UNIVERSITÀ DEGLI STUDI DI ROMA
"TOR VERGATA"

FACOLTÀ DI INGEGNERIA

Tesi di Laurea Specialistica in Ingegneria dell'Automazione

Risoluzione al problema dello SLAM tramite filtri particellari e filtro di Kalman Esteso

Relatore
Ing. F. Martinelli

Laureando
Luciano Spinello

Anno Accademico 2004/2005

Indice

A mia madre.

Capitolo 1

Introduzione

Il problema dello SLAM, acronimo in lingua inglese di ***S**imultaneous **L**ocalization **A**nd **M**apping*, è uno degli argomenti più discussi e studiati nel campo della robotica, passo fondamentale per lo studio e la costruzione di un veicolo completamente capace di essere autonomo in un ambiente sconosciuto. Lo SLAM consiste nella capacità di un robot mobile di localizzare se stesso e generare una mappa in un ambiente completamente ignoto.

Questo obiettivo viene completato mediante l'utilizzo di diverse tecniche, che affrontano il problema in modo diverso. Lo SLAM viene risolto principalmente mediante l'utilizzo di filtri di Kalman Esteso; metodiche più recenti applicano tecniche derivate da algoritmi genetici [18] [11]. Un'altra tecnica risolutiva dello SLAM, innovativa ed efficiente, è costituita dal FastSLAM [13], un metodo che fonde i benefici dell'utilizzo dei filtri particellari con il filtro di Kalman Esteso. Questa tecnica costituisce lo stato dell'arte dello SLAM. Il FastSLAM risulta più efficiente in termini computazionali e di occupazione di memoria rispetto alle altre tecniche, più robusto rispetto ai rumori di misura e di moto, più efficace nel risolvere le ambiguità nelle associazioni di dati. Proprio queste particolarità lo rendono adatto ad un utilizzo pratico in ambienti reali.

In questa tesi viene proposta una implementazione del FastSLAM, sviluppata su un reale robot mobile. Il robot mobile utilizzato è il Nomad XR4000 della

Nomadic Technologies, olonomo, dotato di un sensore laser per il rilevamento dell'ambiente, e di sensori infrarosso su tutto il suo perimetro per il rilevamento di ostacoli a corto raggio. Il Nomad XR4000 è dotato di una CPU Intel Pentium III. Questo robot soffre di forti errori sistematici sul suo moto.

Un grande sforzo è stato compiuto per rendere autonomo il robot, implementando tecniche di *obstacle-avoidance*, un algoritmo di esplorazione e controllo del robot. Tutte queste funzioni sono state progettate per funzionare al meglio con le capacità computazionali del robot.

Si è inoltre estesa la teoria del FastSLAM, combinandola con il metodo *ICP SCAN-matching*, per risolvere gli errori sistematici che influenzano il corretto moto del robot mobile. Lo *SCAN-matching* è una metodica di analisi di due scansioni laser successive per stimare lo spostamento effettutato dal robot, qui applicato per tener conto degli errori sul moto. Questa tecnica ha permesso di applicare il FastSLAM con un'alta precisione senza l'utilizzo di sensori odometrici. Il robot, cioè effettua la propria localizzazione, costruisce la mappa dell'ambiente utilizzando le sole misure prelevate dal sensore laser.

L'algoritmo di esplorazione e navigazione autonoma è basato su un metodo di scomposizione a griglie dell'ambiente esplorabile, e si basa sulla scelta della traiettoria più libera da percorrere; è capace anche di selezionare la rotta, tramite l'analisi di un grafo costruito durante la navigazione, per raggiungere le celle non ancora esplorate. L'algoritmo di navigazione effettua, tramite l'utilizzo di controllori proporzionali, il calcolo delle velocità traslazionali e rotazionali di riferimento da fornire al *controller* del motore del robot.

È stata inoltre sviluppata una modlità di costruzione della mappa dell'ambiente.

Il progetto è stato prima sviluppato sulla piattaforma scientifica Matlab, passo necessario per la verifica, la comprensione e lo sviluppo della teoria, e il fondamento per l'applicazione pratica sul robot mobile. È stato scelto questo ambiente matematico come piattaforma simulativa per la sua estrema potenza e flessibi-

lità. Dopo aver eseguito i necessari test e simulazioni si è sviluppato il vero e proprio *software* per il robot, su piattaforma Linux in linguaggio C.

Per utilizzare il robot Nomad XR4000 è stato necessario procedere all'aggiornamento del sistema operativo e effettuare diversi processi di manutenzione su alcune componenti *hardware*, in quanto il robot risultava completamente non funzionante. Sul robot era montata la distribuzione RedHat Linux 5.3, con kernel 2.0.35, e si è aggiornata con una più recente RedHat 7.3, con kernel 2.4.18. Si è proceduto all'aggiornamento e alla configurazione di tutti i moduli necessari al funzionamento dei dispositivi del robot (scheda di cattura video, sistema sonar/infrarosso/*bumper*, sensore laser, controller del motore, scheda *wireless* lan), in modo da renderlo una piattaforma di sviluppo più aggiornata e, sopratttutto, più stabile. Si è montato un processore più potente (Pentium III a 500Mhz) e aumentata la RAM a 384MB. Si sono sostituite le batterie al piombo del Nomad XR4000 costruendo degli adattatori necessari al collegamento delle nuove batterie con il circuito di alimentazione.

