al secondo termine a secondo membro della (3.10), considerando la posizione del robot al tempo $t-1$, si ha:

$$p(s_t, \Theta | z^t, u^t, n^t) = \\ \eta\, p(z_t | s_t, \Theta, n_t) \int p(s_t, \Theta | s_{t-1}, z^{t-1}, u^t, n^t)\, p(s_{t-1} | z^{t-1}, u^t, n^t) ds_{t-1} \tag{3.11}$$

Utilizzando la definizione di probabilità condizionata, il primo termine nell'integrale si può espandere e risulta:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta\, p(z_t | s_t, \Theta, n_t) \cdot \\ \cdot \int p(s_t | \Theta, s_{t-1}, z^{t-1}, u^t, n^t)\, p(\Theta | s_{t-1}, z^{t-1}, u^t, n^t)\, p(s_{t-1} | z^{t-1}, u^t, n^t)\, ds_{t-1} \tag{3.12}$$

Considerando che il primo termine nell'integrale può essere semplificato notando che s_t è funzione di s_{t-1} e di u_t e considerando inoltre che i termini nell'integrale possono essere combinati, si ottiene:

$$p(s_t, \Theta | z^t, u^t, n^t) = \\ \eta\, p(z_t | s_t, \Theta, n_t) \int p(s_t | s_{t-1}, u_t)\, p(s_{t-1}, \Theta | z^{t-1}, u^t, n^t) ds_{t-1} \tag{3.13}$$

Dato che la posizione corrente del robot s_t e l'associazione di dati n_t non forniscono informazioni aggiuntive relative a s_{t-1} e alla mappa Θ senza l'ultima osservazione z_t, la (3.13) può essere riscritta come segue:

$$p(s_t, \Theta | z^t, u^t, n^t) = \\ \eta\, p(z_t | s_t, \Theta, n_t) \int p(s_t | s_{t-1}, u_t)\, p(s_{t-1}, \Theta | z^{t-1}, u^{t-1}, n^{t-1}) ds_{t-1} \tag{3.14}$$

Il risultato di questa ulteriore semplificazione dà come risultato una formula ricorsiva per il calcolo dello *SLAM posterior* al tempo t dato il *posterior* al tempo $t-1$, il *modello di movimento* $p(s_t | s_{t-1}, u_t)$ ed il *modello di misura* $p(z_t | s_t, \Theta, n_t)$.

3.1.2 Il filtraggio con l'*EKF*

La risoluzione dell'integrale nell'equazione (3.14) non è possibile in forma chiusa. Tuttavia è possibile utilizzare il filtro di Kalman esteso per stimare lo *SLAM posterior* come proposto da Smith in [21]. L'*EKF* rappresenta lo *SLAM posterior* come una distribuzione multivariata di Gauss parametrizzata da una media μ_t e da una matrice di covarianza Σ_t: la media descrive lo stato più probabile del robot e dei landmark, mentre la matrice di covarianza esprime la correlazione esistente tra tutte le coppie delle variabili di stato. Il *posterior* può essere espresso in termini probabilistici come:

$$\begin{aligned} p(s_t, \Theta | z^t, u^t, n^t) &= N(x_t; \mu_t, \Sigma_t) \\ x_t &= \{s_t, \theta_{1t}, \ldots, \theta_{Nt}\} \\ \mu_t &= \{\mu_{s_t}, \mu_{\theta_{1t}}, \ldots, \mu_{\theta_{Nt}}\} \\ \Sigma_t &= \begin{bmatrix} \Sigma_{s_t} & \Sigma_{s_t,\theta_1} & \cdots & \Sigma_{s_t,\theta_N} \\ \Sigma_{\theta_1,s_t} & \Sigma_{\theta_1} & \Sigma_{\theta_1,\theta_2} & \vdots \\ \vdots & \Sigma_{\theta_2,\theta_1} & \ddots & \vdots \\ \Sigma_{\theta_N,s_t} & \cdots & \cdots & \Sigma_{\theta_N} \end{bmatrix} \end{aligned}$$

Per robot che si muovono in un piano, il vettore μ_t ha dimensione $2\,N + 3$, dove N è il numero di landmark: tre dimensioni sono necessarie per rappresentare la posizione del robot e due per specificare la posizione di ogni landmark. È facile verificare che la matrice di covarianza ha dimensione $2\,N + 3$ per $2\,N + 3$, e che quindi il numero di parametri necessari a descrivere il *posterior* dell'*EKF* è quadratico rispetto al numero di landmark nella mappa.

Come è noto, il filtro di Kalman è lo stimatore ottimo per un sistema lineare con rumore di tipo Gaussiano; il filtro di Kalman esteso, come già illustrato nella sezione 2.1, è un'estensione del filtro di Kalman ai sistemi non lineari, e questo è reso possibile considerando i modelli di misura e di movimento non lineari, linearizzati[7] attorno allo stato più probabile del sistema.

[7] In generale questa approssimazione è buona se i modelli reali sono approssimativamente lineari e se il passo di campionamento è sufficientemente piccolo.

In generale i problemi reali di *SLAM* non sono lineari, ma il filtro di Kalman esteso porta comunque a risultati molto soddisfacenti, tanto da essere considerato lo standard per eccellenza nel confronto con gli algoritmi di *SLAM* in linea. Comunque, come già detto nella sezione 2.1, l'*EKF* ha come svantaggi la complessità quadratica e l'associazione di dati a ipotesi singola.

3.1.3 Associazione di dati robusta

Nelle applicazioni reali dello *SLAM*, le associazioni di dati n^t sono difficilmente osservabili: in ogni caso se l'incertezza relativa alla posizione dei landmark è bassa rispetto alla distanza media tra i landmark, si possono utilizzare semplici regole euristiche per determinare la corretta associazione di dati. In particolare l'approccio più comune per l'associazione dei dati nello *SLAM* è quello di determinare ogni osservazione utilizzando un criterio di massima verosimiglianza[8]. Se la probabilità massima è al di sotto di una soglia fissata, all'osservazione viene associato un nuovo landmark.

Considerando il caso del filtro di Kalman esteso, la probabilità dell'osservazione, chiamata *innovazione*, può essere scritta come funzione della differenza tra l'osservazione z_t e l'osservazione che ci si aspetta $\hat{z}_{n_t}$. In generale si ha:

$$\begin{aligned}\hat{n}_t &= \arg\max_{n_t} p(z_t|n_t, s^t, z^{t-1}, u^t, \hat{n}^{t-1}) \\ &= \arg\max_{n_t} \frac{1}{\sqrt{|2\,\pi\, Z_t|}}\, exp\left\{-\frac{1}{2}\,(z_t - \hat{z}_{n_t})^T\, Z_t^{-1}\,(z_t - \hat{z}_{n_t})\right\}\end{aligned}$$

dove Z_t è la matrice di covarianza dell'innovazione al tempo t. In termini di *negative log likelihood*, si ha:

$$\hat{n}_t = \arg\min_{n_t} \ln|Z| + (z - \hat{z})^T\, Z^{-1}(z - \hat{z})$$

dove il secondo termine a secondo membro dell'equazione è chiamato distanza di

[8] Questo significa che ad ogni osservazione è associato il landmark che con maggiore probabilità l'ha generata.

Mahalanobis[9] e dove Z è la matrice di covarianza dell'innovazione. Le tipologie di associazioni di dati che utilizzano questa metrica sono chiamate spesso *nearest neighbour*. L'associazione dei dati che utilizza criteri di massima verosimiglianza funziona bene quando è molto più probabile l'associazione di dati corretta piuttosto che le associazioni sbagliate; comunque se l'incertezza nella posizione del landmark è alta, più di una associazione di dati ha probabilità alta. Come è facile aspettarsi, prendere una associazione di dati sbagliata[10] comporta effetti indesiderati sull'accuratezza della mappa risultante. Per ovviare a questo problema è necessario incorporare osservazioni che conducono ad associazioni di dati non ambigue, con il pericolo però di non processare molte osservazioni, soprattutto se l'ambiente circostante risulta molto rumoroso.

3.2 L'algoritmo FastSLAM

3.2.1 Rappresentazione fattorizzata del *posterior*

Gli algoritmi di *SLAM* più utilizzati, sono basati sulla stima del *posterior* sulla mappa e sulla posizione del robot, già definito come $p(s_t, \Theta|z^t, u^t, n^t)$. Il FastSLAM, invece, è basato sulla stima del *posterior* sulla mappa e sul percorso del robot, cioè la quantità definita come $p(s^t, \Theta|z^t, u^t, n^t)$. Utilizzando il *posterior* così definito è possibile fattorizzare il problema nel prodotto di termini più semplici. Conoscendo il percorso del robot, l'osservazione di un landmark non fornisce alcuna informazione sulla posizione degli altri landmark: questo significa che se si conoscesse il percorso reale del robot, si potrebbe stimare la posizione di ogni landmark come una quantità indipendente.

$$p(s^t, \Theta|z^t, u^t, n^t) = p(s^t|z^t, u^t, n^t) \prod_{n=1}^{N} p(\theta_n|s^t, z^t, u^t, n^t) \tag{3.15}$$

[9] La distanza di Mahalanobis è una metrica normalizzata dalle covarianze dell'osservazione e dalla stima del landmark.

[10] Una associazione di dati sbagliata è spesso provocata se i sensori sono molto affetti da rumore.

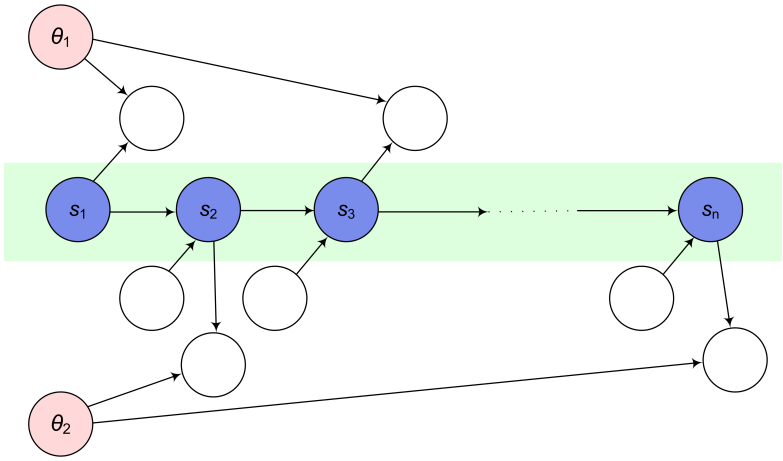

Figura 3.1: *Interpretazione dello* SLAM problem *come una rete dinamica di Bayes (DBN).*

Dalla (3.15) si può notare che lo *SLAM posterior* definito per il FastSLAM può essere fattorizzato, esattamente, nel prodotto del *posterior* del percorso del robot e da N *posterior* dei landmark condizionati al percorso del robot.

Dimostrazione della fattorizzazione del FastSLAM *posterior*

La fattorizzazione del FastSLAM si può ricavare dalla $p(s^t, \Theta | z^t, u^t, n^t)$; dalla definizione di probabilità condizionata, lo *SLAM posterior* può essere riscritto come:

$$p(s^t, \Theta | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t)\; p(\Theta | s^t, z^t, u^t, n^t) \tag{3.16}$$

Per dimostrare la (3.15) è sufficiente mostrare la seguente per tutti i valori non negativi di t:

$$p(\Theta | s^t, z^t, u^t, n^t) = \prod_{n=1}^{N} p(\theta_n | s^t, z^t, u^t, n^t) \tag{3.17}$$

La (3.17) può essere dimostrata per induzione. La prima quantità da ricavare è la probabilità condizionata del landmark θ_{n_t}. Questa quantità può essere riscritta utilizzando la regola di Bayes:

$$p(\theta_{n_t} | s^t, z^t, u^t, n^t) \overset{Bayes}{=} \frac{p(z_t | \theta_{n_t}, s^t, z^{t-1}, u^t, n^t)}{p(z_t | s^t, z^{t-1}, u^t, n^t)}\; p(\theta_{n_t} | s^t, z^{t-1}, u^t, n^t) \tag{3.18}$$

L'osservazione corrente z_t dipende solamente dallo stato del robot e dai landmark osservati, per cui nell'ultimo termine a secondo membro della (3.18) possiamo osservare che la posizione corrente s_t, il controllo corrente u_t e l'associazione di dati corrente n_t non hanno effetto su θ_{n_t} senza l'osservazione corrente z_t. Per cui si può riscrivere la (3.18) come:

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Markov}{=} \frac{p(z_t|\theta_{n_t}, s_t, n_t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} \, p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{3.19}$$

Risolvendo rispetto al secondo termine a secondo membro si ottiene:

$$p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) = \frac{p(z_t|s^t, z^{t-1}, u^t, n^t)}{p(z_t|\theta_{n_t}, s_t, n_t)} \, p(\theta_{n_t}|s^t, z^t, u^t, n^t) \tag{3.20}$$

Il secondo risultato intermedio necessario per poter dimostrazione la fattorizzazione è la probabilità condizionata $p(\theta_{n\neq n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1})$ relativa a quando nessun landmark è osservato. Per cui il *posterior* non cambierà se nessun landmark è osservato, cioè il landmark *posterior* al tempo t rimarrà uguale al tempo $t-1$.

$$p(\theta_{n\neq n_t}|s^t, z^t, u^t, n^t) \overset{Markov}{=} p(\theta_{n\neq n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{3.21}$$

Dati questi risultati intermedi si può dimostrare per induzione la (3.17). In primo luogo si assume per ipotesi per induzione che al tempo $t-1$ vale:

$$p(\Theta|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) = \prod_{n=1}^{N} p(\theta_n|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{3.22}$$

Per $t = 0$ nessuna osservazione è stata incorporata nello *SLAM posterior* e la fattorizzazione (3.17) risulta essere banalmente verificata. In generale quando $t > 0$ si deve espandere il primo membro della (3.17) con la regola di Bayes:

$$p(\Theta|s^t, z^t, u^t, n^t) \overset{Bayes}{=} \frac{p(z_t|\Theta, s^t, z^{t-1}, u^t, n^t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} \, p(\Theta|s^t, z^{t-1}, u^t, n^t) \tag{3.23}$$

Si può di nuovo semplificare come nella (3.18), poiché z_t dipende solo da Θ, s_t e n_t quindi il numeratore della (3.23) può essere semplificato, inoltre la posizione dei landmark presenti nella mappa Θ non dipende da s_t, u_t, n_t senza

l'osservazione corrente z_t, e quindi la (3.23) diviene:

$$p(\Theta|s^t, z^t, u^t, n^t) \overset{Markov}{=} \frac{p(z_t|\theta_{n_t}, s_t, n_t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} \, p(\Theta|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (3.24)$$

Per induzione è quindi possibile sostituire il secondo termine della (3.24),

$$p(\Theta|s^t, z^t, u^t, n^t) \overset{Induzione}{=} \frac{p(z_t|\theta_{n_t}, s_t, n_t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} \prod_{n=1}^{N} p(\theta_n|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (3.25)$$

utilizzando quindi i risultati intermedi ottenuti dalla (3.20) e dalla (3.21) si ottiene:

$$p(\Theta|s^t, z^t, u^t, n^t) = p(\theta_{n_t}|s^t, z^t, u^t, n^t) \prod_{n \neq n_t}^{N} p(\theta_n|s^t, z^t, u^t, n^t) \quad (3.26)$$

che è proprio il prodotto del landmark *posterior* della (3.17), che si voleva dimostrare.

3.2.2 Struttura del FastSLAM

La fattorizzazione dello *SLAM posterior* presentata in (3.15) mette in evidenza come sia ben strutturata la scelta del *posterior*: questa struttura infatti suggerisce che sotto appropriate ipotesi, non è necessario mantenere esplicitamente correlazioni incrociate tra i landmark. Infatti il FastSLAM sfrutta la rappresentazione fattorizzata del *posterior*, utilizzando $N + 1$ filtri indipendenti; facendo questo tutti gli $N+1$ filtri risultano di dimensione ridotta. Il ruolo del filtro particellare nel FastSLAM è quello di stimare il primo termine a secondo membro della (3.15); il secondo termine, cioè gli N *posterior* relativi ai landmark sono stimati invece utilizzando dei filtri di Kalman esteso[11]. Tutti gli *EKF* relativi ai landmark dipendono dal percorso del robot, ed ogni particella del filtro particellare ha il proprio set di *EKF*; in totale ci sono $N \cdot M$ *EKF*, dove M è il numero totale di

[11] La differenza sostanziale con l'implementazione classica dell'*EKF* è che nel FastSLAM ogni singolo *EKF* segue la posizione di un solo landmark, e quindi il filtro è di dimensione piccola e fissata.

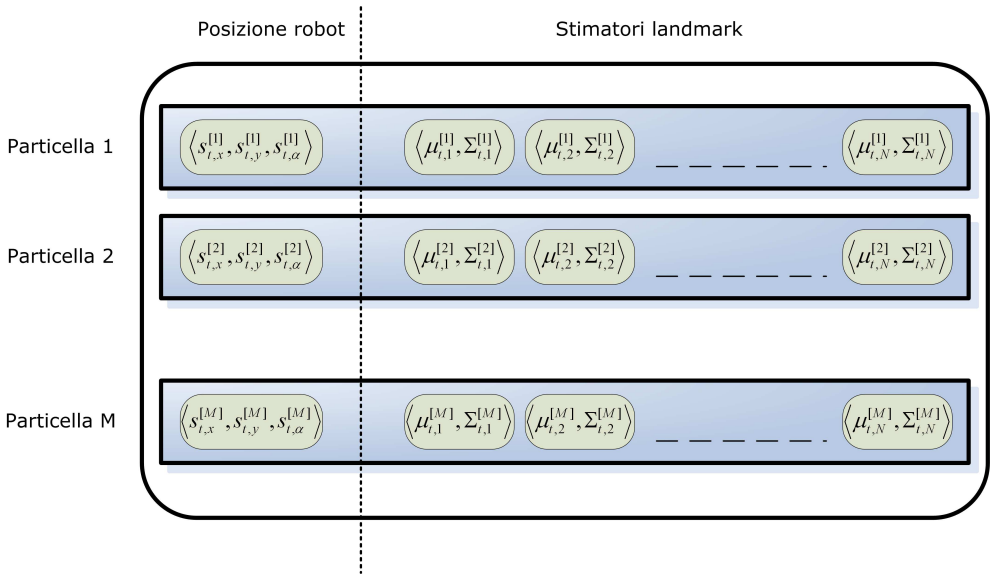

Figura 3.2: *Struttura del filtro particellare: sono presenti M particelle ed ogni particella contiene N* EKF.

particelle e dove N, come già detto, è il numero di landmark. La struttura del FastSLAM è un esempio del filtro particellare Rao-Blackwellized.

Ogni particella del FastSLAM assume la seguente forma:

$$S_t^{[m]} = \left\langle s^{t,[m]}, \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, \ldots, \mu_{N,t}^{[m]}, \Sigma_{N,t}^{[m]} \right\rangle \tag{3.27}$$

$[m] = 1, \ldots, M$ indica l'indice della particella; $s^{t,[m]}$ è la stima del percorso relativa all'$m-esima$ particella, e $\mu_{n,t}^{[m]}$ e $\Sigma_{n,t}^{[m]}$, con $n = 1, \ldots, N$, sono rispettivamente la media e la covarianza della Gaussiana che rappresenta la posizione della *feature* $n-esima$ condizionata al percorso $s^{t,[m]}$. Il filtraggio, cioè il calcolo del *posterior* al tempo t dato il *posterior* al tempo $t-1$ implica la generazione di un nuovo set[12] di particelle S_t dal set di particelle al tempo precedente S_{t-1}. Il procedimento di aggiornamento avviene in quattro passi:

1. Campionamento della nuova posizione del robot per ogni particella, dato il nuovo controllo;

[12]Il nuovo set di particelle contiene il controllo al tempo t, u_t e la misura z_t.

2. Aggiornamento degli *EKF* relativi ai landmark delle *feature* osservate in ogni particella;

3. Calcolo del peso di ogni particella;

4. Estrazione di un nuovo set di particelle utilizzando il *resampling*.

Il primo passo è quello di estrarre una nuova posizione del robot per ogni particella: ogni posizione è aggiunta alla stima del percorso del robot appropriata $s^{t-1,[m]}$. Successivamente i filtri di Kalman esteso relativi ai landmark osservati sono aggiornati con le nuove osservazioni. Il calcolo del peso di ogni particella è fondamentale in quanto i percorsi del robot, relativi ad ogni particella, non sono estratti dal *posterior* reale: per esprimere questa differenza ad ogni particella viene assegnato un peso. Infine viene estratto un nuovo set di particelle S_t dal set di particelle pesate, utilizzando il *resampling*: questo passo è necessario per assicurare che le particelle siano distribuite conformemente al *posterior* reale.

Campionamento di una nuova posizione del robot

Il set di particelle S_t viene calcolato dal set S_{t-1} al tempo $t-1$, dall'osservazione z_t e dal controllo u_t; il primo passo del FastSLAM è quello di generare probabilisticamente campioni di posizioni del robot al tempo t, data ogni particella $S_{t-1}^{[m]}$. Questi campioni si ottengono dal modello probabilistico di movimento, descritto in (3.28).

$$s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]}) \tag{3.28}$$

Questa stima è aggiunta ad un set temporaneo di particelle, insieme al percorso $s_{t-1}^{[m]}$; assumendo che il set di particelle S_{t-1} è distribuito secondo[13]:

$$p(s^{t-1}|z^{t-1}, u^{t-1}, n^{t-1}) \tag{3.29}$$

il nuovo set di particelle è distribuito come:

$$p(s^t|z^{t-1}, u^t, n^{t-1}) \tag{3.30}$$

[13] Questa assunzione è corretta asintoticamente.

La distribuzione di probabilità appena definita è chiamata *proposal distribution* del filtro particellare. Come già detto il modello di movimento può essere una qualsiasi funzione non lineare[14]; inoltre l'estrazione di una nuova posizione del robot è un'operazione che richiede un tempo di computazione costante per ogni particella.

Il modello di movimento del robot assume che la velocità del robot è costante nell'intervallo coperto da ogni controllo u_t, che è bidimensionale e che può essere descritto da una velocità traslazionale v_t ed una rotazionale ω_t. Inoltre si assume che il rumore sul controllo è distribuito secondo una Gaussiana; con la notazione $\mathcal{N}(x\,;\,\mu,\Sigma)$ si intende una distribuzione normale della variabile x con valor medio μ e covarianza Σ. Il modello di movimento è riportato nella (3.31) e nella (3.32).

$$v_t \sim \mathcal{N}(v\,;\,v_t, \alpha_1 v_t + \alpha_2) \tag{3.31}$$

$$\omega_t \sim \mathcal{N}(\omega\,;\,\omega_t, \alpha_3 \omega_t + \alpha_4) \tag{3.32}$$

Con questo modello di movimento è possibile considerare gli errori dovuti allo slittamento e al pattinamento, che sono fenomeni frequenti nei veicoli terrestri[15]. Il primo passo per estrarre una nuova posizione del robot da questo modello è quello di prendere una nuova velocità traslazionale e rotazionale in accordo al controllo osservato; la nuova posizione s_t può essere calcolata simulando il nuovo controllo in avanti dalla posizione precedente s_{t-1}.

3.2.3 Aggiornamento delle stime dei landmark

Il FastSLAM rappresenta le stime dei landmark condizionate $p(\theta_n|s^t, z^t, u^t, n^t)$ utilizzando i filtri di Kalman esteso; si assume che le associazioni di dati n^t sono note, condizione che verrà eliminata più avanti nella trattazione.

Ad ogni particella S_t sono connessi N *EKF*, dato che le stime dei landmark sono dipendenti dal percorso del robot; il calcolo del *posterior* relativo all'$n-$

[14] In contrasto all'*EKF* che richiede la linearizzazione del modello di movimento.

[15] Per una trattazione approfondita dei fenomeni di slittamento e della correzione degli errori sull'odometria si veda [12].

$esima$ posizione θ_n del landmark dipende dal fatto se il landmark θ_n è stato o meno osservato al tempo t. Per il landmark osservato θ_{n_t} si utilizza l'usuale regola di Bayes:

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Bayes}{=} \eta\, p(z_t|\theta_{n_t}, s^t, z^{t-1}, u^t, n^t)\, p(\theta_{n_t}|s^t, z^{t-1}, u^t, n^t) \quad (3.33)$$

Si possono semplificare entrambi i membri della (3.33) utilizzando la proprietà di Markov: l'osservazione z_t dipende soltanto da θ_{n_t}, s_t e da n_t. Allo stesso modo θ_{n_t} non è interessata da s_t, u_t o n_t senza l'osservazione z_t.

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Markov}{=} \eta\, p(z_t|\theta_{n_t}, s_t, n_t)\, p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (3.34)$$

Per $n \neq n_t$, il landmark *posterior* rimane invariato:

$$p(\theta_{n\neq n_t}|s^t, z^t, u^t, n^t) = p(\theta_{n\neq n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (3.35)$$

Il FastSLAM implementa l'equazione di aggiornamento (3.34) con un *EKF*; come nelle soluzioni allo SLAM che utilizzano il filtro di Kalman esteso, questo filtro utilizza una approssimazione lineare Gaussiana per il modello percettivo. Come conseguenza del campionamento di una nuova posizione del robot si ha che con un modello di osservazione lineare Gaussiano, la distribuzione risultante $p(\theta_n|s^t, z^t, u^t, n^t)$ è esattamente Gaussiana, anche se il modello di movimento è non lineare.

Il modello di misura non lineare descritto dalla funzione $g(s_t, \theta_{n_t})$ verrà approssimato utilizzando una espansione di Taylor del primo ordine; lo stimatore del landmark è condizionato ad un percorso fissato del robot, così questa espansione è relativa esclusivamente a θ_{n_t}. Assumendo che il rumore di misura è Gaussiano con covarianza R_t, si possono scrivere le seguenti:

$$\hat{z}_t = g(s_t^{[m]}, \mu_{n_t,t-1}) \quad (3.36)$$

$$G_{\theta_{n_t}} = \nabla_{\theta_{n_t}}\, g(s_t, \theta_{n_t})|_{s_t=s_t^{[m]}\,;\,\theta_{n_t}=\mu_{n_t,t-1}^{[m]}} \quad (3.37)$$

$$g(s_t, \theta_{n_t}) \approx \hat{z}_t + G_\theta(\theta_{n_t} - \mu_{n_t,t-1}^{[m]}) \quad (3.38)$$

Utilizzando questa approssimazione, il primo termine del prodotto (3.34) è distribuito come segue:

$$p(z_t|\theta_{n_t}, s_t, n_t) \sim \mathcal{N}(z_t; \hat{z}_t + G_\theta(\theta_{n_t} - \mu_{n_t,t-1}^{[m]}), R_t) \quad (3.39)$$

Il secondo termine del prodotto nella (3.34) è anch'esso Gaussiano e coincide con lo stato dell'*EKF* al tempo $t-1$.

$$p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \sim \mathcal{N}(\theta_{n_t}; \mu^{[m]}_{n_t,t-1}, \Sigma^{[m]}_{n_t,t-1}) \tag{3.40}$$

Il valor medio e la covarianza del prodotto nella (3.34) possono essere ottenute utilizzando le equazioni di aggiornamento dell'*EKF*.

$$\begin{aligned}
\hat{z}_t &= g(s^{[m]}_t, \mu_{n_t,t-1}) && (3.41)\\
G_{\theta_{n_t}} &= \nabla_{\theta_{n_t}} g(s_t, \theta_{n_t})|_{s_t=s^{[m]}_t \,;\, \theta_{n_t}=\mu^{[m]}_{n_t,t-1}} && (3.42)\\
Z_{n,t} &= G_{\theta_{n_t}} \Sigma^{[m]}_{n_t,t-1} G^T_{\theta_{n_t}} + R_t && (3.43)\\
K_t &= \Sigma^{[m]}_{n_t,t-1} G^T_{\theta_{n_t}} Z^{-1}_{n,t} && (3.44)\\
\mu^{[m]}_{n_t,t} &= \mu^{[m]}_{n_t,t-1} + K_t\,(z_t - \hat{z}_t) && (3.45)\\
\Sigma^{[m]}_{n_t,t} &= (I - K_t\, G_{\theta_{n_t}})\, \Sigma^{[m]}_{n_t,t-1} && (3.46)
\end{aligned}$$

Dato che il filtro relativo ad ogni landmark è di dimensione costante, l'aggiornamento del filtro è un'operazione che richiede un tempo di computazione costante; il tempo totale necessario ad aggiungere un'osservazione non dipende dal numero totale di landmark. Come riportato in figura 3.3, il robot osserva la distanza e l'angolo relativi ai landmark vicini. Assumendo che la posizione attuale del robot s_t è descritta come $\langle s_{t,x}, s_{t,y}, s_{t,\theta}\rangle$ e la posizione corrente del landmark è $\langle \theta_{n_t,x}, \theta_{n_t,y}\rangle$; la funzione di misura $g(s_t, \theta_{n_t})$ può essere scritta come in (3.47).

$$g(s_t, \theta_{n_t}) = \begin{bmatrix} r(s_t, \theta_{n_t}) \\ \phi(s_t, \theta_{n_t}) \end{bmatrix} = \begin{bmatrix} \sqrt{(\theta_{n_t,x} - s_{t,x})^2 + (\theta_{n_t,y} - s_{t,y})^2} \\ \tan^{-1}\left(\frac{\theta_{n_t,y} - s_{t,y}}{\theta_{n_t,x} - s_{t,x}}\right) - s_{t,\theta} \end{bmatrix} \tag{3.47}$$

La matrice Jacobiana $G_{\theta_{n_t}}$ risulta pari a:

$$G_{\theta_{n_t}} = \begin{bmatrix} \frac{\theta_{n_t,x} - s_{t,x}}{\sqrt{q}} & \frac{\theta_{n_t,y} - s_{t,y}}{\sqrt{q}} \\ -\frac{\theta_{n_t,y} - s_{t,y}}{q} & \frac{\theta_{n_t,x} - s_{t,x}}{q} \end{bmatrix} \tag{3.48}$$

con:

$$q = (\theta_{n_t,x} - s_{t,x})^2 + (\theta_{n_t,y} - s_{t,y})^2$$

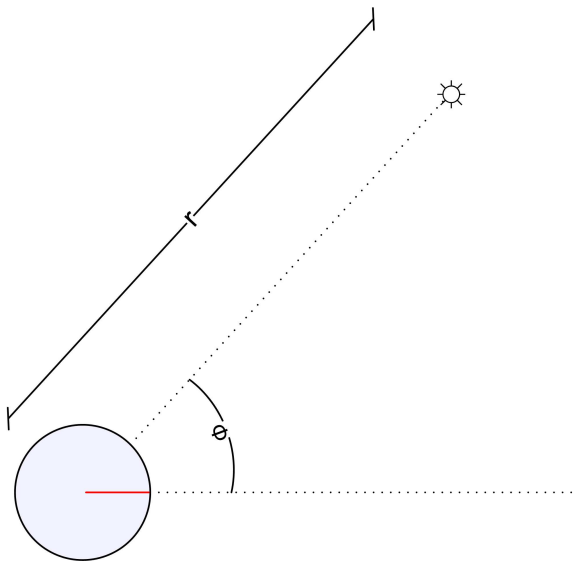

Figura 3.3: *Osservazione del landmark a distanza r e con angolo ϕ.*

3.2.4 Calcolo dei pesi delle particelle

I pesi delle particelle sono definiti come il rapporto tra la *target distribution* e la *proposal distribution*. Facendo l'assunzione che asintoticamente i percorsi in $s^{t-1,[m]}$ sono stati generati in accordo alla *target distribution* al tempo $t-1$, $p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})$, si può ricavare la *proposal distribution* come:

$$p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})\, p(s_t^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t) \tag{3.49}$$

I pesi $w_t^{[m]}$ quindi valgono:

$$w_t^{[m]} = \frac{p(s^{t,[m]}|z^t, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})\, p(s_t^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t)} \tag{3.50}$$

Utilizzando la definizione di probabilità condizionata si può espandere il membro a numeratore:

$$w_t^{[m]} = \frac{p(s_t^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t)\, p(s^{t-1,[m]}|z^t, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})\, p(s_t^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t)} \tag{3.51}$$

e quindi:

$$w_t^{[m]} = \frac{p(s^{t-1,[m]}|z^t, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})} \tag{3.52}$$

Il numeratore si può espandere utilizzando la regola di Bayes:

$$w_t^{[m]} \overset{Bayes}{=} \eta \frac{p(z_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, p(s^{t-1,[m]}|z^{t-1}, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})} \tag{3.53}$$

per la proprietà di Markov si può scrivere:

$$\begin{aligned} w_t^{[m]} &\overset{Markov}{=} \eta \frac{p(z_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})} \\ &= \eta\, p(z_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t) \end{aligned} \tag{3.54}$$

Utilizzando il teorema della probabilità totale due volte, per esprimere la (3.54) rispetto a s_t e a θ_{n_t}.

$$\begin{aligned} w_t^{[m]} &= \eta \int p(z_t|s_t, s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, p(s_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t) ds_t = \\ &\overset{Markov}{=} \eta \int p(z_t|s_t, s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, p(s_t|s^{t-1,[m]}, u^t)\, ds_t = \\ &= \eta \int\int p(z_t|\theta_{n_t}, s_t, s^{t-1,[m]}, z^{t-1}, u^t, n^t) \cdot \\ &\quad \cdot p(\theta_{n_t}|s_t, s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, d\theta_{n_t} p(s_t|s^{t-1,[m]}, u^t)\, ds_t \end{aligned}$$

Utilizzando di nuovo la proprietà di Markov si ha:

$$\begin{aligned} w_t^{[m]} &\overset{Markov}{=} \eta \int\int p(z_t|\theta_{n_t}, s_t, n_t)\, p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1})\, d\theta_{n_t} \cdot \\ &\quad \cdot p(s_t|s^{t-1,[m]}, u^t)\, ds_t \end{aligned} \tag{3.55}$$

I tre termini nella (3.55) sono tutti Gaussiani. Nella (3.56) è riportato il modello di misura, nella (3.57) la stima del landmark al tempo $t-1$ e nella (3.58) il modello di movimento.

$$p(z_t|\theta_{n_t}, s_t, n_t) \sim \mathcal{N}(z_t; g(\theta_{n_t}, s_t), R_t) \tag{3.56}$$

$$p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1}) \sim \mathcal{N}(\theta_{n_t}; \mu_{n_t,t-1}^{[m]}, \Sigma_{n_t,t-1}^{[m]}) \tag{3.57}$$

$$p(s_t|s^{t-1,[m]}, u^t) \sim \mathcal{N}(s_t; \hat{s}_t, P_t) \tag{3.58}$$

Linearizzando g è possibile calcolare $w_t^{[m]}$ in forma chiusa: utilizzando inoltre il teorema di convoluzione, si ottiene la (3.59).

$$w_t^{[m]} \sim \mathcal{N}(z_t; \hat{z}_t, L_t) \tag{3.59}$$

con:

$$L_t = G_s P_t G_s^T + G_\theta \Sigma_{n_t,t-1}^{[m]} + R_t \tag{3.60}$$

Infine, ponendolo in una forma differente, il peso della $m - esima$ particella è dato da:

$$w_t^{[m]} = |2\,\pi\, L_t^{[m]}|^{\frac{1}{2}}\, \exp\left\{\frac{1}{2}(z - \hat{z}_t^{[m]})^T\, L_t^{-1}\, (z - \hat{z}_t^{[m]})\right\} \tag{3.61}$$

3.2.5 *Resampling* delle particelle

Dopo aver assegnato i pesi alle particelle, deve essere preso un nuovo set di particelle S_t da questo set con probabilità proporzionali ai pesi. Esistono molte tecniche per fare il *resampling*[16], una in particolare, l'algoritmo sistematico di Madow [10] è facile da implementare e produce risultati accurati. Una semplice implementazione dell'algoritmo di *resampling* richiede un tempo lineare rispetto al numero N dei landmark; in generale solo una piccola parte del totale dei landmark sarà osservata durante un intervallo, quindi copiare tutte le particelle è l'approccio meno efficiente. Implementazioni più sofisticate (come in [13]) possono essere applicate per ridurre la complessità computazionale a $O(\log N)$.

3.2.6 Il *posterior* del percorso del robot

La fattorizzazione dello SLAM utilizzando il percorso effettuato dal robot può sembrare una cattiva scelta, in quanto la lunghezza delle particelle del FastSLAM tende a crescere nel tempo; in ogni caso nessuna delle equazioni di aggiornamento del FastSLAM dipende dal percorso totale del robot, ma dipende soltanto dalla posizione più recente $s_{t-1}^{[m]}$, che è l'informazione necessaria per aggiornare il set di particelle. È chiaro quindi che per la parametrizzazione di ogni particella l'algoritmo può tralasciare tutte le posizioni del robot esclusa l'ultima: questo permette di ovviare al problema della crescita nel tempo della dimensione delle particelle.

[16] A proposito delle tecniche di *resampling* si veda [3].

3.2.7 L'associazione di dati sconosciuta

La limitazione più grande del FastSLAM finora descritto è l'assunzione che le associazioni di dati n^t sono note, cosa che in pratica succede raramente. In questa sezione il FastSLAM verrà esteso a domini in cui il *mapping* tra le osservazioni ed i landmark non è conosciuto. Nel problema classico dello SLAM, la soluzione all'associazione di dati sconosciuta è quella di scegliere n_t tale che massimizzi la verosimiglianza della misura z_t, date tutte le informazioni disponibili.

$$\hat{n}_t = \arg\max_{n_t} p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t) \tag{3.62}$$

Dove il termine $p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t)$ è chiamato verosimiglianza (*likelihood*) e questo approccio prende il nome di stimatore a massima verosimiglianza (*maximum likelihood* - ML). L'associazione di dati ML prende anche il nome di *nearest neighbour*, e la *negative log likelihood* è interpretata come una distanza: quest'ultima è anche chiamata distanza di Mahalanobis, e quindi lo stimatore ML seleziona le associazioni di dati minimizzando la distanza di Mahalanobis.

Nell'*EKF* l'associazione di dati è singola, e questo porta a degli errori che si ripercuotono nella costruzione della mappa talvolta con conseguenze gravi. Per capire quando avvengono questi errori occorre studiare l'incertezza nell'associazione dei dati.

3.2.8 L'incertezza nell'associazione dei dati

L'incertezza nello *SLAM posterior* è dovuta sostanzialmente a due fattori:

- rumore di misura;
- rumore di movimento.

Effetto del rumore di misura sull'associazione di dati

All'aumentare del rumore di misura, le distribuzioni delle possibili osservazioni di ogni landmark diventano sempre più incerte: se il rumore di misura è abbastanza

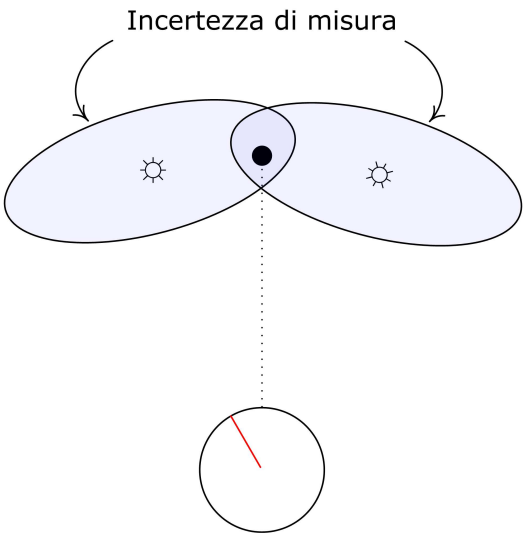

Figura 3.4: *Ambiguità di misura.*

elevato, le distribuzioni delle osservazioni di landmark vicini tendono a sovrapporsi, creando ambiguità nell'identità dei landmark. In figura 3.4 è riportato un esempio di ambiguità di misura: le due ellissi rappresentano le possibili osservazioni di due differenti landmark. L'osservazione, rappresentata con un punto nero, può essere stata generata sia da un landmark che dall'altro. Attribuire un'osservazione al landmark sbagliato, chiaramente aumenta l'errore nella mappa e nella posizione del robot. Dato che l'osservazione può essere stata generata da due landmark vicini con la stessa probabilità, l'effetto dell'osservazione sulle posizioni dei landmark e del robot sarà piccolo; la covarianza di un landmark sarà leggermente sovrastimata, mentre quella dell'altro sarà sottostimata.

Effetto del rumore di movimento sull'associazione di dati

L'ambiguità nell'associazione di dati causata dal rumore di movimento può avere conseguenze gravi sulla stima. Elevato rumore di movimento porta ad alta incertezza nella posizione, dopo aver effettuato un controllo; se questa incertezza è abbastanza elevata, la scelta di posizioni diverse nella distribuzione delle posizioni del robot porta a differenze sostanziali nelle ipotesi di associazione di dati per le

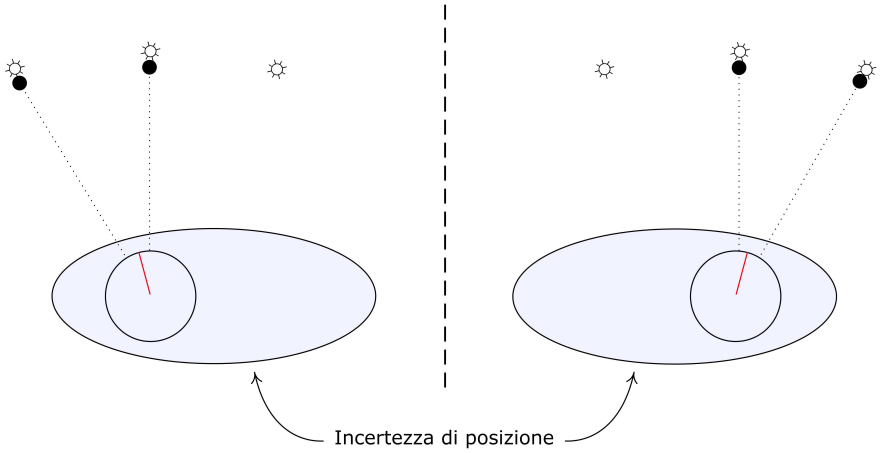

Figura 3.5: *Ambiguità di posizione.*

successive osservazioni. L'ambiguità nel movimento è facilmente prodotta da un significativo errore nella rotazione del robot; se l'algoritmo di SLAM scegliesse l'associazione di dati sbagliata per una singola osservazione, con molta probabilità anche tutte le rimanenti associazioni di dati sarebbero sbagliate. Inoltre la scelta di un gran numero di associazioni di dati sbagliate porta alla divergenza nel filtro di Kalman esteso. In figura 3.5 è riportato un esempio di ambiguità di posizione.

Associazione di dati *per-particle*

Come già illustrato nella sezione 2.1 l'*EKF* ha un approccio a singola ipotesi per l'associazione di dati, mentre il FastSLAM utilizza un approccio a ipotesi multipla. Ogni particella infatti rappresenta un differente percorso ipotizzato per il robot, e quindi le decisioni per l'associazione di dati possono essere prese su base particellare. Le particelle che prendono la corretta associazione di dati ricevono un peso maggiore, mentre particelle che prendono associazioni di dati sbagliate ricevono pesi bassi e possono successivamente venire rimosse nel passo di *resampling*.

L'associazione di dati *per-particle* ha molti vantaggi rispetto all'associazione di dati standard ML; in primo luogo, dato che l'ambiguità di movimento, come

già detto, è la forma più grave di associazione di dati ambigua, condizionare le decisioni di associazione di dati sui percorsi ipotizzati del robot sembrerebbe la scelta migliore. Inoltre l'associazione di dati su base *per-particle* rende il problema di più facile risoluzione. Nell'*EKF* l'incertezza della posizione del landmark è dovuta sia all'incertezza nella posizione del robot che all'incertezza dovuta al rumore di misura. Nel FastSLAM, invece, l'incertezza sulla posizione del robot è rappresentata dall'intero set di particelle. I filtri relativi ai landmark in ogni singola particella non sono affetti dal rumore di movimento perché sono condizionati ad un percorso specifico del robot[17].

Un altro vantaggio dell'associazione di dati *per-particle basis* è che in un dato istante alcune particelle possono ricevere una determinata associazione di dati, plausibile, ma non corretta; in seguito però, il robot può effettuare nuove osservazioni che confutano le scelte fatte in precedenza: le particelle con le associazioni di dati sbagliate riceveranno quindi un peso basso ed eventualmente verranno eliminate dal filtro. Questo significa che l'effetto di un'associazione di dati sbagliata fatta in un istante passato può essere rimosso dal filtro nel momento in cui nuove osservazioni confermano l'inesattezza delle associazioni di dati passate. Il *resampling* è l'unico mezzo, non euristico, che permette di correggere in modo statisticamente valido le vecchie associazioni di dati sbagliate.

Associazione di dati ML *per-particle* L'approccio più semplice per un'associazione di dati *per-particle* è quello di applicare l'associazione di dati ML estesa ad ogni particella. Dato che gli stimatori dei landmark sono *EKF*, la verosimiglianza in (3.62) può essere calcolata utilizzando le innovazioni: nella (3.63) è riportato il valore della verosimiglianza; se il valore di questa cade al di sotto di una soglia p_0, viene aggiunto un nuovo landmark alla particella.

$$p(z_t|s^{t,[m]}, z^{t-1}, u^t, n^{t-1}) = \frac{1}{\sqrt{|2\,\pi\, Z_{n,t}|}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_{n,t})^T Z_{n,t}^{-1}(z_t - \hat{z}_{n,t})\right\} \tag{3.63}$$

[17] Questo è particolarmente utile se il robot ha un movimento molto affetto da errore, mentre possiede sensori molto accurati.

Dato che la componente più consistente dell'ambiguità nell'associazione di dati deriva dall'incertezza nella posizione del robot, l'associazione di dati ML si comporta meglio nel FastSLAM piuttosto che nell'*EKF*. Una parte delle particelle prenderanno nuove posizioni del robot più consistenti con la posizione reale del robot; queste posizioni riceveranno associazioni di dati corrette e giustificheranno bene le osservazioni. Le particelle che prenderanno posizioni lontane da quella reale del robot, invece, riceveranno associazioni di dati inesatte.

3.2.9 Aggiunta di nuovi landmark

L'aggiunta di nuovi landmark può essere una decisione difficile da prendere, così come negli algoritmi basati sul filtro di Kalman esteso; questo è vero particolarmente quando una sola misura è insufficiente a descrivere un landmark in tutte le sue dimensioni. Se la funzione di misura $g(\theta_{n_t}, s_t)$ è invertibile è comunque possibile inizializzare un nuovo landmark con una singola misura. Ogni osservazione definisce una Gaussiana:

$$\mathcal{N}(z_t; \hat{z}_t + G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t}), R_t) \tag{3.64}$$

questa Gaussiana può essere espressa in maniera esplicita come:

$$\frac{1}{\sqrt{|2\,\pi\, Z_{n,t}|}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))^T \cdot \right. \\ \left. \cdot R_t^{-1}\,(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))\right\} \tag{3.65}$$

Si definisce la funzione J come segue:

$$J = \frac{1}{2}(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))^T\, R_t^{-1}\,(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1})) \tag{3.66}$$

La derivata seconda di J rispetto a θ_{n_t} è l'inversa della matrice di covarianza della Gaussiana nelle coordinate del landmark.

$$\frac{\partial J}{\partial \theta_{n_t}} = -(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))^T\, R_t^{-1}\, G_{\theta_{n_t}} \tag{3.67}$$

$$\frac{\partial^2 J}{\partial \theta^2_{n_t}} = G^T_{\theta_{n_t}}\, R_t^{-1}\, G_{\theta_{n_t}} \tag{3.68}$$

Di conseguenza una osservazione può essere utilizzata per creare un nuovo landmark come segue:

$$\mu_{n_t,t}^{[m]} = g^{-1}(s_t^{[m]}, z_t) \tag{3.69}$$

$$\Sigma_{n_t,t}^{[m]} = \left(G_{\theta_{n_t},t}^T R^{-1} G_{\theta_{n_t},t}\right)^{-1} \tag{3.70}$$

$$w_t^{[m]} = p_0 \tag{3.71}$$

Invece di utilizzare le (3.69), (3.70), (3.71) si può utilizzare una procedura di inizializzazione più semplice per aggiungere un nuovo landmark; invece di calcolare la covarianza iniziale corretta, la covarianza può essere calcolata impostando la varianza di ogni parametro dei landmark a un valore alto ed incorporare la prima osservazione.

$$\mu_{n_t,t}^{[m]} = g^{-1}(s_t^{[m]}, z_t) \tag{3.72}$$

$$\Sigma_{n_t,t}^{[m]} = K \cdot I \tag{3.73}$$

Dove alti valori di K portano ad una approssimazione della vera covarianza, ma possono portare anche ad instabilità numerica.

3.3 Miglioramenti del FastSLAM

In questa sezione verranno introdotti dei miglioramenti al FastSLAM illustrato nelle precedenti sezioni. La versione finale dell'algoritmo è quella utilizzata nell'implementazione descritta nel capitolo 4.

Il miglioramento principale è sul *sampling* di una nuova posizione del robot: sostanzialmente quando si campiona una nuova posizione, la nuova *proposal distribution* deve tener conto non solo del controllo corrente, ma anche delle misure sensoriali più recenti; per ottenere una distribuzione che si uniformi a questo, il FastSLAM linearizza il modello di movimento allo stesso modo in cui è linearizzato nello SLAM basato su *EKF*. In questo modo la *proposal distribution* può essere calcolata in forma chiusa. Approcci di questo tipo sono stati già proposti da Doucet e altri in [4].

Concettualmente questa modifica è semplice, ma necessita di particolari accortezze. In primo luogo occorre dimostrare la convergenza dell'algoritmo con una sola particella. Secondariamente occorre modificare la procedura per la scelta dei campioni (e quindi la *proposal distribution*).

3.3.1 Miglioramento della *proposal distribution*

Come già mostrato in precedenza vengono riportati i modelli di misura e di movimento come funzioni non lineari con rumore Gaussiano:

$$p(z_t|s_t, \Theta, n_t) = g(s_t, \theta_{n_t}) + \epsilon_t \tag{3.74}$$

$$p(s_t|u_t, s_{t-1}) = h(s_{t-1}, u_t) + \delta_t \tag{3.75}$$

dove ϵ_t e δ_t sono variabili che rappresentano il rumore Gaussiano con covarianza R_t e P_t rispettivamente.

Invece di prendere una nuova posizione s_t dal modello di movimento già introdotto $p(s_t|u_t, s_{t-1})$, il FastSLAM prende una nuova posizione dal seguente modello di movimento, che tiene conto anche dell'ultima osservazione z_t:

$$s_t^{[m]} \sim p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \tag{3.76}$$

dove $s^{t-1,[m]}$ è il percorso del robot fino al tempo $t-1$ relativo alla $m-esima$ particella. La (3.76) incorpora la misura sensoriale più recente z_t, la sua associazione di dati n_t ed il controllo u_t, che insieme formano la nuova informazione disponibile al tempo t.

Per effettuare il *sampling* dalla (3.76) occorre un'analisi più approfondita: in primo luogo il nuovo modello di movimento deve essere riscritto in termini di distribuzioni note, includendo anche i modelli di movimento e di misura, oltre che le stime delle *feature*. Come primo passo occorre espandere la *proposal distribution* utilizzando la regola di Bayes:

$$p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \overset{Bayes}{\propto}$$
$$\overset{Bayes}{\propto} \eta\, p(z_t|s_t, s^{t-1,[m]}, u^t, z^{t-1}, n^t)\, p(s_t|s^{t-1,[m]}, u^t, z^{t-1}, n^t) \tag{3.77}$$

La posizione del robot s_t al secondo termine dipende solo dalla posizione al tempo $t-1$, s_{t-1} e dal controllo u_t. Per la proprietà di Markov, si può scrivere:

$$p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \overset{Markov}{=} \eta\, p(z_t|s_t, s^{t-1,[m]}, u^t, z^{t-1}, n^t)\, p(s_t|s_{t-1}^{[m]}, u_t) \tag{3.78}$$

Il teorema della probabilità totale è utilizzato come segue:

$$\begin{aligned} p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \;=\; \eta \int p(z_t|\theta_{n_t}, s_t, s^{t-1,[m]}, u^t, z^{t-1}, n^t) \cdot \\ \cdot p(\theta_{n_t}|s_t, s^{t-1,[m]}, u^t, z^{t-1}, n^t)\, d\theta_{n_t}\, p(s_t|s_{t-1}^{[m]}, u_t) \end{aligned} \tag{3.79}$$

Il primo termine nell'integrale non è altro che il modello di misura $p(z_t|s_t, \theta_{n_t}, n_t)$. Il secondo termine può essere semplificato in quanto s_t, n_t e u_t non forniscono informazioni riguardo a θ_{n_t} senza z_t.

$$\begin{aligned} p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \overset{Markov}{=} \eta \int p(z_t|\theta_{n_t}, s_t, n_t) \cdot \\ \cdot p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1})\, d\theta_{n_t}\, p(s_t|s_{t-1}^{[m]}, u_t) \end{aligned} \tag{3.80}$$

con:

$$p(z_t|\theta_{n_t}, s_t, n_t) \;\sim\; \mathcal{N}(z_t; g(s_t, \theta_{n_t}), R_t) \tag{3.81}$$

$$p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1}) \;\sim\; \mathcal{N}(\theta_{n_t}; \mu_{n_t,t-1}^{[m]}, \Sigma_{n_t,t-1}^{[m]}) \tag{3.82}$$

$$p(s_t|s_{t-1}^{[m]}, u_t) \;\sim\; \mathcal{N}(s_t; h(s_{t-1}^{[m]}, u_t), P_t) \tag{3.83}$$

L'espressione (3.80) mostra che la *sampling distribution* è la convoluzione di due Gaussiane; la funzione g può essere sostituita da un'approssimazione lineare, con un'espansione di Taylor al primo ordine.

$$\hat{s}_t \;=\; h(s_{t-1}^{[m]}, u_t) \tag{3.84}$$

$$\hat{z}_t \;=\; g(\hat{s}_t, \mu_{n_t,t-1}^{[m]}) \tag{3.85}$$

$$G_\theta \;=\; \nabla_{\theta_{n_t}}\, g(s_t, \theta_{n_t})\big|_{s_t=\hat{s}_t, \theta_{n_t}=\mu_{n_t,t-1}^{[m]}} \tag{3.86}$$

$$G_s \;=\; \nabla_{s_t}\, g(s_t, \theta_{n_t})\big|_{s_t=\hat{s}_t, \theta_{n_t}=\mu_{n_t,t-1}^{[m]}} \tag{3.87}$$

$$g(s_t, \theta_{n_t}) \;\approx\; \hat{z}_t + G_\theta(\theta_{n_t} - \mu_{n_t,t-1}^{[m]}) + G_s(s_t - \hat{s}_t) \tag{3.88}$$

per una data particella, $\hat{s}_t$ e $\hat{z}_t$ possono essere interpretate rispettivamente come la posizione predetta e l'osservazione predetta al tempo t; le matrici G_θ e G_s sono gli Jacobiani di g: ossia sono rispettivamente le derivate di g rispetto a θ_{n_t} e a s_t.

Dal teorema di convoluzione si ottiene:

$$\mathcal{N}(z_t; \hat{z}_t + G_s s_t - G_s \hat{s}_t, Z_t) \tag{3.89}$$

dove:

$$Z_t = R_t + G_\theta \Sigma^{[m]}_{n_t,t-1} G_\theta^T \tag{3.90}$$

è la covarianza della Gaussiana riportata in (3.89). La *proposal distribution* è data dal prodotto della Gaussiana e del termine più a destra nella (3.80); il prodotto può essere scritto come:

$$p(s_t | s^{t-1,[m]}, z^t, u^t, n^t) = \xi \, \exp\{-y_t\} \tag{3.91}$$

dove l'esponente è:

$$y_t = \frac{1}{2} \left[(z - \hat{z}_t - G_s s_t + G_s \hat{s}_t)^T Z_t^{-1} (z - \hat{z}_t - G_s s_t + G_s \hat{s}_t) + \right. \\ \left. + (s_t - \hat{s}_t)^T P_t^{-1} (s_t - \hat{s}_t) \right] \tag{3.92}$$

L'espressione per l'esponente è quadratica nella variabile s_t, quindi la distribuzione (3.91) è Gaussiana. La media e la covarianza della (3.91) sono date dalla derivata prima e seconda di y_t rispetto a s_t:

$$\begin{aligned} \frac{\partial y_t}{\partial s_t} &= -G^T Z_t^{-1} (z_t - \hat{z}_t - G_s s_t + G_s \hat{s}_t) + P_t^{-1} (s_t - \hat{s}_t) = \\ &= (G_s^T Z_t^{-1} G_s + P_t^{-1}) \, s_t - G_s^T Z_t^{-1} (z_t - \hat{z}_t + G_s \hat{s}_t) - P_t^{-1} \hat{s}_t \end{aligned} \tag{3.93}$$

$$\frac{\partial^2 y_t}{\partial s_t^2} = G_s^T Z_t^{-1} G_s + P_t^{-1} \tag{3.94}$$

La covarianza $\Sigma^{[m]}_{s_t}$ della *sampling distribution* si ottiene invertendo la (3.94).

$$\Sigma^{[m]}_{s_t} = \left[G_s^T Z_t^{-1} G_s + P_t^{-1} \right] \tag{3.95}$$

La media $\mu_{s_t}^{[m]}$ della *sample distribution* si ottiene ponendo a zero la (3.93):

$$\begin{aligned}\mu_{s_t}^{[m]} &= \Sigma_{s_t}^{[m]}\left[G_s^T Z_t^{-1}(z_t - \hat{z}_t + G_s\hat{s}_t^{[m]}) + P_t^{-1}\hat{s}_t\right] \\ &= \Sigma_{s_t}^{[m]}G_s^T Z_t^{-1}(z_t - \hat{z}_t) + \Sigma_{s_t}^{[m]}\left[G_s^T Z_t^{-1}G_s + P_t^{-1}\right]\hat{s}_t^{[m]} \\ &= \Sigma_{s_t}^{[m]}G_s^T Z_t^{-1}(z_t - \hat{z}_t) + \hat{s}_t^{[m]} \qquad (3.96)\end{aligned}$$

Le (3.95) e (3.96) parametrizzano l'approssimazione Gaussiana del FastSLAM alla nuova *sampling distribution* $p(s_t|s^{t-1,[m]}, z^t, u^t, n^t)$; questa Gaussiana è costruita per ogni particella in S_{t-1}, ed un nuovo campione è preso e posizionato in un set di particelle temporaneo.

3.3.2 Estensione a osservazioni multiple

Per incorporare osservazioni multiple, l'ipotesi semplificativa di incorporare sequenzialmente le osservazioni non può essere applicata con i miglioramenti descritti: la *proposal distribution* infatti prende anche le osservazioni prima che i filtri di ogni landmark siano aggiornati.

L'aggiunta di una osservazione nella *proposal distribution* è equivalente ad una iterazione di un aggiornamento dell'*EKF* per la misura. Osservazioni multiple possono essere aggiunte applicando l'aggiornamento dell'*EKF* più volte, una per ogni osservazione: all'aumentare delle osservazioni aggiunte, la *proposal distribution* continua a restringersi. Un campione è preso dalla *proposal distribution* dopo aver aggiunto ogni osservazione, in modo da aggiornare i filtri dei landmark e da calcolare i pesi delle particelle.

La *proposal distribution* iniziale è posta come una Gaussiana con media $\hat{s}_t$ e covarianza P_t, successivamente la *proposal distribution* viene aggiornata come segue:

$$\Sigma_{s_t,0} = P_t \qquad (3.97)$$

$$\mu_{s_t,0} = \hat{s}_t \qquad (3.98)$$

$$\Sigma_{s_t,n} = \left[G_{s,n}^T Z_{t,n}^{-1} + \Sigma_{s_t,n-1}^{-1}\right]^{-1} \qquad (3.99)$$

$$\mu_{s_t,n} = \mu_{s_t,n-1} + \Sigma_{s_t,n}G_{s_t,n}^T Z_{t,n}^{-1}\left(z_t - \hat{z}_{t,n}\right) \qquad (3.100)$$

Quando vengono incorporate osservazioni multiple, occorre porre particolare attenzione alla creazione di nuovi landmark. Se la prima delle K osservazioni simultanee predice l'aggiunta di un nuovo landmark, la nuova posizione del robot $s_t^{[m]}$ verrà presa dal modello di movimento $p(s_t|s_{t-1}, u_t)$; se questo modello è affetto da rumore, la posizione iniziale del nuovo landmark potrebbe essere non accurata. Se l'osservazione che genera un nuovo landmark è processata per ultima nelle K osservazioni, allora la nuova posizione del robot $s_t^{[m]}$ sarà presa dalla distribuzione modificata $\mathcal{N}(\theta_n, \mu_{s_t,n}, \Sigma_{s_t,n})$, che è stata ridefinita dalle $K-1$ osservazioni precedenti: in questo modo la posizione iniziale del landmark sarà molto più accurata.

Come regola generale quindi, se in un istante vengono incorporate più osservazioni, quelle che generano nuovi landmark devono essere processate per ultime[18]. Anche la conoscenza della configurazione attuale del robot può essere utilizzata per ordinare le osservazioni in modo tale da processare per ultimi i nuovi landmark.

La scelta accurata dell'ordinamento delle osservazioni assicura un'accuratezza nella posizione dei nuovi landmark, e quindi in generale un'accuratezza maggiore nella costruzione della mappa.

3.3.3 Convergenza del FastSLAM

Il FastSLAM ha la caratteristica che può convergere con una sola particella, particolare molto inusuale per i filtri particellari; dato che non serve *resampling*, perché c'è solo una particella, il FastSLAM in questo caso è un algoritmo a tempo costante. Il tempo richiesto per incorporare una nuova osservazione non è affetto dalla dimensione della mappa.

Nel filtro particellare standard, se c'è una sola particella, il *resampling* non può cambiare la traiettoria del robot, ed il filtro diverge. Nel FastSLAM invece, la *proposal distribution* incorpora anche le osservazioni, e con una singola particella

[18] Ad esempio processandole in due passi.

il modello di movimento può minimizzare l'errore della mappa, proponendo una nuova posizione per il robot conforme alle ultime osservazioni.

In questa sezione verrà dimostrata la convergenza del FastSLAM per una sola particella, nel caso particolare dello SLAM lineare e Gaussiano (*LG-SLAM*). Dimostrare la convergenza del filtro per una sola particella implica dimostrare la convergenza per tutte le M particelle.

La dimostrazione porta a due risultati importanti: il primo risultato è la dimostrazione della convergenza per un algoritmo a tempo costante. Il secondo risultato implica che mantenere l'intera matrice di covarianza non è necessario a garantire la convergenza.

Come già detto verrà provata la convergenza per il caso particolare di *LG-SLAM*: i problemi di *LG-SLAM* hanno i seguenti modelli di movimento e di misura:

$$g(s_t, \theta_{n_t}) = \theta_{n_t} - s_t \tag{3.101}$$

$$h(u_t, s_{t-1}) = u_t + s_{t-1} \tag{3.102}$$

La struttura dell'*LG-SLAM* può essere vista come un robot che opera in uno spazio Cartesiano, dotato di bussola priva di rumore, e da sensori che misurano le distanze dalle *feature* lungo gli assi coordinati. In questo scenario il *mapping* tra le osservazioni ed i landmark è supposto essere noto.

La proprietà di convergenza principale del FastSLAM è la seguente:

Teorema 1 *Il FastSLAM Lineare-Gaussiano converge alla mappa reale con* $M = 1$ *particelle se tutte le* feature *sono osservate infinitamente, e se la posizione di una* feature *è nota a priori.*

Dimostrazione della convergenza del FastSLAM

Prima di mostrare la prova, è necessario riformulare le equazioni di aggiornamento per il problema SLAM-LG, con le apposite definizioni di g ed h della (3.101)

e (3.102). Si inizia con il riscrivere le equazioni per il calcolo della *proposal ditribution*:

$$\hat{s}_t = h(s_{t-1}^{[m]}, u_t) = s_{t-1}^{[m]} + u_t \tag{3.103}$$

$$\hat{z}_t = g(\hat{s}_t, \mu_{n_t,t-1}^{[m]}) = \mu_{n_t,t-1}^{[m]} - s_{t-1}^{[m]} - u_t \tag{3.104}$$

$$G_\theta = \nabla_{\theta_n} g(s_t, \theta_n)|_{s_t=\hat{s}_t;\theta_n=\mu_{n_t,t-1}^{[m]}} = I \tag{3.105}$$

$$G_s = \nabla_{s_t} g(s_t, \theta_n)|_{s_t=\hat{s}_t;\theta_n=\mu_{n_t,t-1}^{[m]}} = -I \tag{3.106}$$

$$Z_t = R_t + G_\theta \Sigma_{n_t,t-1}^{[m]} G_\theta^T = R_t + \Sigma_{n_t,t-1}^{[m]} \tag{3.107}$$

$$\Sigma_{s_t} = \left[G_s^T Z_t G_s + P_t^{-1}\right]^{-1} = \left[(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} + P_t^{-1}\right]^{-1} \tag{3.108}$$

$$\mu_{s_t} = \Sigma_{s_t} G_s^T Z_t^{-1}(z_t - \hat{z}_t) + \hat{s}_t \tag{3.109}$$

$$= -\Sigma_{s_t}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} \cdot \tag{3.110}$$

$$\cdot(z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + s_{t-1}^{[m]} + u_t \tag{3.111}$$

$$s_t^{[m]} \sim \mathcal{N}(s_t;\, \mu_{s_t}, \Sigma_{s_t}) \tag{3.112}$$

Analogamente le equazioni di aggiornamento degli stimatori di posizione dei landmark sono descritte dalle:

$$\mu_{n_t,t} = \mu_{n_t,t-1} + \Sigma_{n_t,t-1}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) \tag{3.113}$$

$$\Sigma_{n_t,t} = (I - \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1})\Sigma_{n_t,t-1}^{[m]} \tag{3.114}$$

Le equazioni per la computazione dei pesi può essere ignorata poiché questa dimostrazione vale per $M = 1$ particelle. La procedura di *resampling* del filtro particellare non è quindi più applicata. È necessario, ai fini della dimostrazione, introdurre le variabili d'errore $\alpha_t^{[m]}$ e $\beta_{n_t,t}^{[m]}$, rispettivamente indicanti l'errore assoluto sulla posizione del robot e sulla posizione del landmark, al tempo t:

$$\alpha_t^{[m]} = s_t^{[m]} - s_t \tag{3.115}$$

$$\beta_{n_t,t}^{[m]} = \mu_{n,t}^{[m]} - \theta_n \tag{3.116}$$

Il landmark con posizione conosciuta viene identificato come *anchoring feature* e si assume essere il landmark θ_1, senza perdita di generalità. La prova di conver-

genza del FastSLAM è verificata tramite dei lemmi. Il primo Lemma caratterizza gli effetti degli errori della mappa β con gli errori sulla posizione α:

Lemma 1 *Se l'errore* $\beta_{n_t,t}^{[m]}$ *della* feature z_t *osservata è di grandezza inferiore all'errore di posizione* $\alpha_t^{[m]}$*, allora* $\alpha_t^{[m]}$ *diminuisce nel suo valore atteso come risultato della misura. Se, invece,* $\beta_{n_t,t}^{[m]}$ *è di grandezza maggiore di* $\alpha_t^{[m]}$*, quest'ultimo può crescere ma nel valore atteso non eccederà mai* $\beta_{n_t,t}^{[m]}$*.*

Dimostrazione del Lemma 1: L'errore atteso della posizione del robot al tempo t è dato da:

$$E[\alpha_t^{[m]}] = E[s_t^{[m]} - s_t] = E[s_t^{[m]}] - E[s_t] \tag{3.117}$$

Il primo termine può esser ricalcolato tramite la distribuzione di *sampling* data dalla (3.112), e il secondo termine è ottenuto dal modello di moto lineare della (3.102):

$$\begin{aligned} E[\alpha_t^{[m]}] &= E[-\Sigma_{s_t}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + s_{t-1}^{[m]} - u_t] \\ &\quad -E[u_t + s_{t-1}] \\ &= -\Sigma_{s_t}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(E[z_t] - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + \underbrace{s_{t-1}^{[m]} - s_{t-1}}_{\alpha_{t-1}^{[m]}} \end{aligned} \tag{3.118}$$

È possibile notare che nello SLAM-LG l'attesa è $E[z_t] = \theta_{n_t} - E[s_t] = \theta_{n_t} - u_t - s_{t-1}$, da questo l'espressione tra parentesi nella (3.118) diventa:

$$\begin{aligned} E[z_t] - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t &= \theta_{n_t} - u_t - s_{t-1} - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t \\ &= \theta_{n_t} - s_{t-1} - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} \\ &= \alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]} \end{aligned} \tag{3.119}$$

Utilizzando questo risultato nella (3.118) e sostituendo $\Sigma_{s_t}^{[m]}$ dalla (3.109) risulta:

$$\begin{aligned} E[\alpha_t^{[m]}] &= \alpha_{t-1}^{[m]} + \Sigma_{s_t}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(\beta_{n_t,t-1}^{[m]} - \alpha_{t-1}^{[m]}) \\ &= \alpha_{t-1}^{[m]} + \left[(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} + P_t^{-1}\right]^{-1}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(\beta_{n_t,t-1}^{[m]} - \alpha_{t-1}^{[m]}) \\ &= \alpha_{t-1}^{[m]} + \left[I + (R_t + \Sigma_{n_t,t-1}^{[m]})P_t^{-1}\right]^{-1}(\beta_{n_t,t-1}^{[m]} - \alpha_{t-1}^{[m]}) \end{aligned} \tag{3.120}$$

Le matrici R_t, $\Sigma^{[m]}_{n_t,t-1}$ e P_t^{-1} sono semidefinite positive, perciò l'inversa di $I + (R_t + \Sigma^{[m]}_{n_t,t-1})P_t^{-1}$ sarà anch'essa semidefinita positiva, con autovalori tutti minori di uno. Questa osservazione prova il Lemma 1. In particolare l'errore della posizione attesa $\alpha^{[m]}_t$ si restringe se $\beta^{[m]}_{n_t,t-1}$ è più piccolo di $\alpha^{[m]}_{t-1}$. Succede il contrario, invece, se $\alpha^{[m]}_{t-1}$ è più piccolo di $\beta^{[m]}_{n_t,t-1}$. L'equazione (3.120) evidenzia che $\alpha^{[m]}_t$ incrementerà nel suo valore atteso, ma solamente di un valore proporzionale alla differenza. Questo ci assicura che $\alpha^{[m]}_t$ non supera $\beta^{[m]}_{n_t,t}$ nel suo valore atteso.

Lemma 2 *Se il robot osserva una* anchoring feature*, il valore atteso dell'errore sulla posizione del robot diminuirà.*

Dimostrazione del Lemma 2: Per una *anchoring feature* θ_1, si può sfruttare il fatto che $\Sigma^{[m]}_{1,t} = \beta^{[m]}_{1,t} = 0$. Per cui la dimostrazione del Lemma 2 viene direttamente dalla (3.120):

$$\begin{aligned} E[\alpha^{[m]}_t] &= \alpha^{[m]}_{t-1} + \left[I + (R_t + 0)P_t^{-1}\right]^{-1}(0 - \alpha^{[m]}_{t-1}) \\ &= \alpha^{[m]}_{t-1} - \left[I + R_t P_t^{-1}\right]^{-1}\alpha^{[m]}_{t-1}) \end{aligned} \tag{3.121}$$

Quindi, se il robot osserva una *anchoring feature*, l'errore sulla posizione $\alpha^{[m]}_t$ è garantito diminuire. Se l'errore è già zero, allora il valore atteso rimane nullo.

Lemma 3 *Se l'errore di posizione $\alpha^{[m]}_{t-1}$ è di grandezza più piccola dell'errore $\beta^{[m]}_{n_t,t-1}$ della* feature *osservata, osservare s_t diminuisce $\beta^{[m]}_{n_t,t}$ nel valore atteso. Se, invece, $\alpha^{[m]}_{t-1}$ è più grande dell'errore di misura $\beta^{[m]}_{n_t,t}$, quest'ultimo può aumentare, ma non superare il valore atteso di $\alpha^{[m]}_{t-1}$.*

Dimostrazione del Lemma 3: La dimostrazione di questo Lemma è analoga a quella del Lemma 1. Dalla (3.113) segue che l'errore atteso della stima del *landmark*, dopo l'aggiornamento della sua posizione risulta essere:

$$\begin{aligned} E[\beta^{[m]}_{n_t,t}] &= E[\mu^{[m]}_{n_t,t} - \theta_{n_t}] = E[\mu^{[m]}_{n_t,t}] - \theta_{n_t} \\ &= E[\mu^{[m]}_{n_t,t-1} + \Sigma^{[m]}_{n_t,t-1}(R_t + \Sigma^{[m]}_{n_t,t-1})^{-1}(z_t - \mu^{[m]}_{n_t,t-1} + s^{[m]}_{t-1} + u_t)] - \theta_{n_t} \\ &= \mu^{[m]}_{n_t,t} + \Sigma^{[m]}_{n_t,t-1}(R_t + \Sigma^{[m]}_{n_t,t-1})^{-1}(E[z_t] - \mu^{[m]}_{n_t,t-1} + s^{[m]}_{t-1} + u_t)] - \theta_{n_t} \end{aligned} \tag{3.122}$$

Con la (3.119) è possibile scrivere la precedente come:

$$\begin{aligned} E[\beta_{n_t,t}^{[m]}] &= \mu_{n_t,t-1}^{[m]} + \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(\alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]})] - \theta_{n_t} \\ &= \beta_{n_t,t-1}^{[m]} + \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(\alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]})] \\ &= \beta_{n_t,t-1}^{[m]} + (I + R_t\Sigma_{n_t,t-1}^{[m]-1})^{-1}(\alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]})] \end{aligned}$$

Le matrici $\Sigma_{n_t,t-1}^{[m]}$ e R_t sono semidefinite positive, mentre per la $(I+R_t\Sigma_{n_t,t-1}^{[m]-1})^{-1}$ gli autovalori sono tutti positivi e minori di uno, che prova il Lemma 3.

Dimostrazione del teorema: Sia $\hat{\beta}_t^{[m]}$ l'errore più grande delle *feature* al tempo t:

$$\hat{\beta}_t^{[m]} = \arg\max_{\beta_{n_t,t}^{[m]}} |\beta_{n_t,t}^{[m]}| \tag{3.123}$$

Il Lemma 3 dichiara che il valore atteso di questo errore può aumentare, ma soltanto se l'errore assoluto della posizione del robot $\alpha_{t-1}^{[m]}$ supera in grandezza questo errore. Tuttavia, questo varrà per poche iterazioni. In particolare il Lemma 1 garantisce che il valore atteso di $\alpha_{t-1}^{[m]}$ può solamente diminuire se questo accadesse. Oltretutto, il Lemma 2 afferma che ogni volta che la *anchoring feature* viene osservata, questo errore diminuisce di una quantità finita, qualunque sia la grandezza di $\hat{\beta}_t^{[m]}$. Perciò $\alpha_{t-1}^{[m]}$ diventerà (nel suo valore atteso) più piccola in grandezza dell'errore più grande delle *feature*. Appena avviene questo, il Lemma 3 afferma che il valore atteso dell'errore più grande delle *feature* diminuisce ogni volta che la *feature*, a cui è associato questo errore, è osservata. È quindi immediato riconoscere che sia $\alpha_t^{[m]}$ che $\hat{\beta}_t^{[m]}$ convergono a zero. L'osservazione dell'*anchoring feature* induce una riduzione finita (si veda la (3.120)). Per incrementare $\alpha_{t-1}^{[m]}$ al suo precedente valore atteso, l'errore complessivo atteso sulle *feature* diminuisce secondo la (3.123). Questa procedura porta a un asintotica diminuzione dell'errore sulle *feature* a zero. Considerando che questo errore è un limite superiore per l'errore di posizione atteso (Lemma 1), si ottiene la convergenza del valore atteso dell'errore sulla posizione del robot. Il Teorema di convergenza implica inoltre un Corollario, sotto menzionato, che caratterizza la convergenza del FastSLAM Lineare Gaussiano a più di una particella:

Corollario 1 *Il FastSLAM converge nei valori attesi allo SLAM-LG se tutte le*

feature *sono osservate infinitamente e la posizione di una* feature *è conosciuta in anticipo.*

3.3.4 Chiusura dei *loop*

Nel FastSLAM la capacità di chiudere i *loop* è legata al numero di particelle M; il numero minimo di particelle necessarie a chiudere il *loop* (e quindi il numero minimo di particelle necessarie per ottenere una mappa accurata) è difficile da quantificare, e dipende da molti fattori, ad esempio dai parametri dei modelli di movimento e di misura e dalla densità dei landmark nella mappa. In generale una migliore diversità nel *sample set* garantisce una migliore accuratezza nella costruzione della mappa, poiché le nuove osservazioni possono influire sulla posizione del robot in maniera più consistente.

Con i miglioramenti descritti in questa sezione il numero di particelle necessarie per chiudere il *loop* è inferiore rispetto a quello necessario dal FastSLAM originale.

3.3.5 Il FastSLAM in ambienti dinamici

Per l'algoritmo finora descritto si è assunto che il robot navigasse in ambienti statici; nel mondo reale, questa è un'assunzione piuttosto restrittiva se si considera l'esplorazione in ambienti *indoor* o comunque in ambienti in cui sono presenti ostacoli mobili. Il FastSLAM permette di raggiungere comunque un livello di accuratezza molto buono anche in ambienti dinamici. Infatti la natura particellare e probabilistica dell'algoritmo permette di escludere gli ostacoli mobili, utilizzando le osservazioni ed il *resampling* delle particelle. Ad esempio se si considera un ostacolo mobile che attraversa il range dei sensori del robot, questo può essere riconosciuto come una *feature* e quindi in un primo momento, l'associazione di dati può associarlo ad un landmark; successivamente, il filtro escluderà questo eventuale landmark, se l'associazione di dati lo riterrà opportuno (ad esempio se la distanza di Mahalanobis è maggiore della soglia per l'associazione ad una

feature esistente oppure se è minore della soglia per l'aggiunta di una nuova *feature*). Inoltre il passo di *resampling* delle particelle contribuisce all'esclusione delle *feature* "poco probabili", dando loro poco peso, oppure eliminandole dal filtro. Il FastSLAM, quindi, gode di un certo grado di robustezza rispetto agli ostacoli dinamici.

Capitolo 4

Implementazione *FastSLAM*

In questo capitolo verrà descritta la tecnica di implementazione del FastSLAM, descritto precedentemente. In primo luogo verrà mostrata in dettaglio la struttura dell'ambiente simulativo in *Matlab*, successivamente verrà descritta l'implementazione dell'algoritmo in linguaggio *C* sul robot mobile *XR4000* della famiglia *Nomad*. Infine verranno mostrati risultati sperimentali effettuati nel Laboratorio di Automatica presso l'Università di Roma "Tor vergata".

4.1 Implementazione su *Matlab*

La prima parte della progettazione del FastSLAM è stata fatta sfruttando l'ambiente di programmazione ad alto livello fornito da *Matlab*. In questa prima fase si è organizzato l'algoritmo in diverse funzioni, per dare una struttura modulare al progetto:

- `inizialize_particles`;
- `predict_true`;
- `add_control_noise`;
- `predict`;

- `get_observation`;
- `compute_jacobians`;
- `data_association_unknown`;
- `compute_weight`;
- `resample_particles`;
- `sample_proposal`;
- `feature_update`;
- `multivariate_gauss`;
- `add_feature`;
- `create_map`;
- `adjust_map`.

In questa prima fase il robot è assunto puntiforme, ma questa ipotesi non è semplificativa, in quanto interessa soltanto le routine di *obstacle avoidance* e di navigazione, che sono state implementate direttamente sul robot, tenendo in considerazione le dimensioni reali. I sensori[1] presenti sul robot, laser *Sick S500* e sensori odometrici, sono stati modellizzati considerando che sono affetti da rumore di tipo Guassiano a media nulla (ipotesi realistiche); inoltre il moto del robot è stato modellizzato come un moto uniformemente accelerato. Le traiettorie fatte eseguire al robot virtuale sono state differenti, così come le mappe virtuali, nelle quali il robot navigava sono state scelte tra più o meno complicate: in particolare si è creata un'interfaccia per rendere più immediata la costruzione delle mappe, in figura 4.1 è riportato uno *screenshot* della *GUI*. La simulazione in ambiente *Matlab* ha portato a risultati soddisfacenti ed ha giocato un ruolo fondamentale nella successiva implementazione in *C* sul robot. Infatti i parametri

[1]Per una descrizione dettagliata dei sensori si rimanda alla sezione 4.3.1 e al manuale [17].

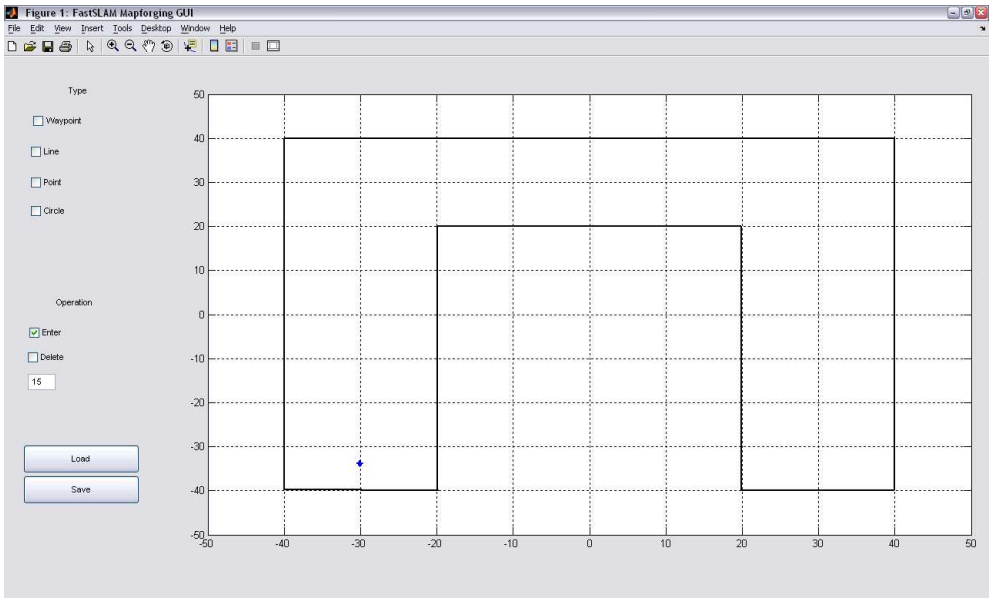

Figura 4.1: *Interfaccia per la costruzione delle mappe.*

dell'algoritmo, quali soglie per l'associazione di dati, il numero di particelle, ed i parametri per il *resampling*, messi a punto durante la fase simulativa, hanno prodotto buoni risultati anche nella successiva implementazione sul robot.

4.1.1 Struttura dell'ambiente simulativo

In questa sezione verrà illustrato il *layout* dell'ambiente simulativo implementato in *Matlab*. Verranno illustrate singolarmente le funzioni principali sviluppate, che rappresentano l'algoritmo FastSLAM.

initialize_particles

Come primo passo è necessario inizializzare le particelle presenti nel filtro, con la funzione `initialize_particles`; i pesi delle M particelle vengono posti al valore $\frac{1}{M}$, la posizione iniziale di ogni singola particella è posta a $\{0, 0, 0\}$, la covarianza P_v è pari ad una matrice 3×3 di zeri, ed infine si impone che ogni particella non abbia alcuna *feature* presente, all'inizio della simulazione.

predict_true

In questa funzione viene generata la posizione reale del robot al passo dt; la legge di moto utilizzata per generare questa posizione è la seguente:

$$s_{v,1,r,t} = s_{v,1,r,t-1} + (v\,dt + \frac{1}{2}\,a\,dt^2)\,\cos(\omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 + s_{v,3,r,t-1}) \quad (4.1)$$

$$s_{v,2,r,t} = s_{v,2,r,t-1} + (v\,dt + \frac{1}{2}\,a\,dt^2)\,\sin(\omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 + s_{v,3,r,t-1}) \quad (4.2)$$

$$s_{v,3,r,t} = s_{v,3,r,t-1} + \omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 \quad (4.3)$$

dove v rappresenta la velocità traslazionale (supposta costante nell'istante dt), a è l'accelerazione traslazionale, ω è la velocità rotazionale (anch'essa supposta costante nell'istante dt), $\dot{\omega}$ è l'accelerazione rotazionale, ed infine $s_{v,1,r,t}$, $s_{v,2,r,t}$ rappresentano le coordinate reali del robot al tempo t, lungo l'asse x, y e dove $s_{v,3,r,t}$ rappresenta l'*heading* reale.

add_control_noise

Dopo aver generato l'ingresso di controllo, e quindi la traiettoria di posizione e velocità che il robot doveva eseguire, l'ingresso di controllo nominale è stato "sporcato" con del rumore Gaussiano a media nulla; lo scopo è stato quello di rendere più realistica la simulazione.

predict

La predizione della posizione del robot al tempo t, nota la posizione al tempo $t-1$ è effettuata dalla funzione `predict`. Utilizzando le (4.1), (4.2), (4.3), è stato possibile ricavare gli Jacobiani di posizione J_v e di controllo J_u, necessari

a calcolare la matrice di covarianza P_v:

$$J_v = \begin{bmatrix} 1 & 0 & -(v\,dt + \frac{1}{2}\,a\,dt^2)\,\sin(\delta_G + s_{v,3,t-1}) \\ 0 & 1 & (v\,dt + \frac{1}{2}\,a\,dt^2)\,\cos(\delta_G + s_{v,3,t-1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$J_u = \begin{bmatrix} dt\,\cos(\delta_G + s_{v,3,t-1}) & -(v\,dt + \frac{1}{2}\,a\,dt^2)\,dt\,\sin(\delta_G + s_{v,3,t-1}) \\ dt\,\sin(\delta_G + s_{v,3,t-1}) & (v\,dt + \frac{1}{2}\,a\,dt^2)\,dt\,\cos(\delta_G + s_{v,3,t-1}) \\ 0 & dt \end{bmatrix} \tag{4.5}$$

dove:

$$\delta_G = \omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2$$

La matrice di covarianza P_v è quindi calcolata come:

$$P_{v,t} = J_v\,P_{v,t-1}\,J_v^T + J_u\,Q\,J_u^T \tag{4.6}$$

dove Q è la matrice di covarianza del modello di movimento. Infine la posizione predetta al tempo t è calcolata come segue:

$$s_{v,1,t} = s_{v,1,t-1} + (v\,dt + \frac{1}{2}\,a\,dt^2)\,\cos(\omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 + s_{v,3,t-1}) \tag{4.7}$$

$$s_{v,2,t} = s_{v,2,t-1} + (v\,dt + \frac{1}{2}\,a\,dt^2)\,\sin(\omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 + s_{v,3,t-1}) \tag{4.8}$$

$$s_{v,3,t} = s_{v,3,t-1} + \omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 \tag{4.9}$$

È opportuno notare che la matrice di covarianza P_v deve essere azzerata dopo ogni osservazione, in quanto accumula l'incertezza del robot tra una misura e l'altra.

get_observation

Con questa funzione si simula il riconoscimento delle *feature* all'interno del campo visivo del robot, descritto dall'equazione:

$$(x - s_{r,1})^2 + (y - s_{r,2})^2 - \rho^2 = 0 \tag{4.10}$$

dove $s_{r,1}$ e $s_{r,2}$ sono le coordinate reali del robot e ρ è il range del campo visivo, parametro variabile e scelto pari al range reale del laser del robot.

In simulazione vengono generate mappe in cui il robot virtuale può navigare; queste mappe vengono create disegnando tramite un editor rette, cerchi e punti, oggetti che poi sono discretizzati in punti, in modo tale da rendere discrete le osservazioni. In seguito vengono esclusi i punti che non appartengono al campo visivo, poi questo set di punti viene passato ad un algoritmo di *z-buffering*, che elimina i punti che non possono essere visti dal robot, perché coperti ed infine queste osservazioni discrete vengono "sporcate" con un rumore Gaussiano utilizzando la matrice di covarianza del rumore di osservazione. In questo modo ad ogni passo si ottiene un set discreto di punti che, con buona approssimazione, può essere considerato come una scansione del laser presente sul robot.

Nella seconda fase della *routine* `get_observation`, si sono studiati due algoritmi di estrazione di *feature*, e se ne è verificata l'efficacia con test simulativi ripetuti. I due algoritmi in questione sono:

- estrazione dei segmenti;
- estrazione degli angoli IPAN99.

Estrazione dei segmenti Il modello di osservazione del robot restituisce la misura relativa tra esso e l'ambiente circostante. Per l'estrazione di segmenti un metodo molto utilizzato è la trasformata di Hough o la trasformata di Radon, che hanno la stessa complessità computazionale. In *Matlab* si è implementato un nuovo metodo basato su quello di Radon; questo consiste nel far passare per tutti i punti osservati nel range visivo delle rette, che variano il loro orientamento $\alpha \in [-\pi, \pi]$. Vengono quindi contati il numero di punti che appartengono a tali rette. Da queste informazioni si costruisce un istogramma in cui sulle ascisse sono presenti i gradi della pendenza delle rette e sulle ordinate il numero di punti incontrati da esse. Questo istogramma viene automaticamente analizzato, utilizzando una soglia oltre la quale vengono prese le rette candidate ad essere segmenti. L'operazione di sogliatura è utilizzata principalmente per filtrare

misure spurie e il carattestico rumore gussiano di misura. Delle rette candidate vengono quindi trovati gli estremi, definendo così i segmenti. Processi successivi definiscono i punti medi e l'angolazione relativa rispetto al robot. Si è prestata particolare attenzione alle rette che hanno punti in comune, come gli angoli, per fare in modo che il metodo li individuasse sempre, essendo questi delle *feature* di particolare interesse per il FastSLAM.

Utilizzare una soglia per selezionare le rette candidate è una procedura particolarmente delicata. Accade spesso che sopra questa soglia ci siano molte rette candidate che hanno pendenza molto simile per il passaggio di punti molto vicini. È evidente che una situazione del genere indica in realtà che tale retta passa per tutti quei punti. È opportuna, quindi, una procedura di media che trovi questi casi e li unifichi. Questa è un'operazione complessa e computazionalmente onerosa, perché è necessario verificarla, ed eventualmente applicarla, per tutte le rette candidate. Può accadere, in condizioni particolari, quali ad esempio quelle in cui il robot è vicino a un muro o rileva un contorno particolarmente frastagliato, che questo processo di media sia un lavoro ancor più gravoso; per questo motivo si è modificato l'algoritmo in modo tale che se si è in condizioni in cui sono state rilevate moltissime rette simili sopra la soglia, tale soglia viene modificata alzandola e l'algoritmo riavviato. Questa modalità dinamica di assegnazione della soglia ha portato notevoli miglioramenti nei casi critici, portando a un tempo di elaborazione quasi omogeneo per tutte le condizioni simulative nei vari ambienti.

Estrazione degli angoli IPAN99 Viene qui esposto un algoritmo molto innovativo e performante per il riconoscimento di angoli. Una lettura di dati del sensore laser restituisce una sequenza, cosidetta catena, di punti descritta dalla coppia distanza e orientamento. Sulle catene di punti l'algoritmo IPAN99 sviluppato da Dmitry Chetverikov e Zsolt Szabó [2] si comporta bene; il principale vantaggio rispetto agli altri algoritmi di *corner detection*, escludendo la maggiore velocità, è che è in grado di funzionare su catene di punti non equispaziate, mentre per altri algoritmi come quelli proposti da Rosenfeld-Johnston [18], Rosenfeld-Weszka [19], Freeman-Davis [6] e Beus-Tiu [1] è obbligatorio. Test sperimentali

effettuati dimostrano che l'IPAN99 è del 50% più veloce del Rosenfeld-Johnston e il vantaggio aumenta all'aumentare della complessità della forma da riconoscere.

Gli angoli, o meno banalmente punti ad alta curvatura, sono individuati tramite un algoritmo che consta di due passaggi successivi. Questo algoritmo definisce un angolo in un modo semplice e intuitivo, cioè come un punto in cui può essere inscritto un triangolo. Una curva può essere rappresentata da una sequenza di punti p_i. I punti ordinati sono campionati densamente sulla curva ma può non esistere una spaziatura regolare tra essi.

Il primo passo dell'algoritmo analizza la sequenza e propone i candidati tra i punti che definiscono gli angoli. Il secondo passaggio, invece, consta di un post-processamento per eliminare i candidati superflui.

Primo passo: L'analizzatore, o *detector*, prende ogni punto della curva p e prova a inscrivere la curva in un triangolo variabile definito dai tre punti (p_i^-, p_i, p_i^+). Questi tre punti devono soddisfare le seguenti condizioni:

$$d_{min} \leqslant |p_i - p_i^+| \leqslant d_{max} \tag{4.11}$$

$$d_{min} \leqslant |p_i - p_i^-| \leqslant d_{max} \tag{4.12}$$

$$\alpha \leqslant \alpha_{max} \tag{4.13}$$

Dove d_{min}, d_{max}^2 e α_{max} sono costanti fissate a priori e rappresentano i parametri dell'algoritmo, p_i^- e p_i^+ sono, rispettivamente, il punto precedente e il punto successivo al punto in esame p_i. I punti p_i^- e p_i^+ possono essere non strettamente il precedente o il successivo ma possono essere in realtà ad eguale distanza fissata, quindi è possibile settare p_i^- e p_i^+ distanti entrambi tre punti da p_i. Per comodità p_i è definito in coordinate cartesiane tramite il vettore $p_i = \langle x_i, y_i \rangle$, il modulo $|p_i - p_i^+|$ della (4.12) definisce la distanza a tra p_i e p_i^+, il modulo $|p_i - p_i^-|$ della (4.13) definisce la distanza b tra p_i e p_i^-, l'angolo α è proprio l'angolo di apertura del triangolo, si veda per chiarezza la figura 4.2. Il valore di α proviene direttamente dal teorema di Carnot:

$$\alpha \overset{Carnot}{=} \arccos\left(\frac{a^2 + b^2 - c^2}{2ab}\right) \tag{4.14}$$

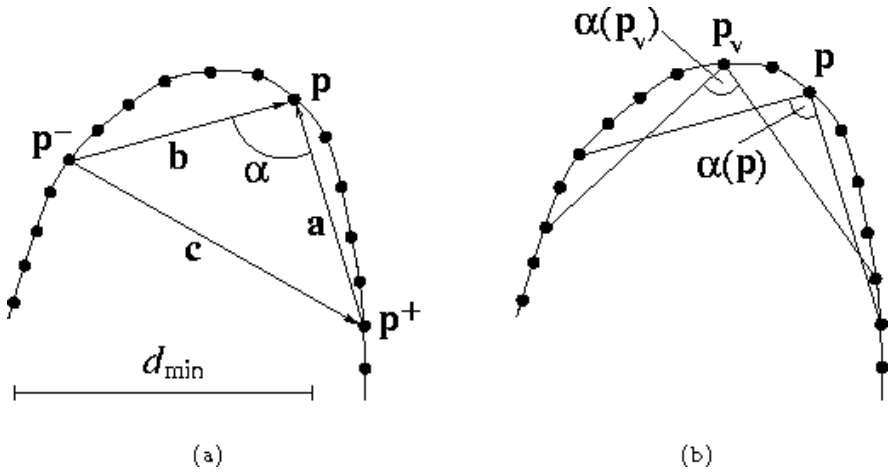

Figura 4.2: *Rappresentazione geometrica del triangolo inscritto nella curva* p.

Tutte quelle variazioni del triangolo che soddisfano le tre condizioni (4.12) sono chiamate ammissibili. Il detector ricerca p_i per tutta la curva. Oltretutto se si considerano i due vettori definiti da $b = (b_x, b_y)$ e da $c = (c_x, c_y)$ è facile distinguere se l'angolo è convesso o concavo. Basta verificare il segno di $b_x c_y - b_y c_x$: se è $\geqslant 0$ allora è convesso altrimenti è concavo. $\pi - |\alpha p_i|$ è definito come lo *sharpness* dell'angolo.

Secondo passo: In questo secondo passaggio vengono scartati tutti i punti candidati con un criterio di massimo *sharpness*, questo perchè il *detector* può considerare punti candidati più punti associati a uno stesso angolo. Un punto candidato p viene scartato se un suo vicino p_v valido, cioè un punto candidato che soddisfa $|p - p_v| \leq d_{max}$, ha uno *sharpness* maggiore cioè un $p_v : \alpha(p) \geq \alpha(p_v)$.

L'algoritmo IPAN99, mostrando qualità migliori rispetto all'algoritmo di estrazione dei segmenti nonché una velocità computazionale maggiore, è stato scelto ed implementato in linguaggio *C* successivamente sul robot.

compute_jacobians

Questa funzione effettua, conoscendo la posizione di una particella del robot e la posizione di un landmark selezionato, il calcolo dell'osservazione predetta, dello Jacobiano relativo alla distanza della *feature* selezionata rispetto agli stati del robot, dello Jacobiano relativo alla distanza della *feature* selezionata rispetto agli stati del landmark, e della covarianza dell'osservazione della *feature* condizionata alla posizione del robot. Si ricorda che la posizione del robot è definita da $s_v = \langle s_{v,x}, s_{v,y}, s_{v,\alpha} \rangle$ e la posizione del landmark selezionato da $\theta_n = \langle \theta_{n,x}, \theta_{n,y} \rangle$. Il vettore z_p rappresenta la distanza euclidea tra la posizione del robot s_v e la posizione del landmark θ_n selezionato.

Si può definire:

$$z = h(s_v, \theta_n) = \begin{bmatrix} \sqrt{(s_{v,x} - \theta_{n,x})^2 + (s_{v,y} - \theta_{n,y})^2} \\ \text{atan}(s_{v,y} - \theta_{n,y}, s_{v,x} - \theta_{n,x}) \end{bmatrix} \tag{4.15}$$

da cui si calcola lo Jacobiano H_v relativo alla distanza della *feature* selezionata rispetto agli stati del robot:

$$H_v = \begin{bmatrix} \frac{\partial h_1}{\partial s_v} \\ \frac{\partial h_2}{\partial s_v} \end{bmatrix} \tag{4.16}$$

$$= \begin{bmatrix} \frac{\partial h_1}{\partial s_{v,x}} & \frac{\partial h_1}{\partial s_{v,y}} & \frac{\partial h_1}{\partial s_{v,\alpha}} \\ \frac{\partial h_2}{\partial s_{v,x}} & \frac{\partial h_2}{\partial s_{v,y}} & \frac{\partial h_2}{\partial s_{v,\alpha}} \end{bmatrix} \tag{4.17}$$

$$= \begin{bmatrix} \frac{s_{v,x} - \theta_{n,x}}{\sqrt{(s_{v,x} - \theta_{n,x})^2 + (s_{v,y} - \theta_{n,y})^2}} & \frac{s_{v,y} - \theta_{n,y}}{\sqrt{(s_{v,x} - \theta_{n,x})^2 + (s_{v,y} - \theta_{n,y})^2}} & 0 \\ \frac{-(s_{v,y} - \theta_{n,y})}{(s_{v,x} - \theta_{n,x})^2 + (s_{v,y} - \theta_{n,y})^2} & \frac{s_{v,x} - \theta_{n,x}}{(s_{v,x} - \theta_{n,x})^2 + (s_{v,y} - \theta_{n,y})^2} & -1 \end{bmatrix} \tag{4.18}$$

Analogamente si calcola lo Jacobiano H_f relativo alla distanza della *feature*

selezionata rispetto agli stati del landmark:

$$H_f = \begin{bmatrix} \frac{\partial h_1}{\partial \theta_n} \\ \frac{\partial h_2}{\partial \theta_n} \end{bmatrix} \tag{4.19}$$

$$= \begin{bmatrix} \frac{\partial h_1}{\partial \theta_{n,x}} & \frac{\partial h_1}{\partial \theta_{n,y}} \\ \frac{\partial h_2}{\partial \theta_{n,x}} & \frac{\partial h_2}{\partial \theta_{n,y}} \end{bmatrix} \tag{4.20}$$

$$= \begin{bmatrix} \frac{-(s_{v,x}-\theta_{n,x})}{\sqrt{(s_{v,x}-\theta_{n,x})^2+(s_{v,y}-\theta_{n,y})^2}} & \frac{-(s_{v,y}-\theta_{n,y})}{\sqrt{(s_{v,x}-\theta_{n,x})^2+(s_{v,y}-\theta_{n,y})^2}} & 0 \\ \frac{s_{v,y}-\theta_{n,y}}{(s_{v,x}-\theta_{n,x})^2+(s_{v,y}-\theta_{n,y})^2} & \frac{-(s_{v,x}-\theta_{n,x})}{(s_{v,x}-\theta_{n,x})^2+(s_{v,y}-\theta_{n,y})^2} & -1 \end{bmatrix} \tag{4.21}$$

Il calcolo della covarianza dell'osservazione della *feature* $n-esima$ condizionata alla posizione del robot S_f, è effettuata tramite la seguente:

$$Sf = H_f \cdot P_f \cdot H_f^T + R \tag{4.22}$$

dove P_f è la matrice di covarianza della posizione del landmark $n-esimo$ ed R la matrice di covarianza dell'errore di misura.

data_association_unknown

In questa funzione è stata implementata l'associazione dei dati sconosciuta con criterio a massima verosimiglianza o *nearest-neighbour*. Inizialmente si è implementata l'associazione di dati con un criterio di ricerca *nearest-neighbour* lineare, la complessità di questa funzione risulta così $O(N)$; successivamente si è implementata una versione semplificata in cui la ricerca viene effettuata solo sulle *feature* più vicine, scartando le *feature* distanti dalla *feature* correntemente analizzata (metodo a mappe locali).

La ricerca delle *feature* vicine a quella corrente viene fatta utilizzando un criterio di massima verosimiglianza, ossia: vengono calcolate due distanze chiamate *nis* (*normalized innovation squared*, cioé la distanza di Mahalanobis) e *nd* (*normalized distance*) che sono definite come segue:

$$nis = v^T S_f^{-1} v \tag{4.23}$$

$$nd = nis + \log(\det(S_f)) \tag{4.24}$$

dove:

$$v = z - z_p \tag{4.25}$$

e dove S_f, z_p sono calcolati dalla funzione `compute_jacobians`.

Dopo aver ricavato le distanze relative ad ogni *feature* rispetto alla *feature* corrente, occorre testare se queste sono comprese entro delle determinate soglie: in caso affermativo viene registrato l'indice della *feature* "vicina" alla *feature* corrente, il che indica che questa, con buona probabilità è la stessa *feature*. Se invece le distanze non soddisfacessero la condizione precedente, allora per la *feature* sarebbero possibili due ipotesi: in un caso potrebbe essere molto distante dalla *feature* corrente, ma non abbastanza distante dal poter essere associata ad una nuova *feature*, oppure potrebbe essere abbastanza distante da poter essere invece associata ad una nuova *feature*. In questo ultimo caso la funzione genera una nuova *feature* aggiornando la tabella di associazione dei dati.

compute_weight

Il calcolo dei pesi di ogni singola particella viene effettuato calcolando prima gli Jacobiani H_v, S_f e z_p come riportato precedentemente e successivamente calcolando la seguente quantità, nota come covarianza d'innovazione, che include l'incertezza della posizione del robot e delle *feature*:

$$S = H_v \, P_v \, H_v^T + S_f \tag{4.26}$$

infine il peso di ogni particella viene calcolato come:

$$w = w_{old} \cdot p(z_t | s_{t-1}) \tag{4.27}$$

dove:

$$p(z_t | s_{t-1}) = \frac{e^{-\frac{1}{2} v^T S^{-1} v}}{2 \pi \sqrt{\det(S)}} \tag{4.28}$$

resample_particles

Il passo di *resample* descritto nella sezione 3.2.5 è eseguito in questa funzione. Se la varianza dei pesi delle particelle è tale che N è minore di una soglia fissata,

allora viene effettuato il *resampling*. Valori maggiori della soglia implicano *resampling* più frequenti, mentre valori minori implicano *resampling* meno frequenti. Sperimentalmente si è verificato che un tasso di *resampling* basso permette una maggiore diversità nelle particelle, ma una velocità di convergenza più bassa, mentre un alto tasso di *resampling* implica una minore diversità nelle particelle che è in generale da evitare, come riportato in [9].

In particolare il *resampling* è eseguito prima di effettuare il *sample proposal* in modo tale da garantire una migliore diversità tra le particelle; inoltre è necessario tener presente che ogni particella può osservare un numero di *feature* diverse rispetto alle altre, è quindi opportuno prestare particolare attenzione quando si effettua il *resampling* a non confondere le *feature* di diverse particelle. In questa funzione si è tenuto conto anche di questo aspetto.

sample_proposal

Questa funzione è addetta al calcolo della *proposal distribution* e al campionamento da essa. Per ogni *feature* vengono calcolate le quantità z_p, H_v e Q_f come riportato nella funzione `compute_jacobians`.

Utilizzando le matrici in (4.18), (4.21) e (4.22), si possono calcolare:

$$P_{v,t} = \left(H_v^T\, Q_f^{-1}\, H_v + P_{v,t-1}^{-1}\right) \tag{4.29}$$

$$s_{v,t} = s_{v,t-1} + P_v\, H_v^{-1}\, Q_f^{-1}\, v \tag{4.30}$$

con:

$$v = z - z_p$$

dove $P_{v,t}$ è la *proposal covariance* e dove $s_{v,t}$ è la *proposal mean*.

Dopo aver calcolato la *proposal distribution* occorre prelevare dei campioni da essa. Questo passo è effettuato calcolando per la posizione del robot $s_{v,t}$ una distribuzione di probabilità multivariata di Gauss[2] con media $s_{v,t}$ e covarianza $P_{v,t}$ e azzerando la covarianza della particella.

[2]Il calcolo della distribuzione di probabilità multivariata di Gauss è riportato più avanti nel testo.

feature_update

Dopo aver selezionato una nuova posizione dalla *proposal distribution*, questa viene assunta essere perfetta, ed ogni aggiornamento delle *feature* può essere calcolato in maniera indipendente e senza incertezza sulla posizione.

In particolare vengono calcolate con la funzione `compute_jacobians` le grandezze z_p, H_f ed S_f, che poi vengono utilizzate dal filtro di Kalman per calcolare l'aggiornamento della posizione e della covarianza delle *feature*. Il filtro di Kalman esteso è calcolato utilizzando la fattorizzazione di Cholesky[3], che è numericamente più stabile rispetto all'implementazione standard del filtro. In particolare dallo stato $[s_{f,t-1}, P_{f,t-1}]$, l'innovazione $[v, R]$ ed il modello di osservazione linearizzato h, viene calcolata l'innovazione $[s_{f,t}, P_{f,t}]$. In primo luogo viene calcolata S:

$$S = H\, P_{f,t-1}\, H^T + R \tag{4.31}$$

Successivamente S viene resa simmetrica:

$$S = \frac{1}{2}\,(S + S^T) \tag{4.32}$$

Poi si procede al calcolo della fattorizzazione di Cholesky di S, chiamata $S_{cholesky}$, che viene infine utilizzata per il calcolo dell'aggiornamento della posizione e della matrice di covarianza:

$$W = P_{f,t-1}\, H^T\, S^{-1}_{cholesky}\, S^{-1^T}_{cholesky} \tag{4.33}$$

$$s_{f,t} = s_{f,t-1} + W\, v \tag{4.34}$$

$$P_{f,t} = P_{f,t-1} - P_{f,t-1}\, H^T\, S^{-1}_{cholesky}\, S^{-1^T}_{cholesky}\, H\, P^T_{f,t-1} \tag{4.35}$$

dove H è il modello di osservazione.

[3]Per una trattazione approfondita sulla decomposizione di Cholesky si rimanda a [8] e a [15].

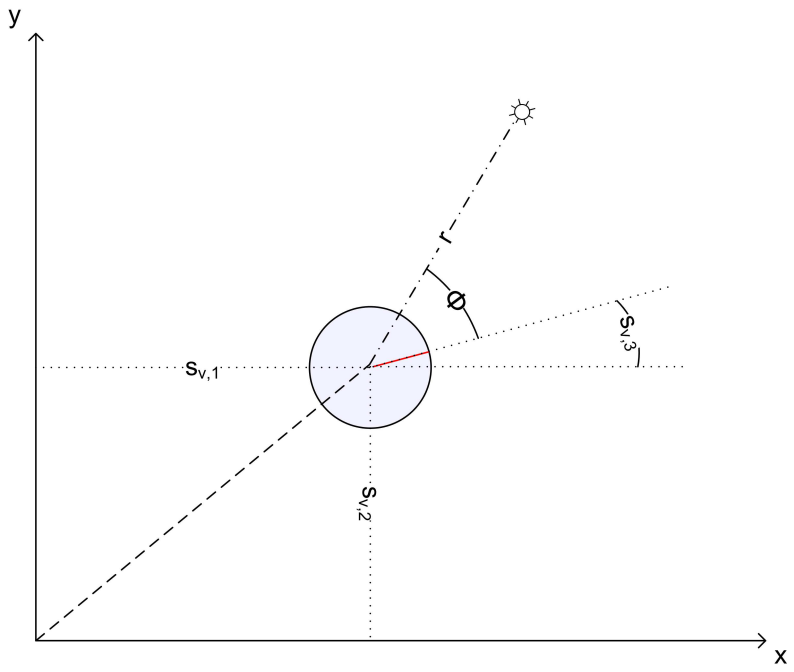

Figura 4.3: *Robot e landmark nel riferimento assoluto.*

multivariate_gauss

Questa funzione restituisce un set di campioni (*sample set*) da un vettore di posizioni medie s_v e da una matrice di covarianza P_v. Il set di campioni è scelto casualmente da una distribuzione multivariata Gaussiana.

Dopo aver scelto il set di campioni, la matrice di covarianza P_v viene resettata a zero, per non accumulare l'errore sulla posizione del robot.

add_feature

Se l'algoritmo di associazione di dati sconosciuta riconosce nell'ambiente la presenza di una o più nuove *feature*, queste devono essere incorporate nel filtro. La funzione addetta a tale scopo è `add_feature`, che calcola la posizione e la covarianza delle nuove *feature*. La posizione della nuova *feature* è data dalla

seguente:

$$s_f = \begin{bmatrix} s_{v,1} + r\ \cos(s_{v,3} + \phi) \\ s_{v,2} + r\ \sin(s_{v,3} + \phi) \end{bmatrix} \tag{4.36}$$

la covarianza P_f è invece calcolata come segue:

$$G_z = \begin{bmatrix} \cos(s_{v,3} + \phi) & -r\ \sin(s_{v,3} + \phi) \\ \sin(s_{v,3} + \phi) & r\ \cos(s_{v,3} + \phi) \end{bmatrix} \tag{4.37}$$

$$P_f = G_z\, R\, G_z^T \tag{4.38}$$

create_map

Questa funzione è stata creata per generare la mappa dell'ambiente virtuale in cui si trova il robot; sostanzialmente la mappa viene generata utilizzando come posizione del robot, quella della particella con peso maggiore, che servirà per riportare le osservazioni nel riferimento assoluto. Con `create_map` si sono ottenuti buoni risultati, che sono stati migliorati con la seguente funzione.

adjust_map

Dopo aver generato la mappa semplicemente utilizzando la posizione della particella di peso maggiore, si è proceduto alla scrittura di una funzione che tenesse conto degli aggiustamenti successivi, effettuati in base alla riosservazione delle *feature* nell'ambiente, in modo tale da aggiustare anche la mappa agganciandola alle *feature* estratte dall'algoritmo di osservazione dell'ambiente. Il risultato è stata una mappa che gode di una buona continuità e di una buona risoluzione.

4.1.2 Risultati simulativi

In questa sezione verranno esposti i risultati ottenuti dalle simulazioni effettuate in Matlab.

In primo luogo si è proceduto alla creazione della mappa ed alla successiva discretizzazione, di cui un esempio è riportato in figura 4.4. I parametri utilizzati

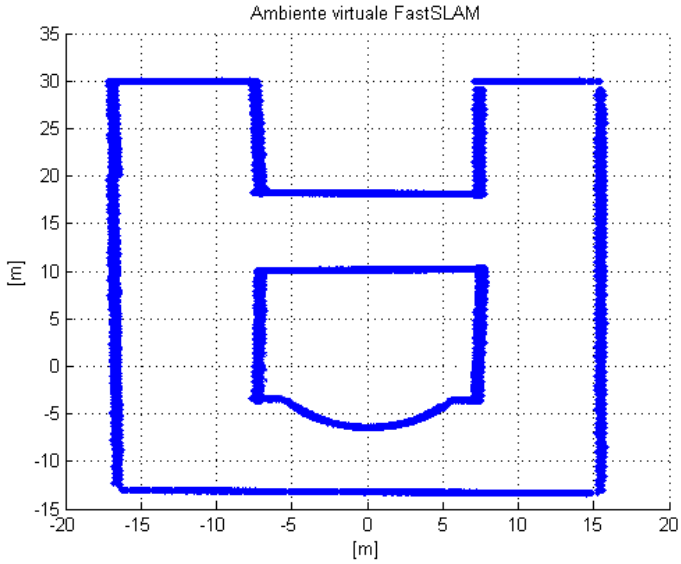

Figura 4.4: *Mappa dell'ambiente virtuale discretizzata.*

nella simulazione sono quelli riportati in tabella 4.1. Questi sono stati modificati

σ_V	0.1
σ_Ω	$3.0\,\frac{\pi}{180}$
σ_R	0.1
σ_B	$1.0\,\frac{\pi}{180}$
$N_{particles}$	100
$N_{effective}$	$0.9\,N_{particles}$
$gate\_reject$	9
$gate\_augment$	5000

Tabella 4.1: Parametri standard utilizzati per la simulazione del FastSLAM.

per studiare il comportamento del FastSLAM al variare dei parametri.

In primo luogo si è modificato il parametro σ_Ω che rappresenta la varianza relativa alla velocità angolare del robot; in particolare questo parametro è stato aumentato al valore $\sigma_\Omega = 6.0\,\frac{\pi}{180}$. Il risultato della simulazione con questi parametri è riportato in figura 4.5. Nella figura è stata riportata con il colore rosso

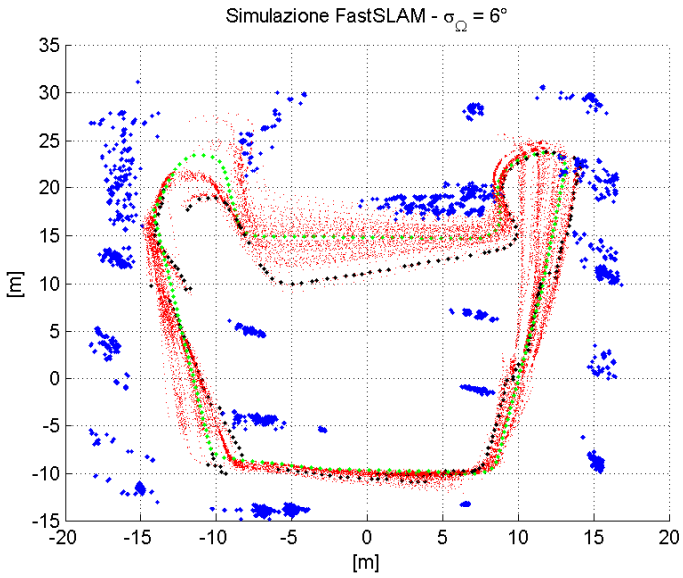

Figura 4.5: *Simulazione con parametri* $\sigma_\Omega = 6.0\,\frac{\pi}{180}$.

la distribuzione delle particelle presenti nel filtro, con il colore verde la traiettoria reale del robot, con il colore nero la posizione della particella di peso maggiore e con il colore blu la distribuzione delle *feature* nell'ambiente.

Come si può osservare, il risultato, soprattutto nella parte centrale della mappa, non è molto accurato, e la traiettoria della particella di peso maggiore si discosta sensibilmente dalla traiettoria reale del robot. In ogni caso, come è facile osservare, il filtro riesce a correggere la posizione, non appena il robot ritorna vicino alla posizione iniziale e non appena riosserva le prime *feature* alle quali era associata una bassa incertezza.

Successivamente, ripristinando i parametri ai valori riportati in tabella 4.1 e diminuendo il parametro $N_{effective}$ al valore $0.1\,N_{particles}$, il risultato della simulazione è riportato in figura 4.6. Come si può osservare, le distribuzioni delle particelle e delle posizioni dei landmark risultano abbastanza sparse, all'aumentare dello spazio percorso dal robot: questo è dovuto appunto da uno scarso tasso di *resampling*. Il risultato è che prima della riosservazione delle *feature* iniziali la distribuzione delle particelle è visibilmente sparsa, e quindi si è accumulato

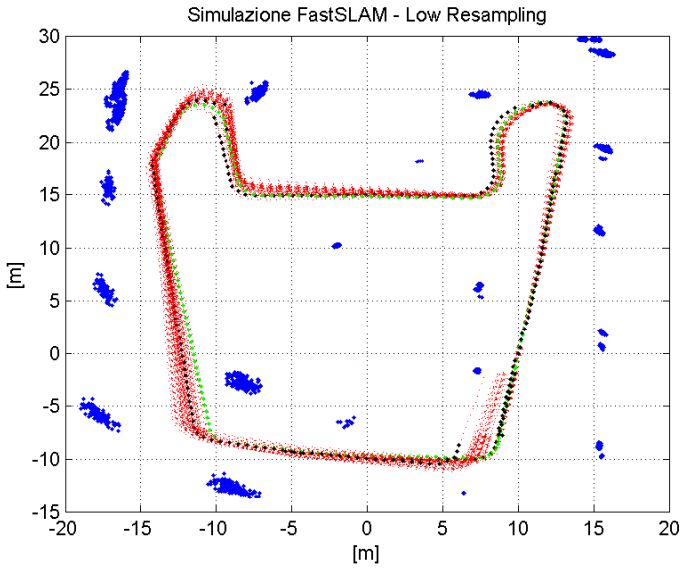

Figura 4.6: *Simulazione con parametri* $N_{effective} = 0.1\, N_{particles}$.

un ampio margine di incertezza, che, dopo la riosservazione delle prime *feature*, viene azzerato e la posizione del robot stimata con buona approssimazione.

Utilizzando invece un alto tasso di *resampling*, dell'ordine di $0.9\, N_{particles}$, si ottiene il risultato riportato in figura 4.7. Da questa simulazione è facile osservare come la traiettoria reale del robot si scosta di poco da quella della particella di peso maggiore in ogni punto della mappa, e come di conseguenza il loop viene chiuso. L'effetto dell'alto tasso di *resampling* si ripercuote molto sulla distribuzione delle particelle, che risulta sempre spazialmente contenuta.

Ripristinando i parametri ai valori riportati in tabella 4.1 e modificando il parametro $gate\_reject$ al valore 100, si ottiene il risultato in figura 4.8. Il parametro $gate\_reject$ rappresenta il valore massimo della distanza di Mahalanobis relativa ad una coppia di *feature* entro il quale una *feature* osservata viene associata ad una già inglobata nel filtro. Aumentare questo parametro significa aumentare la probabilità di incorporare una *feature* abbastanza distante da una già presente nel filtro, con il rischio di effettuare quindi una associazione di dati sbagliata. Nel caso in esame è facile osservare inoltre che le particelle riman-

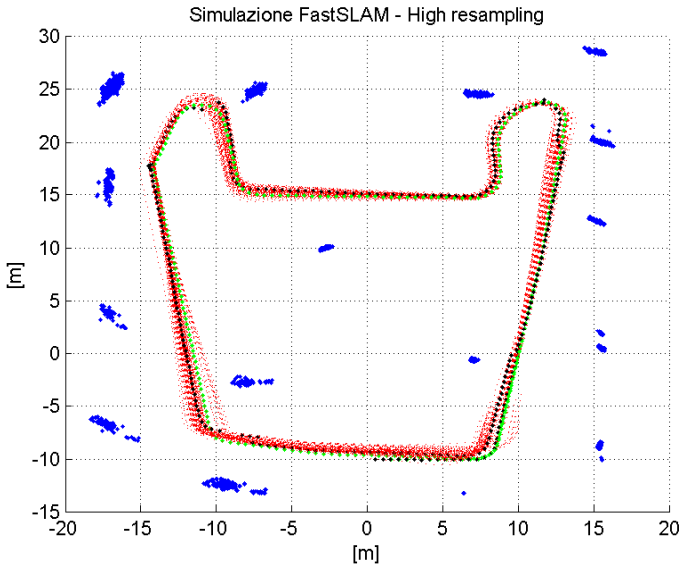

Figura 4.7: *Simulazione con parametri* $N_{effective} = 0.9\, N_{particles}$.

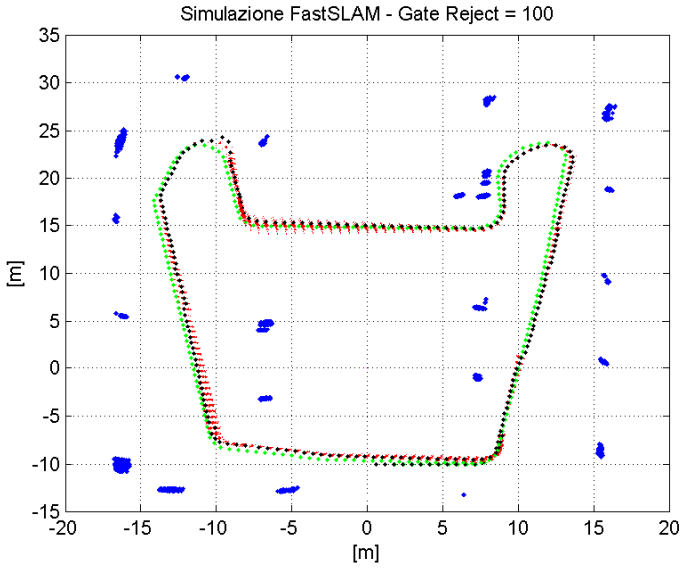

Figura 4.8: *Simulazione con parametri* $gate\_reject = 100$.

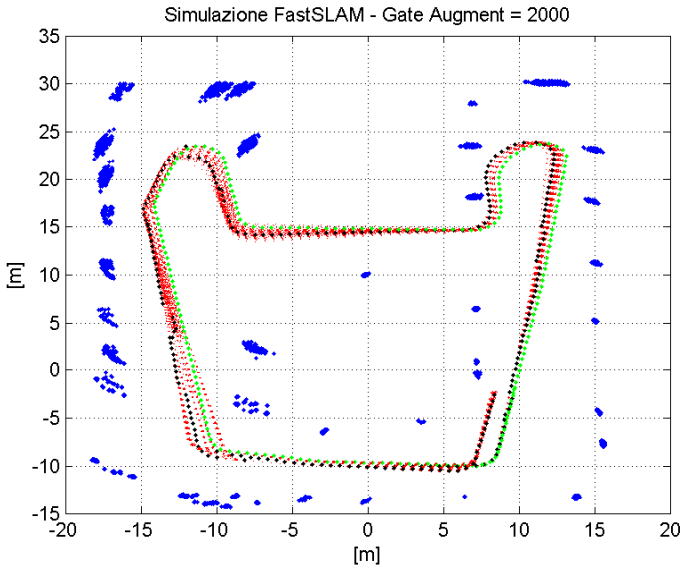

Figura 4.9: *Simulazione con parametri* $gate\_augment = 2000$.

gono molto compatte: questo è dovuto al fatto che la probabilità di osservare *feature* già viste aumenta notevolmente, con la scelta di questo parametro, e di conseguenza aumenta anche il *resampling*. Inoltre il valore troppo elevato di questo parametro implica una maggiore incertezza sulla posizione del robot, in quanto diversi landmark vicini, con questo valore di soglia, vengono associati ad un unico landmark, e quindi la precisione sulla stima della posizione chiaramente diminuisce. Il risultato di questa simulazione mostra inoltre che la traiettoria descritta dalla particella di peso maggiore si discosta quasi in tutto il percorso dalla posizione reale del robot; in ogni caso il loop viene chiuso.

L'ultimo parametro modificato nelle simulazioni è il $gate\_augment$ che rappresenta il valore minimo della distanza di Mahalanobis relativa ad una coppia di *feature* oltre il quale una *feature* osservata viene inglobata nel filtro come una nuova *feature*. Utilizzando il valore 2000, il risultato della simulazione è quello riportato in figura 4.9. Osservando la figura si nota come la traiettoria reale del robot si discosti in maniera significativa da quella della particella di peso maggiore del filtro. Questo è dovuto all'associazione di dati errata, legata al valore troppo basso del parametro $gate\_augment$. La scelta di un valore troppo basso,

come in questo caso, ha portato all'aggiunta di nuove *feature* anche se queste si discostavano di poco da *feature* già presenti nel filtro. Come si può osservare alla fine del percorso, il loop non viene chiuso e questo è dovuto al fatto che la *feature* osservata (che era stata precedentemente osservata e già inglobata nel filtro) viene di nuovo inglobata nel filtro come una nuova *feature*, e questo causa la discontinuità che si riscontra nella traiettoria riportata in figura 4.9.

In conclusione a questa serie di simulazioni si è notato che i valori in tabella 4.1, sono quelli ottimali per l'esecuzione del FastSLAM nell'ambiente riportato in figura 4.4. L'analisi delle simulazioni effettuate in Matlab, ha mostrato come i vari parametri modificabili nell'algoritmo FastSLAM influiscono soprattutto sulla distribuzione particellare e sulla distribuzione relativa alla posizione dei landmark e quindi sulla traiettoria stimata del robot.

4.2 Pre-implementazione in *C*

Dopo aver implementato preventivamente le varie funzioni descritte nella sezione 4.1, il passo successivo è stato quello di convertire le funzioni scritte in *Matlab* in linguaggio *C*.

Per la programmazione si è sfruttata la modularità già adottata in *Matlab* che ha permesso di studiare e testare ogni singola funzione, indipendentemente dalle altre, ed infine, per una validazione generale, si è studiato il comportamento di tutte le funzioni contemporaneamente, che compongono il FastSLAM.

Dopo aver riscritto la funzione generica in linguaggio *C*, per i test si sono sfruttate le potenzialità del *Matlab* di lettura e scrittura via *TCP-IP*, per eseguire i test di validazione delle funzioni. Utilizzando infatti questa capacità di comunicazione via *ethernet*, è stato possibile dopo aver ricreato la funzione in *C* clone di quella in *Matlab*, eseguire la simulazione completa ad esclusione della funzione da testare, che veniva fatta eseguire su un altro calcolatore in ambiente Linux con il file binario compilato dal sorgente *C*. Gli input e output della funzione da validare sono stati trasmessi via interfaccia *ethernet*, da un computer con sis-

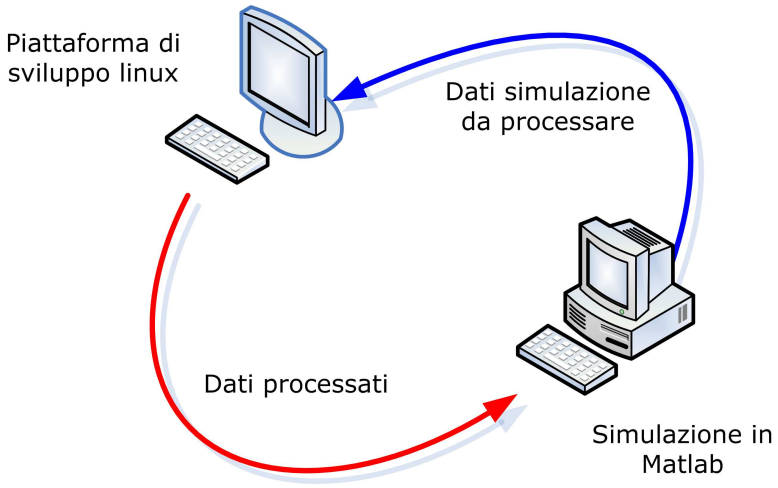

Figura 4.10: *Schema del* porting *da* Matlab *ad ambiente* Linux.

tema operativo Windows con in esecuzione *Matlab* con la simulazione verso un computer con sistema operativo Linux, con in esecuzione la funzione da testare, e viceversa; lo schema relativo a questa fase di preimplementazione è riportato in figura 4.10. Ricreare le funzioni in *C* è stata quindi una operazione più semplice e sicura, utilizzando questa strategia di implementazione.

4.3 Implementazione sul robot

In questa sezione verranno esposti i problemi incontrati e le soluzioni adottate nell'implementazione del filtro particellare per lo *SLAM problem*.

Come detto nella sezione 4.2, la fase di pre-implementazione e validazione delle singole funzioni è stata una fase cruciale nello sviluppo dell'algoritmo del *FastSLAM*; precedentemente a questa fase, si sono studiate le librerie, e l'architettura del robot *Nomad XR4000*, sul quale sono state effettuate le prove sperimentali.

Figura 4.11: *Il robot Nomad XR4000.*

4.3.1 Il *Nomad XR4000*

Il Nomad XR4000 è un sistema mobile avanzato che incorpora un sistema integrato di controllo, networking, power management e sensoristica utile allo studio nella manipolazione mobile, visione robotica, machine learning e navigazione sensoriale.

Hardware del robot

Il Nomad XR4000 è costituito da un computer basato su processore Intel Pentium III collegato tramite delle schede di interfacciamento al sistema sensoriale standard (costituito da sensori sonar, sensori di prossimità ad infrarosso, e sensori tattili) al controller del motore e all'unità di potenza che fornisce alimentazione a tutte le apparecchiature. La strumentazione sensoriale del robot è inoltre completata da una bussola digitale, da un sistema laser di rilevamento ostacoli e da un sistema di visione ad alta velocità.

Il Nomad XR4000 è un prisma a 24 facce di diametro $62\,cm$ e altezza $85\,cm$,

sostenuto da quattro ruote di metallo. Pesa circa $115\,Kg$ al netto delle sue quattro batterie[4], per un peso complessivo di $\sim 160\,Kg$. In questa trattazione verranno descritti soltanto i sensori e l'hardware in generale, utilizzato nella sperimentazione, per una descrizione dettagliata si rimanda al manuale [17].

Il sistema olonomo Nomad XR4000 Il sistema *XR C8 Holonomic Drive System*© è un sistema di guida olonomo a tre gradi di libertà (x, y, θ) senza restrizioni dovute allo spazio libero di operazione sul terreno, alle vibrazioni o alle complessità meccaniche. Questo risultato è stato raggiunto utilizzando due motori elettricamente indipendenti, uno per la sterzata e l'altro per il moto traslazionale, per ognuna delle quattro ruote che muovono il robot. L'utilizzo di queste quattro ruote lo rende un sistema ad otto assi sovravincolato. Per controllare questo sistema l'XR4000 utilizza uno speciale controller per il motore dotato di tre DSP e di un microcontrollore a 32-bit dedicato, in modo da controllare e stimare la posizione di questi otto assi.

Un corpo rigido vincolato al piano ha fino a tre gradi di libertà. Nello spazio Cartesiano questi sono rappresentati da x, y e θ. Le stesse regole si applicano ad un robot che si muove su un piano; così la posizione del Nomad XR4000 è rappresentata dai tre parametri appena definiti. Inoltre l'XR4000 può accelerare in ogni direzione in qualsiasi momento ed è quindi un sistema olonomo.

Sensori di prossimità ad infrarosso Sensus 350™ Il sistema Sensus 350™ utilizza 24 trasduttori infrarosso posizionati sia sul perimetro dell'anello superiore del robot, sia su quello inferiore. Questi sensori hanno l'utilità di essere sensori con una frequenza di acquisizione molto alta per una rilevazione di ostacoli vicini ($30-50\,cm$). La rilevazione viene effettuata emettendo una radiazione infrarossa generata attraverso LED del tipo *high-current* e rilevando la quantità di energia di ritorno dall'ostacolo attraverso fotodiodi infrarossi. Sapendo che la quantità di energia riflessa è inversamente proporzionale alla distanza dell'oggetto, è possibile calcolarne la distanza da esso. L'energia di ritorno è anche funzione del

[4] Ogni batteria pesa $12\,Kg$.

coefficiente di riflessione dell'oggetto. Ostacoli con alto coefficiente di riflessione hanno la capacità di riflettere grandi quantità di energia emessa dalla sorgente infrarossa, viceversa oggetti con un basso coefficiente la riflettono in maniera inferiore. La differenza nel coefficiente di riflessione tra gli oggetti può causare errori sostanziali se non tenuto in conto.

Ogni sensore è composto di due emettitori infrarosso Siemens SFH 34-3GaAs e di un fotodiodo al silicio Siemens SFH 20030F[5]. I due emettitori e il ricevitore sono montati orizzontalmente, sulla stessa retta, con il fotodiodo ricevitore tra i due emettitori. Per rilevare l'energia riflessa, una lettura infrarosso viene effettuata con gli emettitori accesi, e un'altra con gli emettitori spenti, in tale modo la differenza tra le due letture è proporzionale all'energia riflessa da un oggetto ma indipendente dall'energia infrarosso presente nell'ambiente di lavoro.

I fattori che influenzano le misurazioni ottenute dal Sensus 350^{TM} sono:

fattori geometrici : il principio di funzionamento del sensore di prossimità ad infrarosso è che più l'oggetto è lontano, meno quantità di emissione riflessa riceverà il fotodiodo. L'angolo di riflessione rispetto alla superficie dell'oggetto riveste un ruolo importante. Quando la superficie dell'oggetto riceve la radiazione infrarossa, parte dell'energia in arrivo viene riflessa e parte viene diffratta. Se la superficie illuminata è normale, o quasi, all'asse del ricevitore, la maggior parte della luce riflessa verrà rediretta all'indietro, contrariamente, man mano che la superficie diviene parallela all'asse dell'emettitore soltanto l'energia diffratta raggiungerà il ricevitore. La distanza e l'angolazione sono i due più importanti fattori che influenzano le misurazioni ma anche la geometria dell'oggetto ha un'importanza non trascurabile. Se l'oggetto è piccolo, la lettura sarà dipendente dall'area di oggetto illuminata, che è funzione della sua grandezza. Inoltre, se l'oggetto non ha superfici piane, la radiazione riflessa verso i sensori sarà frutto di una complessa combinazione di energia riflessa e diffratta proveniente dalle parti illuminate dell'oggetto. La figura 4.12 mostra i livelli di energia misurati

[5]Inserito in un involucro plastico (*delrin housing package*).

da $0\,cm$ a $90\,cm$, per angoli da 0^0 a 75^0, le misure sono state effettuate al buio su un foglio di comune carta per fotocopiatrice.

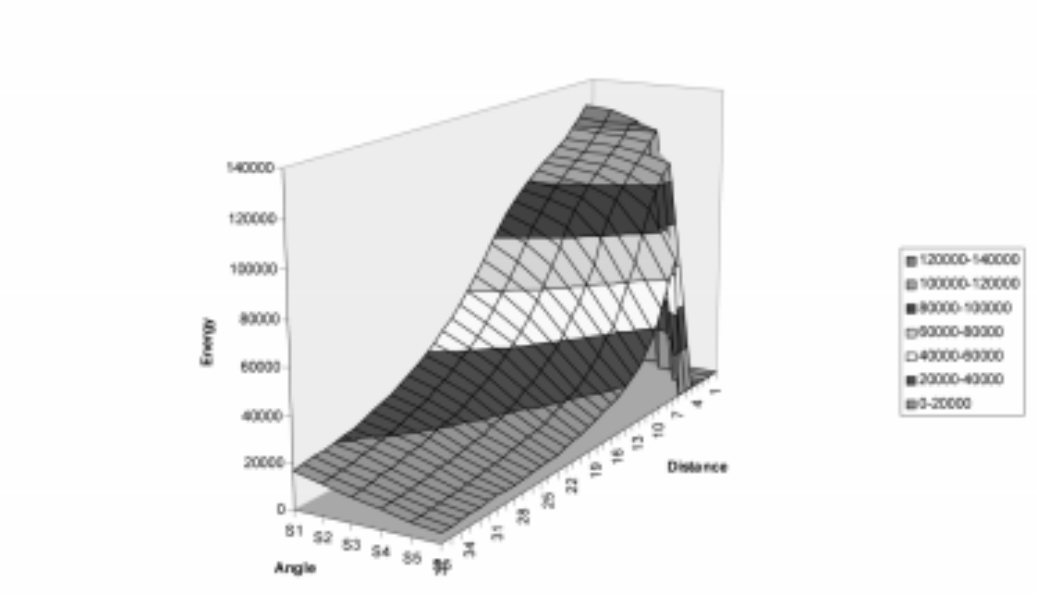

Figura 4.12: *Grafico energia / angolazione / distanza su bersaglio di superficie opaca al buio.*

fattori di illuminazione : il sensore calcola la differenza tra la luce ricevuta con l'emettitore spento e con quello acceso. Se la luce presente nell'ambiente ha sufficiente energia nello spettro dell'infrarosso, il sensore può saturare facilmente e la differenza tra le due misurazioni può essere molto piccola (poco contrasto). Il risultato è una perdita sostanziale di sensibilità. I dati di figura 4.13 sono stati rilevati in un ambiente illuminato con luce fluorescente, i dati di figura 4.14 invece con luce incandescente; comparando questa con la figura 4.12 è visibile quanto i valori energetici siano poco influenzati dalla luce fluorescente, poichè contiene una piccola porzione di spettro infrarosso.

colore e caratteristiche superficiali : il colore dell'oggetto, o più precisamente il coefficiente di riflessione nello spettro infrarosso, influenza la pre-

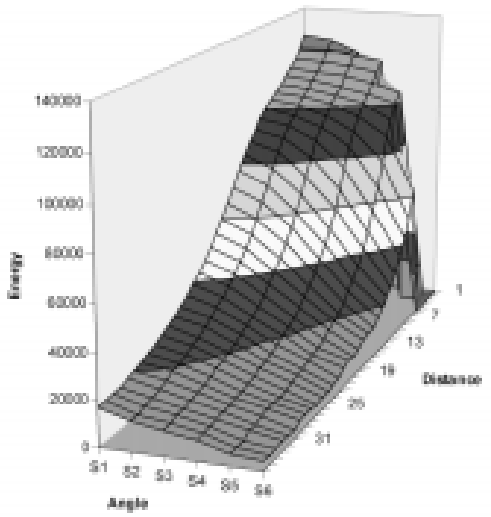

Figura 4.13: *Grafico energia / angolazione / distanza su bersaglio di superficie opaca in luce fluorescente.*

cisione della misurazione, così come le caratteristiche superficiali dell'oggetto: superfici lucide evidenziano un picco energetico per angoli vicini alla normale dell'asse del ricevitore[6]. Materiali come cemento o tessuto assorbono molta più energia radiante. Il grafico di figura 4.15 mostra l'energia misurata a 0 gradi (superficie dell'oggetto normale all'asse del sensore), per una superficie nera lucida, per un foglio nero opaco, per una tavola bianca lucida, e per un comune foglio di carta bianca. Alcuni materiali riflettono in modo differente lo spettro della luce visibile e infrarossa. Oggetti che assorbono luce visibile[7] non necessariamente assorbono luce infrarossa e viceversa.

Sistema laser Sick Sensus 550 Il Sensus 550 è un sistema di telemetria basato sul sensore Sick elettro-ottico LMS-200. Esso fornisce 180^0 di rilevamento

[6] Cioè hanno un comportamento speculare.

[7] Chiamati anche *black objects*.

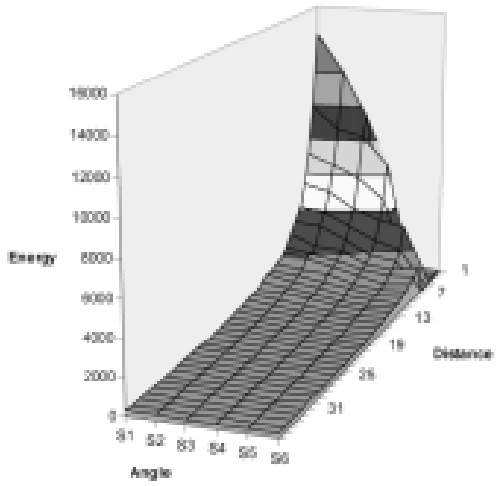

Figura 4.14: *Grafico energia / angolazione / distanza su bersaglio di superficie opaca in luce incandescente.*

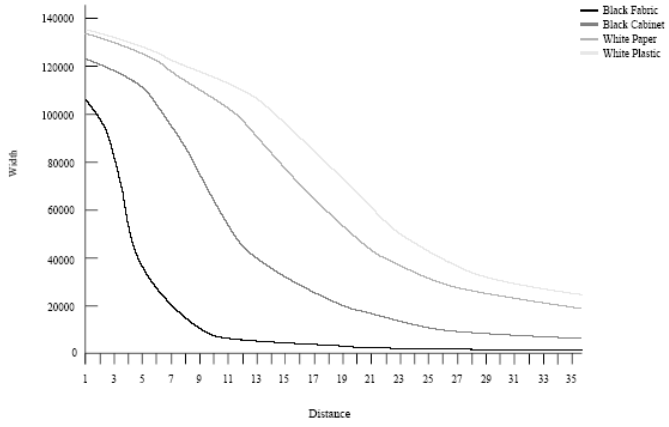

Figura 4.15: *Grafico energia / distanza ad angolazione* 0*: materiale bianco / nero, materiale opaco / lucido.*

planare ad incrementi di 0.5^0, complessivamente 360 campioni. Ogni semipiano completo di rilevazione avviene con frequenza $20\,Hz$.

Software del robot

Per motivi di prestazione e affidabilità, si è utilizzato un sistema operativo più recente rispetto a quello fornito con l'XR4000, e la scelta è caduta sulla versione 7.3 della distribuzione Red Hat di Linux con kernel 2.4.18-3, sul quale è stato possibile installare versioni più aggiornate e quindi più stabili dei driver. Differenti prove sperimentali hanno dimostrato la reale affidabilità di questa configurazione.

Originariamente sul sistema Nomad XR4000 era installato un sistema operativo Red Hat 5.3 con kernel 2.0.35, sul quale erano montati i driver delle periferiche integrate e non; il passaggio ad una versione più aggiornata di Linux Red Hat e conseguentemente ad una più aggiornata del kernel è stato ritenuto necessario per rendere più performante il sistema in vista delle applicazioni future di robotica nonché, come già detto, per renderlo più stabile; inoltre tutto il software compatibile con il kernel 2.4.x può essere installato sul Nomad.

Programmazione del robot

La struttura dati fondamentale, attorno alla quale ruota l'intera programmazione del robot, è `N_RobotState`, nella quale sono contenuti i dati e le configurazioni dei sensori; un puntatore a questa struttura è restituito dalla funzione `N_GetRobotState`. Per settare i parametri desiderati, si deve modificare il contenuto di questa struttura. In generale, i nomi delle funzioni con le quali si leggono dati ed informazioni hanno il prefisso *get,* viceversa per impostare le configurazioni dello stesso sottosistema si usa il prefisso *set.*

Nel seguito verranno esposti alcuni concetti fondamentali per la programmazione dell'XR4000, per il resto dei comandi si rimanda il lettore a [17].

Modalità di movimentazione Ci sono due possibili modalità di controllo del movimento del Nomad.

Joint mode Questa modalità tratta gli assi dell'XR4000 come giunti. In questo modo, il giunto ha una posizione, una direzione positiva, una negativa che sono definite rispetto al corpo al quale è collegato. Nella modalità *joint,* il Nomad XR4000 ha tre giunti collegati al centro del robot. Il giunto $\mathcal{Y}$ crea un movimento lineare lungo la direzione in avanti ed indietro rispetto al robot. Il giunto $\mathcal{X}$ crea un movimento lineare lungo la direzione di sinistra e di destra rispetto al robot. Il giunto di rotazione crea un movimento rotazionale rispetto al centro del robot.

Global mode Con questa modalità è possibile controllare gli assi del Nomad XR4000 rispetto ad un sistema di riferimento fisso (globale). Si può pensare al sistema di riferimento globale come ad un sistema che viene creato appena il robot viene avviato[8], e rimane fisso per tutto il tempo in cui il Nomad rimane acceso. Nella modalità *global,* il movimento lungo l'asse $\mathcal{Y}$, causa sempre un movimento lungo la direzione $\mathcal{Y}$ in riferimento all'angolo con cui è orientato il robot.

Modalità degli assi Ogni asse del robot può essere controllato in una modalità separata[9]. Questa modalità è specificata nel campo `Mode` della struttura `N_Axis` , per ogni asse. La struttura `N_Axis` fa parte della struttura `N_AxisSet` di `N_RobotState` .

Velocity Mode Nella modalità *Velocity,* l'utente può selezionare la velocità da assegnare ad un asse; la velocità viene mantenuta a quel valore finché non ne viene inserito un altro. L'utente può anche selezionare il parametro

[8] Oppure analogamente quando viene richiamata la funzione `N_SetIntegratedConfiguration`.

[9] Da non confondere con le modalità di controllo *Global* e *Joint*.

accelerazione, che rappresenta quanto velocemente può cambiare la velocità, se essa deve diminuire o aumentare. Questa modalità è specificata impostando il campo `Mode` a `N_AXIS_VELOCITY`.

Position Mode Nella modalità *Position* l'utente seleziona una posizione di destinazione per un asse, e l'asse si muove fino a quella posizione e si ferma. L'utente specifica inoltre anche una velocità desiderata ed una accelerazione. La velocità desiderata è la velocità con cui si muove l'asse per raggiungere la posizione desiderata, mentre l'accelerazione rappresenta quanto velocemente può cambiare la velocità, se essa deve diminuire o aumentare. Le due modalità possibili per la posizione sono *absolute* o *relative* e vengono specificate impostando rispettivamente il campo `Mode` a `N_AXIS_POSITION_ABSOLUTE` oppure a `N_AXIS_POSITION_RELATIVE`. La modalità di posizione *absolute* consente il movimento degli assi alla posizione assoluta rispetto alla posizione zero degli assi. La modalità di posizione *relative* consente di muovere gli assi alla posizione relativa alla posizione corrente.

La *Configurazione Integrata* Ogni robot della famiglia Nomad stima continuamente (durante il movimento) la propria posizione in coordinate Cartesiane (x, y e θ) rispetto alle coordinate del sistema di riferimento fisso. Questa è chiamata in generale *dead-reckoning position*[10], ed è un ulteriore *sensore* che indica dove si trova il robot nell'ambiente (sensori odometrici). Tuttavia l'utilità di questo *sensore* è limitata, poiché è affetto da molti errori (slittamenti delle ruote, etc.). Questo è stimato misurando i cambiamenti di posizione (Δx, Δy e $\Delta\theta$), su piccolissimi incrementi di tempo (circa $5\,ms$), e vengono integrati nel tempo (per questo viene chiamata *Configurazione Integrata*).

[10] *Dead-reckoning* significa vagare ad occhi chiusi.

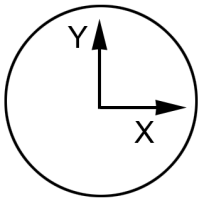

Figura 4.16: *Posizione degli assi nel nomad XR4000.*

4.3.2 Struttura del programma e strutture dati

L'implementazione del FastSLAM sul robot, ha ricalcato la struttura implementativa della simulazione in *Matlab*. Nella programmazione si sono utilizzate le librerie GSL[11] (*GNU Scientific Library*), librerie matematiche, che hanno permesso di implementare con maggiore facilità strutture dati ad alto livello, come matrici e vettori; inoltre operazioni matriciali complesse come ad esempio la fattorizzazione di Cholesky oppure operazioni delicate come inversioni di matrici, sono anch'esse contemplate dalla libreria GSL.

Prima di eseguire qualsiasi altra istruzione, viene chiamata la funzione di inizializzazione delle strutture `particles`, che contengono i dati delle particelle del FastSLAM e delle *feature*. La struttura dati di ogni particella ha la seguente forma:

```
struct particle
{
    gsl_matrix *xv;          // Robot pose
    gsl_matrix *Pv;          // Robot pose covariance matrix

    double w;                // Particle weight

    int Nf;                  // Number of features
```

[11]Per una descrizione dettagliata delle funzioni presenti nella libreria GSL, si rimanda al manuale [7].

```
    struct feature *features; // Pointer to feature list
};
```

La funzione di inizializzazione si occupa di settare a zero i campi di ogni variabile presente al suo interno, tranne il peso che viene posto pari a $\frac{1}{N_{particles}}$, e di inizializzare il puntatore alla lista delle *feature* a `NULL`. La struttura *feature* ha la seguente forma:

```
struct feature
{
    gsl_matrix *xf;       // Feature pose
    gsl_matrix *Pf;       // Feature pose covariance matrix

    gsl_matrix *zp;       // Feature predicted observation
    gsl_matrix *Hv;       // Jacobian vehicle states
    gsl_matrix *Hf;       // Jacobian feature states
    gsl_matrix *Sf;       // Innovation covariance of feature
                          // observation given
                          // the vehicle pose

    int index_feature;    // Index of feature

    struct feature *prev; // Pointer to previous structure
    struct feature *next; // Pointer to next structure
};
```

Una ulteriore struttura dati utilizzata nella computazione è la seguente:

```
struct obsv_list
{
    gsl_matrix *idf;           // current index of feature
    gsl_matrix *z;             // current observation
```

```
    int Index;                  // Index of obsv_list

    struct obsv_list *prev;     // Pointer to previous structure
    struct obsv_list *next;     // Pointer to next structure
};
```

questa struttura viene utilizzata nella funzione `resample_obs`, per effettuare il *resample* delle osservazioni in accordo al *resample* delle particelle.

Per la gestione di queste liste, sono state implementate più funzioni utilizzate per la selezione, la cancellazione e l'aggiunta di nuovi elementi nelle liste. Le funzioni addette a tali scopi sono incluse nei file `list_manage_feat.c` e `list_manage_obsv.c`.

Le funzioni principali che compongono l'algoritmo FastSLAM sono `predict` e `FastSlam_Engine`.

predict Come spiegato nella sezione 4.1, la funzione `predict` effettua la predizione della posizione del robot s_t al tempo t, nota la posizione s_{t-1} al tempo $t-1$. La chiamata a questa funzione avviene sempre, ad ogni ciclo di computazione; questa funzione riceve in ingresso le velocità impartite dal controllo ai motori, lette però dal sistema di configurazione integrata: questo è stato necessario per ottenere dei dati in ingresso alla funzione `predict` più esatti. Inoltre nella computazione è stato necessario calcolare anche l'intervallo di tempo tra un processamento e l'altro dt[12], utilizzato nella funzione per generare la posizione del robot s_t al tempo t.

FastSlam_Engine Questa funzione è il nucleo dell'algoritmo FastSLAM. A differenza della funzione `predict`, che viene chiamata ad ogni ciclo di computazione, la funzione `FastSlam_Engine` viene chiamata solo quando sono disponibili nuovi dati dai laser. Dopo aver effettuato la predizione, le particelle ven-

[12] Questo intervallo di tempo è calcolato con una precisione di $10^{-3}\,s$.

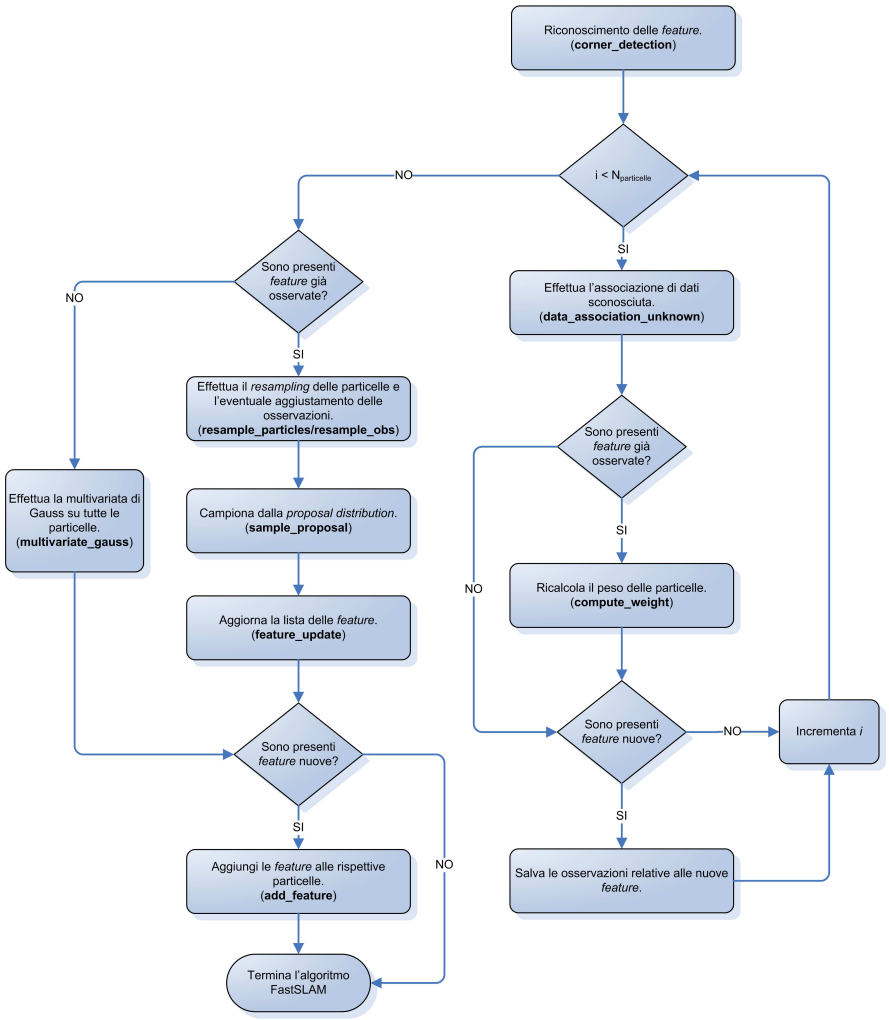

Figura 4.17: *Struttura dell'algoritmo FastSLAM.*

gono passate a questa funzione che aggiorna le strutture `particles`, in accordo all'algoritmo particellare di SLAM.

La struttura dell'algoritmo è riportata in figura 4.17, dove è possibile individuare alcune delle funzioni già descritte nella sezione 4.1. Inizialmente viene richiamata la funzione di riconoscimento delle *feature* (`features_recognition` oppure `corner_detection`), che, se sono presenti, vengono registrate ed utilizzate nei passi successivi di elaborazione. Per tutte le particelle, successivamente viene calcolata l'associazione di dati sconosciuta (con la funzione da-

`ta_association_unknown`) che in uscita restituisce il numero di *feature* nuove e quelle già osservate. Nel caso in cui sono state individuate *feature* già osservate, viene ricalcolato il peso delle particelle con la funzione `compute_weight`, e vengono salvate le osservazioni correnti per la particella corrente delle *feature* già esistenti. Nel caso in cui vi sono nuove *feature*, le osservazioni correnti per la particella corrente delle nuove *feature* vengono salvate.

Successivamente, se sono state individuate *feature* già esistenti, si procede al *resampling* delle particelle e all'eventuale aggiustamento delle osservazioni dopo il *resampling*, operato dalle funzioni `resample_particles` e `resample_obs`. Per tutte le *feature* riosservate, si effettua il campionamento dalla *proposal distribution* e si aggiorna la lista delle *feature*; le funzioni addette sono rispettivamente `sample_proposal` e `feature_update`. Se non sono state riosservate vecchie *feature*, l'algoritmo esegue una multivariata di Gauss su tutte le particelle, con la funzione `multivariate_gauss`.

Infine, se sono state trovate *feature* nuove, la funzione `add_feature` le aggiunge alle rispettive particelle e la funzione `FastSlam_Engine` termina.

4.3.3 Problemi incontrati e soluzioni suggerite per l'implementazione

Nel corso dell'implementazione e della sperimentazione sono stati affrontati dei problemi di carattere numerico, computazionale, fisico-meccanico e di *tuning* dei parametri.

Errori odometrici

Durante le prove preliminari sulla movimentazione del robot, si è osservato che, impartendo ai motori una determinata velocità traslazionale lungo un asse, il robot presentava una deriva consistente che non permetteva al robot di seguire la direzione assegnata, come è possibile osservare in figura 4.18: questo ha rappresentato un grande problema, in quanto questo tipo di errore sistematico, causato

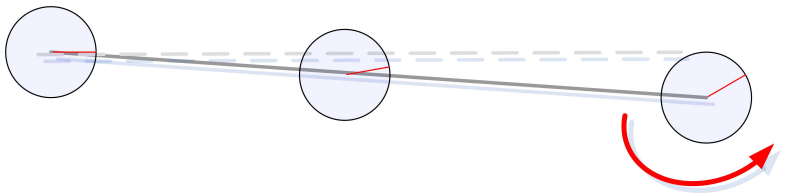

Figura 4.18: *Rappresentazione della deriva nel moto del Nomad XR4000.*

principalmente dalla non perfetta convergenza delle ruote e dal peso mal distribuito, non è contemplato dal filtro particellare, in quanto non modellizzabile esclusivamente come un disturbo di tipo Gaussiano[13]. In prima battuta si è cercato di quantificare la deriva dell'odometria, misurando la distanza percorsa dal robot su un tratto dritto e costruendo un modello della deriva degli encoder; successive prove hanno mostrato che il modello ricavato non si comporta bene quando il robot deve compiere delle traiettorie complesse che ad esempio includono più cambi di direzione.

Un altro modo per ridurre l'influenza degli errori odometrici, suggerito da Montemerlo in [13], è anche quello di osservare l'*heading* del robot, ad esempio con una bussola molto precisa, e di aggiustare i parametri del modello di movimento in base alle letture di questo strumento. Nel caso del *Nomad XR4000*, utilizzare la bussola, non è stato possibile in quanto il luogo in cui sono stati effettuati gli esperimenti è affetto da grandi disturbi elettromagnetici, che rendono impossibile l'utilizzo di qualsiasi bussola. Si è comunque cercato un modo per osservare l'*heading* del robot, e lo si è implementato, ma per via dei disturbi sulle letture della bussola, non è stato possibile verificare l'effettiva incidenza sulla correzione degli errori odometrici.

Nel caso in esame si è aumentato il valore di σ_v, relativo alla velocità traslazionale del robot, in modo tale da aumentare l'incertezza relativa alla posizione.

Tra gli errori di carattere odometrico, si è individuato anche una tendenza

[13] Gli errori odometrici hanno una componente sistematica ed una non sistematica: come mostrato in [12], l'errore di odometria è stato modellato introducendo quattro parametri, due che caratterizzano le componenti non sistematiche ed i rimanenti che caratterizzano le componenti sistematiche.

sistematica del robot a cambiare l'*heading* durante la navigazione (anche se il movimento veniva effettuato seguendo un unico asse, mantenendo lo stesso *heading* letto dai sensori). Questo tipo di disturbo, di natura sistematica, è stato molto più difficile da quantificare, e per tenerne conto si è aumentato il valore σ_ω, elemento della matrice di covarianza Q relativa alla velocità rotazionale del robot, aumentando quindi l'incertezza relativa all'orientamento.

In ogni caso, come verrà illustrato di seguito, grazie alla robustezza del FastSLAM, questi tipi di disturbi vengono comunque in parte filtrati dal filtro particellare.

Stima delle costanti che legano velocità nominale a velocità reale

Successive prove sperimentali hanno inoltre mostrato che impartire una determinata velocità ai motori, rotazionale o traslazionale che sia, non corrisponde con la velocità effettiva che viene impartita ai motori. È stato quindi necessario stimare le costanti che legano la velocità nominale a quella reale.

Per quanto riguarda la velocità traslazionale, si è proceduto assegnando un valore costante alla velocità lungo l'asse x del robot, e mantenendolo tale per tutta la prova. Dopo aver raggiunto il valore di regime, si è avviato il conteggio del tempo trascorso che è stato fermato prima di bloccare l'avanzamento del robot. Successivamente si è misurato lo spazio percorso dal robot, indicato dagli encoder; lo spazio percorso è stato misurato partendo dal momento in cui la velocità ha raggiunto il valore di regime, ed è terminato dopo 5 metri. Ripetendo più volte l'esperimento, e mediando i risultati ottenuti, è stato sufficiente calcolare una proporzione per ottenere la costante di aggiustamento dal valore nominale a quello reale della velocità traslazionale del robot. Analogo procedimento è stato portato a termine per stimare la costante relativa alla velocità rotazionale.

La stima di queste due costanti ha rivestito un ruolo cruciale, in quanto nella predizione i valori delle velocità passati alla funzione per calcolare la posizione del robot s_t al tempo t dovevano essere calcolati con una certa precisione, precisione

non raggiungibile basandosi esclusivamente sulla lettura diretta di questi valori dai motori.

Problemi numerici sulla decomposizione di Cholesky

Un altro problema incontrato, di natura numerica, è stato sulla matrice di covarianza P_v. Ad ogni iterazione dell'algoritmo FastSLAM, ogni volta, dopo aver chiamato la funzione `multivariate_gauss`, è necessario resettare la matrice di covarianza P_v, ponendola completamente a zero, per evitare di accumulare l'incertezza di posizione; anche nell'inizializzazione delle particelle è necessario porre a zero tutti gli elementi di P_v.

Sperimentalmente si è notato che la decomposizione di Cholesky[14] (richiamata più volte nel corso del programma dalla funzione `multivariate_gauss`) presuppone che la matrice da fattorizzare sia definita positiva. Imponendo che gli elementi di P_v siano tutti nulli, in alcune condizioni, è violata la condizione che P_v sia definita positiva. È stato opportuno, quindi, introdurre un fattore ϵ piccolo (dell'ordine di 10^{-7}) sulla diagonale di P_v, per garantire che la decomposizione di Cholesky, sia effettuata correttamente.

Tuning dei parametri del FastSLAM

Nel file `main.h` sono presenti le dichiarazioni delle costanti e dei parametri utilizzati nella sperimentazione. In particolare sono presenti i seguenti parametri:

`NPARTICLES` - Numero di particelle presenti nel filtro;

`GATE_REJECT` - Distanza di Mahalanobis massima per l'associazione di dati;

`GATE_AUGMENT` - Distanza di Mahalanobis minima di una nuova *feature*;

[14] Data una matrice quadrata A, definita positiva, si può trovare una matrice triangolare superiore U, con diagonale positiva tale che $A = U^T U$. Questa scrittura si chiama decomposizione di Cholesky.

`NMIN` - Numero minimo di particelle effettive prima del *resampling*;

`Z_OVERBOUND_DISTANCE` - Distanza massima valida per il laser;

`sigmaV` - Componente della matrice di covarianza Q, relativa alla velocità traslazionale del robot;

`sigmaOmega` - Componente della matrice di covarianza Q, relativa alla velocità rotazionale del robot;

`sigmaR` - Componente della matrice di covarianza R, relativa al raggio di osservazione;

`sigmaB` - Componente della matrice di covarianza R, relativa all'angolo di osservazione.

Il parametro `NPARTICLES` rappresenta il numero di particelle presenti nel filtro del FastSLAM. Un numero alto di particelle aumenta i tempi di computazione, ma permette di raggiungere un livello molto elevato di precisione[15], nella stima della posizione del robot e delle *feature*. Oltre un determinato numero di particelle, si è verificato che la precisione sulla stima non aumenta, mentre aumenta di molto il tempo necessario a portare a termine un ciclo di FastSLAM. Al di sotto di un determinato numero di particelle, per contro, il filtro non gode di una sufficiente diversità particellare, ed i risultati sperimentali hanno mostrato che la stima della posizione delle *feature* e del robot è molto poco accurata[16].

Il parametro `GATE_REJECT` rappresenta la distanza di Mahalanobis massima per l'associazione di dati sconosciuta. Questo valore influisce sull'associazione di dati, nel seguente modo: alla riosservazione di una *feature*, l'algoritmo di

[15] Inoltre un numero elevato di particelle garantisce un'ottimo livello di diversità tra le particelle, che permette appunto di ottenere grande precisione nella stima della posizione del robot.

[16] Si pensi ad esempio al caso in cui vi fossero poche particelle (dell'ordine di $5 - 15$), ed un frequente *resampling*: in questo caso la diversità tra di esse sarebbe tale da poter considerare il set di particelle come una singola particella (in quanto tutte le particelle sono vicinissime), e chiaramente il filtro si comporta male.

associazione di dati, confronta, per mezzo di questa soglia, se la *feature* osservata può essere associata ad una *feature* già vista ed incorporata nel filtro particellare. In caso affermativo, la *feature* viene associata ad una già vista, altrimenti no. Valori bassi di questo parametro implicano che si è molto sicuri dell'osservazione e quindi l'associazione di dati avviene solo nei casi in cui una *feature* osservata è molto vicina ad una già presente nel filtro. Valori elevati, per contro, tengono conto dell'errore che si può commettere sull'osservazione, e quindi una *feature* anche abbastanza distante da una già osservata, può essere associata ugualmente. In questo ultimo caso, però, si rischia di associare ad una *feature* osservata una *feature* presente nel filtro completamente diversa da quella osservata. In questo caso l'associazione di dati fallisce e le conseguenze nella stima della posizione e nella costruzione della mappa sono molto gravi.

Il parametro `GATE_AUGMENT` rappresenta la distanza di Mahalanobis minima che deve avere una *feature* per poter essere inglobata nel filtro particellare. Valori alti di questo parametro implicano che la *feature*, per essere incorportata nel filtro come nuova *feature*, deve essere molto distante dalle altre; valori bassi invece implicano che la *feature* corrente può essere incorporata nel filtro anche se è poco distante dalle altre. In questo secondo caso, se il valore è troppo basso, si rischia di incorporare nel filtro come nuove anche *feature* magari già viste; questo è un caso da evitare.

Il valore di `NMIN` rappresenta il numero minimo di particelle effettive prima del *resampling*; un valore basso implica che il *resampling* verrà effettuato con una cadenza bassa, mentre un valore elevato implica che il *resampling* è effettuato più spesso. Sono da evitare in generale valori o troppo bassi o troppo alti, in quanto nel caso di valori troppo bassi, si rischia di avere un tasso di resampling troppo basso e quindi una troppo elevata diversità nelle particelle, che in qualche caso può portare anche alla divergenza del filtro; valori troppo alti, per contro, implicano un elevato tasso di *resampling* che non permette al filtro di mantenere una sufficiente diversità nelle particelle.

Il parametro `Z_OVERBOUND_DISTANCE`, cioè la distanza massima valida per il laser, è utile per limitare il campo visivo del robot per l'algoritmo di navigazione e

per l'algoritmo di riconoscimento delle *feature*. Diminuire questo valore significa sostanzialmente ridurre, per ogni ciclo di FastSLAM, il numero di *feature* presenti nel campo visivo e quindi aumentare la velocità di computazione, a scapito del tasso di riosservazione di alcune delle *feature* presenti nel reale campo visivo del robot.

`sigmaV` e `sigmaOmega` sono la componente della matrice di covarianza Q, una relativa alla velocità traslazionale e l'altra relativa alla velocità rotazionale del robot.

$$Q = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \tag{4.39}$$

Aumentare questi parametri significa aumentare l'incertezza sia sulla posizione, che sull'orientamento del robot. Nel caso specifico del *Nomad XR4000*, questi valori sono stati scelti abbastanza elevati, per i problemi di errori odometrici descritti precedentemente.

`sigmaR` e `sigmaB` sono la componente della matrice di covarianza R (relativa ai rumori di osservazione), rispettivamente relativi al raggio e all'angolo di osservazione.

$$R = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_b^2 \end{bmatrix} \tag{4.40}$$

Aumentare questi parametri significa aumentare l'incertezza sulle osservazioni. Essendo il laser *Sick S550* molto preciso, i valori di questi parametri sono stati scelti abbastanza contenuti.

4.4 Stima posizione con *scan-matching*

Risultati sperimentali sul modello di moto, hanno portato a considerare un differente approccio per stimare le velocità traslazionali e rotazionali in ingresso alla predizione. Il *feedback* del controller del motore si è dimostrato inaffidabile, in quanto impartendo al motore una determinata velocità, quella non era l'effettiva velocità di procedimento. Si è così introdotto un fattore che tenesse conto

della differenza tra la velocità impartita ai motori e quella effettiva. Nonostante sia stato utilizzato questo fattore, in percorsi estesi, questa stima risultava poco corretta, soprattutto per quanto riguarda la velocità angolare; inoltre i valori riportati dal controller del motore non registrano completamente tutto l'effettivo spostamento del robot, quindi la modellizzazione dell'errore sistematico influisce sull'effettivo funzionamento del filtro particellare. Per tenere conto di questo tipo di errore, si sono prodotti due approcci:

- aumentare enormemente l'incertezza di posizione, modificando i termini della matrice Q;
- l'approccio *ICP Scan-Matching*[17].

4.4.1 Aumento dell'incertezza di posizione

Per tener conto degli errori generati dal sistema sensoriale odometrico del robot, si è in prima battuta aumentato il valore dei termini presenti sulla diagonale della matrice Q, che rappresentano rispettivamente le varianze della velocità traslazionale σ_V e della velocità rotazionale σ_Ω. In questo modo le particelle mostravano una distribuzione planare vasta, e quindi per diminuire la granularità del filtro è stato necessario aumentare di molto il numero delle particelle presenti nel filtro (quasi tre volte tanto), nonché i tempi di elaborazione del FastSLAM. La mappa così generata è risultata essere poco collimata a causa di un'incertezza così grande.

4.4.2 *ICP Scan-Matching*

L'*ICP* (acronimo per *Iterative Closest Point*), è stato utilizzato per effettuare una stima delle velocità tra due istanti di tempo, utilizzando semplicemente le letture del sensore laser.

[17] Per ulteriori approfondimenti sulla tecnica *ICP Scan-Matching* si rimanda a [26].

L'algoritmo *ICP* è un algoritmo iterativo di allineamento. Il suo funzionamento è suddiviso in tre fasi:

1. stabilire corrispondenze tra coppie di punti salienti in due strutture che devono essere allineate in base alla distanza;
2. stimare la trasformazione rigida che meglio mappa il primo membro della coppia nella seconda;
3. applicare questa trasformazione a tutti i punti della prima struttura.

Questi tre passi sono applicati iterativamente fino al raggiungimento della convergenza.

Rigid Iterative Closest Point Algorithm

Sia $\mathcal{S}$ un insieme di N_s punti $\{\vec{s}_1, \ldots, \vec{s}_{N_s}\}$ e sia $\mathcal{M}$ il modello. Si indica con $||\vec{s} - \vec{m}||$ la distanza Euclidea tra i punti $s \in \mathcal{S}$ e $m \in \mathcal{M}$. Sia $CP(\vec{s}, \mathcal{M})$ il punto più vicino in $\mathcal{M}$ al punto saliente $\vec{s}$.

1. Sia $T^{[0]}$ la stima iniziale della trasformazione rigida.
2. Ripetere i seguenti passi da $k = 1$ fino a $k = k_{max}$:
   - **a** Calcolare il set delle corrispondenze $\mathcal{C} = \bigcup_1^{N_s} \left(\vec{s}_i, CP(T^{[k-1]}(\vec{s}_i), \mathcal{M})\right)$.
   - **b** Calcolare la nuova trasformazione Euclidea $T^{[k]}$ che minimizza l'errore quadratico medio tra le coppie di punti contenute in $\mathcal{C}$.

Applicazione dell'*ICP Scan-Matching* per lo SLAM

L'applicazione dell'*ICP Scan-Matching* restituisce un Δ relativo alla posizione precedente. Si sono quindi calcolate le velocità medie intercorse nell'intervallo di tempo dt, da inserire nella funzione di predizione, così sollevata dal compito della rilevazione e correzione dell'errore sistematico.

I risultati sperimentali confermano la validità di questa metodica, in tale modo il FastSLAM applicato anche in ambienti estesi si comporta bene.

4.5 Risultati sperimentali

In questa sezione verranno illustrati i risultati relativi alle prove sperimentali effettuate sul robot mobile *Nomad XR4000* presso il laboratorio di automatica dell'Università di Roma "Tor Vergata".

Nelle varie prove la posizione del robot è stata inizializzata in un punto della mappa, ad esso completamente sconosciuto: in questo punto i sensori odometrici sono stati azzerati. A questo punto è stata avviata la prova, eseguendo da terminale il comando `./FastSLAM`; il programma, come già detto, integra l'algoritmo di navigazione con il FastSLAM. Durante l'esecuzione vengono salvati su disco diversi dati necessari ad analizzare a posteriori il comportamento del robot: questi sono i dati dei sensori ad infrarosso, i dati del laser, i dati dei sensori odometrici ed infine una serie di dati interni al FastSLAM. Con questi dati è stato possibile ricreare un clone del programma `FastSLAM` che è stato eseguito sul robot: con questo programma clone, utilizzabile *offline*, e con i dati raccolti dal robot, è stato possibile ricreare in maniera esatta la prova sperimentale e quindi molti parametri aggiustabili sono stati opportunamente tarati in questo ambiente, senza la necessità di eseguire una prova sul robot per ogni parametro modificato.

I parametri del FastSLAM utilizzati in generale sono quelli riportati in tabella 4.2. Nella tabella 4.2 e 4.3, oltre ai parametri già illustrati nella tabella 4.1, sono presenti i parametri relativi all'algoritmo di *scan-matching*, cioé il parametro NIT, che rappresenta il numero di iterazioni dell'algoritmo ICP per ogni *scan-matching*, il parametro $gate\_1$ che rappresenta la prima soglia dell'associazione di dati per ogni punto nella scansione, il parametro $gate\_1$ che rappresenta la seconda soglia dell'associazione di dati per ogni punto nella scansione, $V_{saturazione}$ che rappresenta la velocità traslazionale massima alla quale può andare il robot

σ_V	1.0
σ_Ω	$1.0\,\frac{\pi}{180}$
$N_{particles}$	30
$N_{effective}$	$0.95\,N_{particles}$
$gate\_reject$	100
$gate\_augment$	800

Tabella 4.2: Parametri standard del FastSLAM utilizzati per la prova sul robot e modificabili.

σ_R	0.1
σ_B	$0.5\,\frac{\pi}{180}$
NIT	20
$gate\_1$	1
$gate\_2$	0.1
$V_{saturazione}$	100
$\Omega_{saturazione}$	100
A	50
$\dot{\Omega}$	50

Tabella 4.3: Parametri standard del FastSLAM utilizzati per la prova sul robot e non modificati.

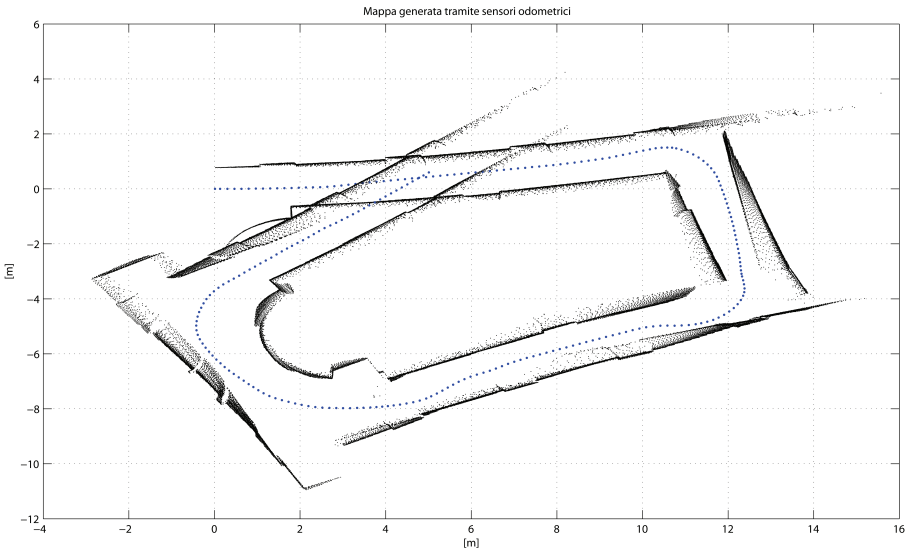

Figura 4.19: *Mappa generata con i sensori odometrici e i dati del laser.*

espressa in mm/s, $\Omega_{saturazione}$ che rappresenta la velocità rotazionale massima alla quale può ruotare il robot espressa in $mrad/s$, A che rappresenta l'accelerazione traslazionale espressa in mm/s^2 ed infine $\dot{\Omega}$ parametro che rappresenta l'accelerazione rotazionale del robot espressa in $mrad/s^2$.

Ogni prova è durata circa 7 minuti e 30 secondi e lo spazio percorso dal robot è stato di circa 36 metri. Nel percorso erano presenti ostacoli mobili, persone che transitavano accanto al robot, e ostacoli fissi, come muri, porte e altri oggetti tipici di ambienti *indoor*. Generalmente le prove sono andate a buon fine, e sono stati prodotti diversi grafici modificando alcuni dei parametri definiti in tabella 4.2, con lo stesso criterio utilizzato nelle simulazioni in Matlab.

Utilizzando i soli sensori odometrici combinati con i dati provenienti dal sensore laser, la mappa costruita è quella riportata in figura 4.19. Come è facile vedere nella figura, la mappa generata con l'ausilio della sola odometria è imperfetta, ed in particolare si può notare che la mappa non viene chiusa. Questo risultato non risulta accettabile per la costruzione della mappa dell'ambiente; si può notare inoltre che si ha una deriva verso sinistra consistente, sinonimo del

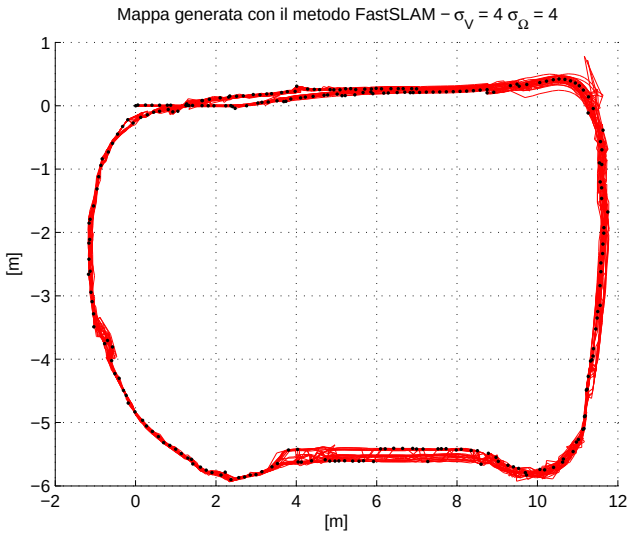

Figura 4.20: *Traiettoria generata con parametri* $\sigma_V = 4$ *e* $\sigma_\Omega = 4$.

fatto che l'odometria è affetta da errori sistematici significativi[18].

Successivamente si è proceduto a modificare alcuni dei parametri fondamentali del FastSLAM, modificandone uno per volta ed analizzando l'influenza di questi sulla stima della posizione del robot e sulla costruzione della mappa.

Inizialmente si sono modificati i parametri σ_V e σ_Ω, ed i risultati della prova, per quanto riguarda la traiettoria delle particelle, riportata in rosso, e quella della particella di peso maggiore, sono riportati in figura 4.20. Nella figura 4.20 si può notare che il loop viene chiuso, nonostante sia stata aumentata di molto l'incertezza sui parametri σ_V e σ_Ω; questo si traduce in un aumento spaziale della distribuzione delle particelle e di conseguenza in una maggiore incertezza sulla posizione del robot e delle *feature*. Nella parte finale si può osservare che quando la distribuzione di particelle si allarga, la posizione della particella di peso maggiore presenta delle oscillazioni: questo comporta degli errori notevoli nella costruzione della mappa.

Il parametro modificato per la seconda prova è stato $N_{effective}$, che è stato

[18] Per una trattazione degli errori odometrici del *Nomad XR4000* si rimanda alla sezione 4.3.3.

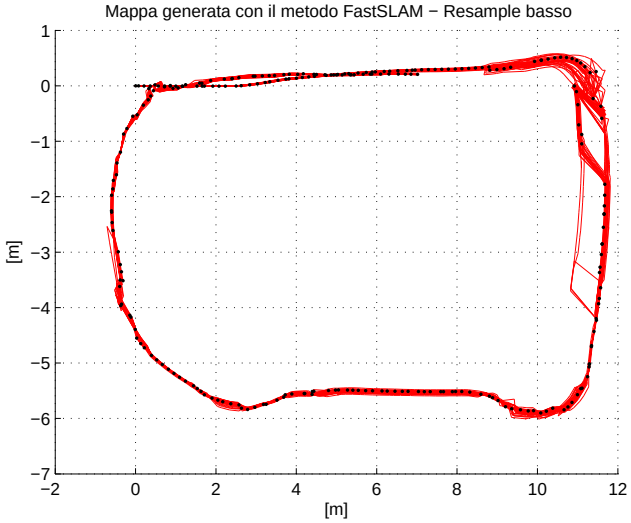

Figura 4.21: *Traiettoria generata con parametri* $N_{effective} = 0.25\, N_{particles}$.

modificato con $0.25\, N_{particles}$. Il risultato è riportato in figura 4.21. Anche in questo caso il loop viene chiuso nonostante lo scarso tasso di *resampling*, ma è evidente che in alcuni punti della traiettoria del robot, la precisione non è affatto accettabile; di conseguenza anche la traiettoria della particella di peso maggiore non è esatta.

L'effetto del basso tasso di *resampling* si ripercuote inoltre anche sulla distribuzione delle particelle, che risulta in alcuni punti molto sparsa spazialmente e anche quella dei landmark associati alle particelle.

Successivamente si è modificato il parametro $gate\_reject$ impostandolo al valore 20; il risultato della prova è riportato in figura 4.22. Come è facile vedere in figura 4.22 la distribuzione delle particelle risulta molto sparsa e aumenta con l'aumentare dello spazio percorso dal robot; questo è dovuto essenzialmente al fatto che le misure effettuate dal laser, e quindi il riconoscimento delle *feature* connesse alle osservazioni, sono abbastanza affette da rumore e quindi un valore troppo basso della soglia $gate\_reject$ è opportuno solo se le misure sono poco affette da rumore; in questo caso l'associazione di dati fallisce in alcuni punti e alcune *feature* nuove vengono associate a *feature* già inglobate nel filtro differenti

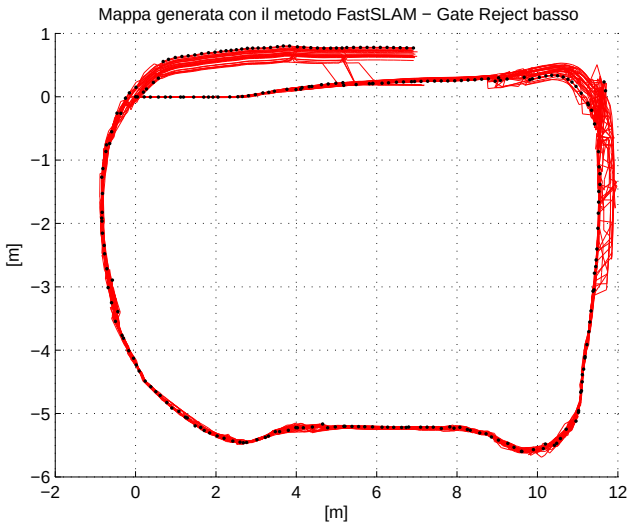

Figura 4.22: *Traiettoria generata con parametri* $gate\_reject = 20$.

però da quelle reali.

Osservando la figura risulta chiaro che il fallimento nell'associazione dei dati in alcuni punti porta inevitabilmente ad un errore troppo grande alla fine del percorso, e quindi il FastSLAM non converge, ossia il loop non viene chiuso. In ogni caso, dalla figura si può notare che alcune particelle, dopo aver riosservato alcune *feature* già viste all'inizio, dopo un *resampling*, convergono alla posizione reale del robot.

Modificando il parametro $gate\_augment$ al valore 2000, il risultato della sperimentazione è quello riportato in figura 4.23. Il risultato di questa prova è in generale soddisfacente; aumentando il valore di $gate\_augment$, si escludono dal filtro più *feature* osservate, in quanto aumenta il *range* tra il $gate\_reject$ ed il $gate\_augment$, che rappresenta il campo entro il quale le *feature* vengono scartate; non inglobando nel filtro queste *feature*, si rischia di avere meno precisione nella stima della posizione, che nella figura 4.23 è visibile in alcune parti del percorso. In generale però la particella di peso maggiore non presenta salti notevoli e la traiettoria di questa non si discosta di molto da quella reale.

Successivamente, ripristinando i parametri in tabella 4.2, si è modificato il

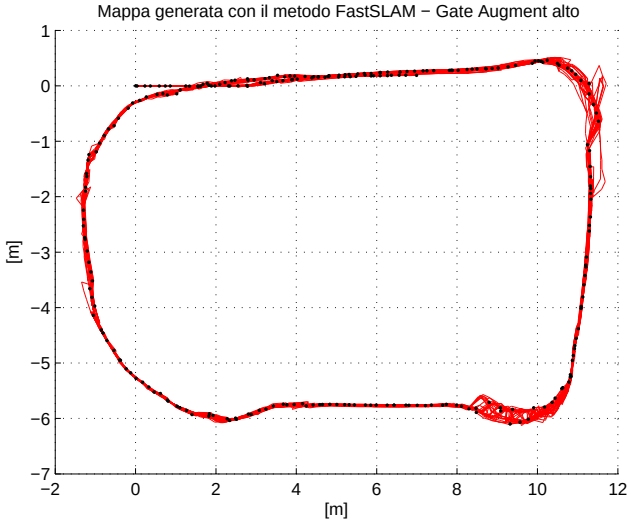

Figura 4.23: *Traiettoria generata con parametri* $gate\_augment = 2000$.

parametro $N_{particles}$ con il valore 100. In figura 4.24 è riportato il risultato della prova. Aumentare il numero di particelle, in generale significa aumentare la risoluzione del filtro particellare, in quanto con un numero maggiore vengono contemplate più traiettorie e quindi aumenta la diversità tra le particelle. Nella figura 4.24 è possibile notare come la distribuzione di particelle in alcuni punti risulta molto sparsa, poiché non viene effettuato il *resampling*, e tende ad allargarsi finché non viene effettuato un ricampionamento delle particelle che escluda quelle meno probabili, come è facile notare nella parte centrale della figura. La particella di peso maggiore non presenta eccessive discontinuità, nonostante la distribuzione sparsa ed il *resampling*, ed il loop viene chiuso.

Infine in figura 4.25 viene mostrato il risultato migliore delle prove sperimentali, relativo ai parametri riportati nelle tabelle 4.2 e 4.3. La mappa costruita con il metodo FastSLAM riportata in figura 4.25 mostra che la traiettoria delle particelle è accettabile in tutto il percorso ed in particolare, alla fine del percorso si può osservare come da una condizione in cui le particelle sono abbastanza sparse, l'osservazione di *feature* già presenti nel filtro comportano un *resampling* delle particelle che porta alla convergenza il FastSLAM.

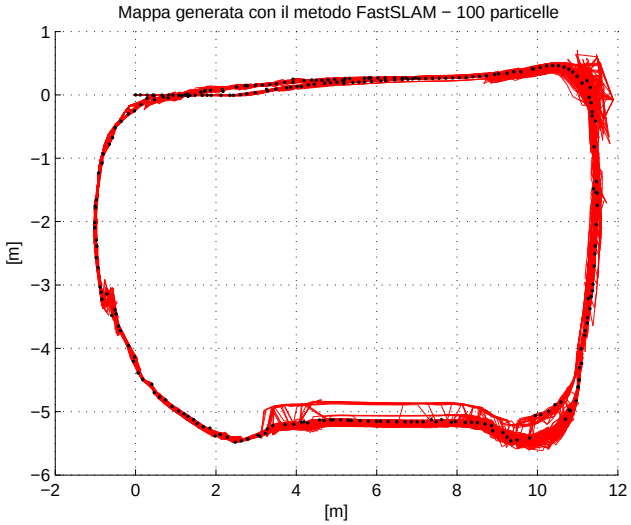

Figura 4.24: *Traiettoria generata con parametri* $N_{particles} = 100$.

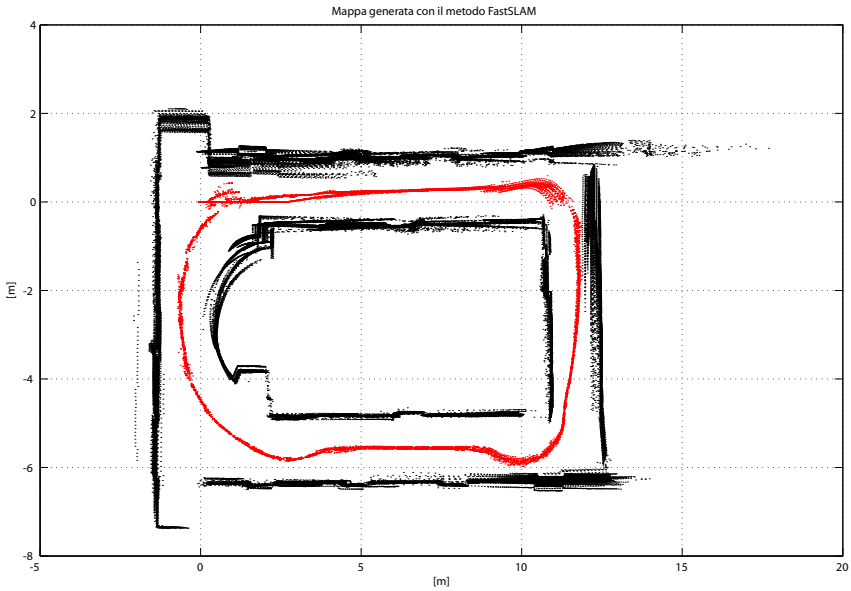

Figura 4.25: *Mappa generata con il metodo FastSLAM con i parametri riportati nelle tabelle 4.2 e 4.3.*

La relativa mappa costruita sulla traiettoria della particella di peso maggiore, e costruita utilizzando come informazioni accessorie la posizione delle *feature* nella mappa nonostante alcune imprecisioni, è di sicuro molto più accurata ed esatta rispetto a quella riportata in figura 4.19, che era stata prodotta utilizzando la sola odometria.

È facile osservare inoltre dal confronto tra la mappa generata con i sensori odometrici e quella creata con il FastSLAM che anche la componente di errore sistematico associata all'odometria è corretta dal filtro particellare.

Capitolo 5

Navigazione autonoma

Per rendere completamente autonomo un robot mobile, oltre ad un algoritmo di localizzazione e di costruzione di mappe, come la particolare tecnica di SLAM descritta nei precedenti capitoli, è necessario fornire il robot anche di un algoritmo di navigazione e controllo.

Nel presente lavoro il problema della navigazione è stato suddiviso in problema di esplorazione e di *obstacle avoidance*.

5.1 Esplorazione di ambienti sconosciuti

Dopo aver verificato l'efficienza della tecnica del FastSLAM si è sviluppato un metodo di navigazione in ambiente sconosciuto. Data la scarsa capacità computazionale del robot, occupato principalmente dall'esecuzione dell'algoritmo di FastSLAM, è stato necessario implementare una efficiente *routine* di esplorazione.

L'ambiente che esplora il robot, è un ambiente totalmente sconosciuto, non è stata data nessuna informazione *a priori*, proprio per rendere il robot capace di navigare autonomamente. La tecnica esplorativa si basa su tecniche di navigazione *grid-based* a dimensione fissa. Questa scelta è stata dettata principalmente da motivazioni computazionali e di occupazione di memoria. Una mappa di tale

tipo è capace di registrare l'esplorazione di $2500\,m^2$ in poco spazio di memoria, e con una ridotta necessità computazionale. L'algoritmo di esplorazione decide la direzione di moto per la navigazione attraverso un semplice modello decisionale, basandosi sulla direzione, registrata dal sensore laser, più libera e sulla base delle informazioni che possiede riguardo allo stato di esplorazione. L'algoritmo di navigazione decide quindi i riferimenti di V e Ω da passare al controller del motore attraverso l'utilizzo di un semplice controllo proporzionale. Per la descrizione dettagliata dell'algoritmo di esplorazione si rimanda a [22].

5.2 *Obstacle avoidance*

Il problema della navigazione e del controllo di un robot mobile è strettamente legato, se soprattutto si considera la navigazione in ambienti *indoor* o in ambienti non statici, al problema dell'*obstacle avoidance*.

Per studiare il modo di evitare un ostacolo nello spazio di lavoro del robot, è necessario innanzitutto considerare la natura degli ostacoli presenti nel luogo in cui opera il robot, nonché la natura dei sensori utilizzati per riconoscere gli ostacoli. Ogni tipologia di *obstacle avoidance* infatti varia al variare dei due aspetti considerati; si pensi ad esempio a come possono essere differenti gli *obstacle avoidance* implementati su due robot, uno che naviga in un ambiente molto esteso, mentre un altro che esplora l'interno di un edificio.

Nel caso specifico trattato in questo lavoro, la tipologia di ostacoli presenti nello spazio operativo del robot sono muri, spigoli, porte e anche ostacoli più complessi come pareti non dritte. Per quanto riguarda la sensoristica, per il *Nomad XR4000* si è scelto di utilizzare il sistema di rilevamento ad infrarosso, composto, come già esposto nella sezione 4.3.1, da due corone ognuna con 24 sensori ad infrarosso con un *range* pari a $30 - 50\,cm$. La scelta è ricaduta su questo tipo di sensori, per via del *range* e per il fatto che questi sono poco affetti da disturbi[1].

[1]Inizialmente si è pensato di utilizzare i sonar presenti, ma l'ipotesi è stata presto accan-

Come nel caso dell'algoritmo di esplorazione, anche per l'*obstacle avoidance* si è prestata particolare attenzione acciocché la complessità computazionale fosse molto ridotta, in quanto la maggior parte della potenza di calcolo è richiesta dal FastSLAM.

Il principio di funzionamento dell'*obstacle avoidance* è il seguente: considerando le letture dei sensori[2] ad infrarosso del robot, si genera un vettore che ha il modulo inversamente proporzionale alla distanza degli ostacoli dal robot e la fase calcolata in maniera analoga sulla base delle misure dei sensori. Dopo aver generato ad ogni passo di *obstacle avoidance* questo vettore risultante, l'algoritmo sceglie la direzione e le velocità di riferimento da impartire ai motori, sulla base di un controllo proporzionale, in modo tale da poter evitare in maniera efficace l'ostacolo.

5.2.1 Descrizione dell'algoritmo di *obstacle avoidance*

L'idea che sta alla base dell'*obstacle avoidance* implementato è un principio geometrico abbastanza semplice. Come è facile vedere nella figura 5.1, ad ogni sensore che individua un ostacolo, è associato un vettore; il vettore caratteristico dell'ostacolo è ricavato considerando come modulo una quantità inversamente proporzionale alla distanza che ogni singolo sensore legge, mediata su tutti i sensori che individuano ostacoli[3]; la fase del vettore, invece viene ricavata dalle fasi dei vettori associati ad ogni singolo sensore. In figura 5.1 è riportato il vettore risultante che rappresenta l'ostacolo incontrato durante la navigazione. Per mostrare che il metodo di scelta del vettore rappresentativo dell'ostacolo è efficiente anche in presenza di più ostacoli, si rimanda alla figura 5.2, dove è riportato un corridoio in cui il robot naviga; è immediato osservare che il vettore

tonata, in quanto la precisione dei sonar, affetti da echi multipli e da altri tipi di disturbi, è molto bassa; inoltre per l'*obstacle avoidance* non era necessario avere un *range* così esteso.

[2]Ogni sensore è distanziato dall'altro di 14^o; la corona dei sensori ad infrarosso ha la seguente distribuzione: $[7.221.6 \ldots 172.8187.2 \ldots 338.4352.8]$.

[3]Il sensore $i-esimo$ con $i = 1, \ldots, 48$ individua un ostacolo se la distanza da questo è minore di una determinata soglia, che verrà descritta più avanti nella trattazione.

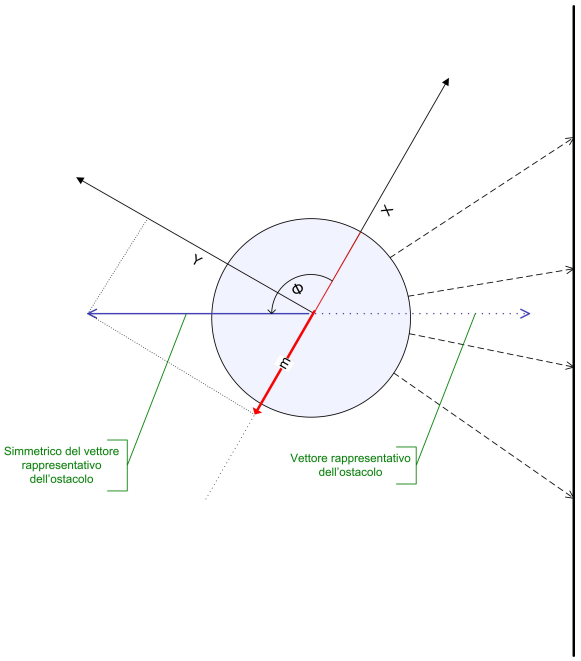

Figura 5.1: *Principio geometrico alla base dell'obstacle avoidance.*

è individuato correttamente. Dopo aver definito questo vettore rappresentativo dell'ostacolo, vengono individuate altre due grandezze: proiettando lungo l'asse x il vettore individuato, la prima grandezza di interesse è il modulo del vettore proiettato lungo x. Considerando poi il simmetrico del vettore rappresentativo dell'ostacolo, si considera come grandezza, la fase di quest'ultimo vettore.

Dopo aver definito queste grandezze, l'algoritmo di *obstacle avoidance* procede alla generazione di valori di riferimento per la velocità di rotazione Ω e per la velocità di traslazione V, scelti sulla base di un controllo proporzionale; questi valori sono scelti, osservando il particolare ostacolo presente nelle vicinanze del robot (e quindi il vettore che lo rappresenta).

Per calcolare l'aggiustamento della velocità traslazionale V si utilizza il modulo della proiezione sull'asse x del vettore rappresentativo dell'ostacolo; in particolare, se il modulo è negativo o nullo, significa che l'ostacolo è posteriore al robot e quindi il valore della velocità V può essere anche quello di crociera, mentre se il modulo risulta positivo, la velocità di riferimento V_r viene scelta in modo

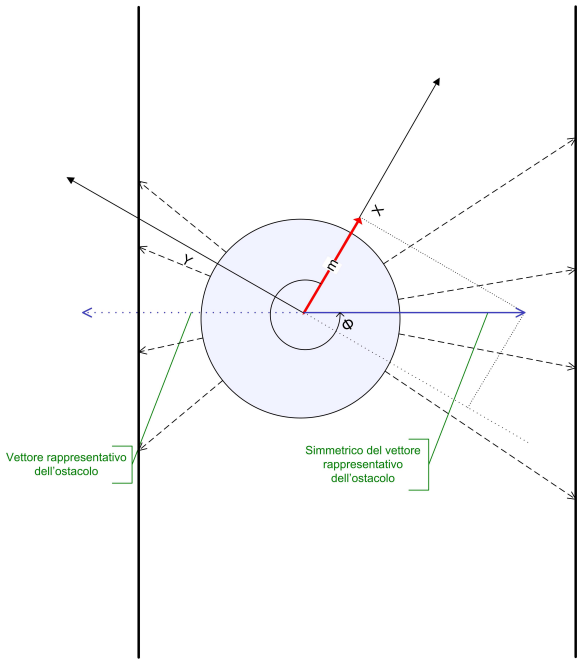

Figura 5.2: *Efficienza dell'obstacle avoidance in presenza di più ostacoli.*

tale che sia proporzionale al valore di questo modulo m_V, come descritto in 5.1.

$$V_r = K_V \, m_V \tag{5.1}$$

Per quanto riguarda l'aggiustamento della velocità rotazionale Ω, si utilizza la fase del simmetrico del vettore rappresentativo dell'ostacolo; in particolare questo significa scegliere una velocità di rotazione di riferimento Ω_r, per allontanarsi dall'ostacolo, proporzionale al valore della fase ϕ_Ω di questo vettore, come descritto in 5.2.

$$\Omega_r = K_\Omega \, \phi_\Omega \tag{5.2}$$

Le costanti dei due controllori P, cioè K_V e K_Ω, sono state tarate in modo opportuno dopo varie fasi intesive di test sul robot; i valori utilizzati, hanno permesso di ottenere un ottimo comportamento nell'evitare ostacoli durante la navigazione.

Implementazione dell'*obstacle avoidance*

L'algoritmo di *obstacle avoidance*, incluso nel file `obstacle_avoidance.c`, è stato implementato in *C*, dividendolo in due funzioni:

- `detect_impact`;
- `obstacle_avoidance`.

`detect_impact` In questa funzione viene richiamata la lettura dei sensori di prossimità ad infrarosso (distanza dall'ostacolo), che vengono salvati in una variabile, che prima vengono confrontate con una soglia `IR_OA_THRESHOLD`, che definisce la soglia sotto la quale individuare se è presente o meno l'ostacolo nelle vicinanze di ogni sensore. Successivamente, dopo aver individuato se è necessario o meno l'intervento dell'*obstacle avoidance*, la funzione restituisce 1 se un ostacolo è troppo vicino, 0 altrimenti.

`obstacle_avoidance` In questa funzione viene implementato l'algoritmo vero e proprio di *obstacle avoidance*. Dopo aver richiamato `detect_impact`, se è necessario l'intervento dell'*obstacle avoidance*, l'algoritmo verifica prima se ci sono abbastanza sensori che individuano ostacoli nelle vicinanze del robot, verifica regolata tramite la soglia `WALL_IMPACT`. Se ci sono abbastanza sensori che individuano ostacoli nelle vicinanze (e se chiaramente `detect_impact` ritorna 1), si procede con il calcolo della somma vettoriale dei vettori relativi a tutti i sensori interessati nell'imminente impatto. A questo punto si calcola il modulo e la fase del vettore risultante; con questi valori si calcolano infine i valori di riferimento V_r e Ω_r da impartire ai motori per garantire l'aggiramento dell'ostacolo.

Inoltre l'*obstacle avoidance* contempla anche il caso in cui vi sia un ostacolo frontale al robot, impartendo dei valori di velocità traslazionale e rotazionale di riferimento molto alti; in particolare, se l'impatto è imminente, e l'ostacolo è ad esempio un muro frontale, la velocità traslazionale di riferimento viene addirittura

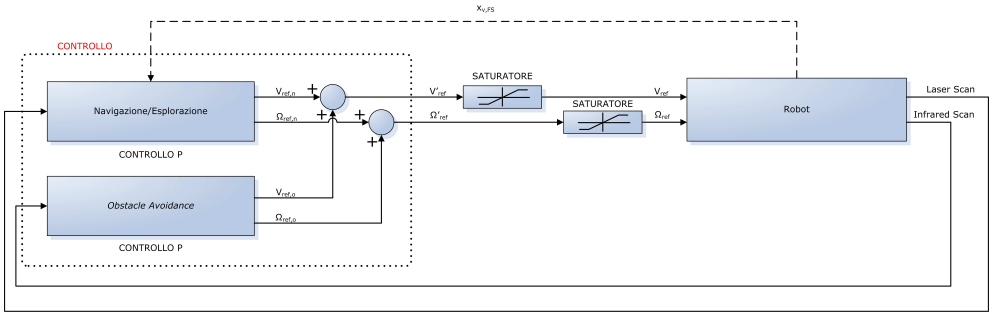

Figura 5.3: *Schema di controllo.*

posta a 0. Nel caso invece di impatto imminente laterale, la velocità rotazionale di riferimento viene impostata pari al valore massimo contemplato dal controllo.

5.3 Sistema di controllo

Viene qui proposto il sistema completo di controllo del robot. I riferimenti V_r e Ω_r decisi dalla funzione `control`, che si occupa dell'algoritmo di navigazione / esplorazione, vengono modificati, se necessario, dall'algoritmo di *obstacle avoidance* (`obstacle_avoidance`). Passano uno stato di saturazione per evitare valori di velocità troppo alti o inferiori al minimo progettuale. Lo schema complessivo di controllo è riportato in figura 5.3.

Capitolo 6

Conclusioni

Lo scopo del lavoro presentato in questa tesi è stato di studiare un metodo *online* per l'implementazione del *FastSLAM*, una particolare tecnica di *Simultaneous Localization And Mapping* basata su una particolarizzazione del filtro particellare *Rao-Blackwellized*. Nel corso della trattazione è stata illustrata la metodologia di implementazione: in primo luogo si è creato un ambiente simulativo in Matlab per generare mappe in cui far navigare il robot virtuale per testare l'algoritmo *FastSLAM* e sono state portate a termine molte simulazioni per testare l'efficacia del metodo di *SLAM* e per impostare i parametri variabili dell'algoritmo. Successivamente si è passati alla pre-implementazione dell'algoritmo in linguaggio *C*, testando ogni singola funzione precedentemente implementata in Matlab e riportandola in *C* in ambiente Linux. Infine dopo aver implementato l'algoritmo di *obstacle avoidance* e quello di esplorazione, si è proceduto a verificare la correttezza del programma completo.

Sono state effettuate molte prove sperimentali sul robot mobile *XR4000* della famiglia *Nomad* presso il Laboratorio di Automatica dell'Università di Roma "Tor Vergata", che hanno condotto a risultati molto soddisfacenti per quanto riguarda sia la navigazione che il *FastSLAM*. Il confronto tra la mappa costruita utilizzando i sensori odometrici e quella costruita con l'ausilio del *FastSLAM* mostra che gli

errori sistematici sull'odometria vengono corretti in maniera soddisfacente e che la mappa costruita utilizzando il *FastSLAM* ha una buona precisione.

6.1 Sviluppi futuri

L'applicazione della tecnica di *SLAM* mostrata in questa tesi può essere applicata a diversi campi delle robotica mobile ed in particolare quando è richiesta la completa autonomia del robot, ad esempio per l'esplorazione di pianeti oppure per l'esplorazione di luoghi non accessibili o pericolosi per l'uomo.

Il *FastSLAM* illustrato in questa tesi ed applicato sperimentalmente sul robot *XR4000* può essere esteso al caso di più robot; in una applicazione di questo tipo oltre che dotare ogni robot di completa autonomia con un algoritmo di *SLAM* e di *obstacle avoidance* occorre anche fornire ogni robot di un algoritmo di esplorazione che minimizzi il tempo necessario alla creazione combinata della mappa, nonché un algoritmo di fusione delle mappe che tenga conto eventualmente della diversità tra i sensori con cui vengono acquisiti i dati dall'ambiente (ad esempio sonar, infrarossi e laser).

Una ulteriore estensione al lavoro qui presentato è quella di aggiungere un algoritmo di pianificazione della traiettoria (*path planning*), che, una volta generata la mappa, permetta di trovare la traiettoria ottima per raggiungere più punti nella mappa.

Bibliografia

[1] H.L. Beus and S.S.H. Tiu. An improved corner detection algorithm based on chain-coded plane curves. *Pattern Recognition*, 20:291–296, 1987.

[2] D. Chetverikov and Z. Szabò. A simple and efficient algorithm for detection of high curvature points in planar curves. *23rd Workshop of the Austrian Pattern Recognition Group*, pages 175–184, 1999.

[3] W. Cochran. *Sampling Techniques*. John Wiley and Sons, third edition, 1977.

[4] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[5] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellized particle filters for dynamic bayes nets. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2000.

[6] H. Freeman and L.S. Davis. A corner finding algorithm for chain-coded curves. *IEEE Trans. on Computers*, 26:297–303, 1977.

[7] M. Galassi, J. Davis, J. Theiler, B. Gough, J. Jungman a nd M. Booth, and F. Rossi. *GNU Scientific Library - Reference Manual*. Free Software Foundation, 1.6 edition, December 2004.

[8] J.E. Gentle. Cholesky factorization. In Berlin: Springer-Verlag, editor, *Numerical Linear Algebra for Applications in Statistics*, pages 93–95, 1998.

[9] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *International Conference on Intelligent Robots and Systems*, pages 207–211, 2003.

[10] W. Madow. On the theory of systematic sampling, ii. *Annals of Mathematical Statistics*, 20:333–354, 1949.

[11] F. A. Marino. Tecniche di esplorazione e pianificazione del moto per il problema dello slam. Master's thesis, Università degli studi di Roma "Tor Vergata", 2003/2004.

[12] A. Martinelli. The odometry error of a mobile robot with a synchronous drive system. *IEEE Transactions on Robotics and Automation*, 18(3):399–405, 2002.

[13] M. Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem With Unknown Data Association*. PhD thesis, School of Computer Science Carnegie Mellon University, 2003.

[14] K. Murphy. Bayesian map learning in dynamic environments. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 1999.

[15] J.C. Nash. The cholesky decomposition. In England: Adam Hilger Bristol, editor, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*, pages 84–93, 1990.

[16] G. Randelli. Approccio genetico allo slam problem. Master's thesis, Università degli studi di Roma "Tor Vergata", 2003/2004.

[17] F. Romanelli and L. Spinello. *Guida al Nomad XR4000*, 2005.

[18] A. Rosenfeld and E. Johnston. Angle detection on digital curves. *IEEE Trans. on Computers*, pages 875–878, 1973.

[19] A. Rosenfeld and J.S. Weszka. An improved method of angle detection on digital curves. *IEEE Trans. on Computers*, 24:940–941, 1975.

[20] S. I. Roumeliotis and G. A. Bekey. Bayesian estimation and kalman filtering: A unified framework for mobile robot localization. In *Proceedings of 2000 IEEE*, 2000.

[21] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5(4):56–68, 1986.

[22] L. Spinello. Risoluzione al problema dello slam tramite filtri particellari e filtro di kalman esteso. Master's thesis, Università degli studi di Roma "Tor Vergata", 2004/2005.

[23] S. Thrun, D. Fox, and W. Burgard. Monte carlo localization with mixture proposal distribution. In *Proceedings of AAAI National Conference on Artificial Intelligence*, 2000.

[24] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, Department of Computer Science, University of North Carolina at Chapel Hill, 2004.

[25] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte. An efficient approach to the simultaneous localisation and mapping problem. In *Proceedings of 2002 IEEE*, 2002.

[26] Z.Y. Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13(2):119–15, 1994.

Elenco delle figure

Elenco delle tabelle