Capitolo 2

Il problema dello SLAM

Il problema dello SLAM, acronimo in lingua inglese di ***S**imultaneous **L**ocalization **A**nd **M**apping*, è uno degli argomenti più discussi e studiati nel campo della robotica, passo fondamentale per lo studio e la costruzione di un veicolo completamente capace di essere autonomo in un ambiente sconosciuto. Lo SLAM consiste nella capacità di un robot mobile di localizzare se stesso e generare una mappa in un ambiente completamente ignoto. Questo obiettivo, a seconda dei modelli di SLAM proposti, viene raggiunto con vari livelli di precisione e diversi tempi di completamento, e anche diversi gradi di autonomia del robot.

Applicazioni di rilievo basate su questo metodo sono: esplorazioni di ambienti remoti, inaccessibili, oppure estremamente rischiosi per un intervento umano. Vari robot basati su SLAM sono stati usati per l'esplorazione remota di pianeti ove non è possibile utilizzare metodiche di posizionamento assoluto. Robot autonomi di questo tipo sono utilizzati per tecniche avanzato di sminamento [1], oppure di esplorazione di relitti sottomarini.

Il robot effettua misure, affette da errore, sul suo moto e sull'ambiente ad esso circostante, sia per effettuare una stima corretta della sua posizione all'interno dell'ambiente non noto in cui è immerso, sia per generare una mappa affidabile. Per questo lo SLAM è un requisito importante anche per il cosidetto *path planning*: costruita una mappa è infatti possibile effettuare studi su come poter

raggiungere un punto da un altro all'interno di un ambiente, per esempio con un criterio di ottimo, come quello di effettuare il percorso nel minor tempo possibile o con il minore dispendio di energia.

Il problema dello SLAM è un problema non banale: se il percorso del robot fosse conosciuto allora la creazione della mappa sarebbe un lavoro diretto, come pure se si conoscesse *a priori* la mappa da cui estrarre la stima della posizione; questa concorrenza di incertezze tra stima della posizione e stima dell'osservazione portano alla formulazione di soluzioni e metodi teorici che cercano di ovviare a questo inconveniente tramite approcci sostanzialmente diversi, di cui di seguito verranno esposte le principali caratteristiche.

2.1 Metodi di stima

La stima delle osservazioni e la stima della posa del robot sono problemi fortemente legati tra loro. Mentre il robot è in movimento esso accumula errore sulla sua posizione attuale, per cui il mondo percepito dai sensori è affetto dal proprio errore di misura e dalla incertezza della posa del robot. Oltretutto il movimento del robot è molto spesso affetto da errori sistematici[1], che si trasferiscono in modo evidente sulla costruzione della mappa stessa. In forma assolutamente generale è possibile affermare che l'errore sulla posa del robot è correlato con l'errore di costruzione della mappa. Non è quindi possibile effettuare una stima della mappa reale senza effettuare una stima della posizione del robot.

Un robot è un oggetto che effettua controlli[2] ed attua delle osservazioni dell'ambiente circostante. Sia i controlli che le osservazioni effettuate sono affette da errore. Ogni controllo o ogni osservazione associata ad un modello di errore è possibile concepirla come un vincolo probabilistico. Ad esempio il controllo vincola il robot rispetto a due pose successive, mentre l'osservazione vincola la

[1] Si definisce errore sistematico la differenza tra il valore reale (di norma sconosciuto) della grandezza in esame e il valore assunto dalla misura effettuata su di essa.

[2] Per controlli del robot si intende nel testo la coppia V, velocità traslazionale, e Ω, velocità angolare

posizione relativa del robot rispetto ad alcuni punti significativi (anche dette *feature*) dell'ambiente. Durante l'esplorazione questi vincoli aumentano tra loro e vengono utilizzati per la stima della posizione del robot e per la stima delle *feature* presenti mappa. In principio questi legami sono vincoli poco forti ma, effettuando delle osservazioni successive delle *feature*, questi incrementalmente aumentano di rigidezza, fino a che se le *feature* venissero riosservate all'infinito, la posizione del robot e la posizione dei punti nella mappa coinciderebbero con la loro posa reale, cioè sarebbero tra loro completamente correlati.

Lo *SLAM posterior* è una distribuzione probabilistica che è soluzione al problema dello SLAM per stimare tutte le possibili pose del robot e dei punti della mappa, date tutte le letture dei sensori e delle osservazioni effettuate nell'esplorazione dell'ambiente.

Lo *SLAM posterior* è definito come:

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

Nella (2.1) s_t è la posa corrente del robot; Θ è la mappa, cioè l'insieme di tutte le *feature* presenti nell'ambiente. La probabilità *posterior*[3] è condizionata da z^t,l'insieme di tutte le osservazioni dei sensori; u^t, l'insieme di tutti i controlli effettuati; n^t, le associazioni di dati che correlano le osservazioni compiute dai sensori alle *feature* in Θ. La mappa Θ di solito è rappresentata, per semplicità, da una collezione di *feature* puntuali, anche chiamati *landmark*, ma qualunque altro modello di ordine superiore è implementabile, come per esempio una mappa che includa segmenti o archi di cerchio. I *landmark* vengono estratti dalle letture dei sensori e sono dei punti facilmente riconoscibili dell'ambiente in cui è immerso il robot (come ad esempio degli spigoli o dei recessi di una parete).

La distribuzione *posterior* è conveniente non solo perché restituisce la migliore stima probabile della posa del robot e dei *landmark* ma anche perché restituisce una stima sull'incertezza di ogni quantità conosciuta attraverso la loro correlazione. Una distribuzione di possibili soluzioni risulta essere una scelta adatta in

[3]Probabilità condizionata di un evento in cui si tenga conto di dati empirici

ambienti rumorosi; inoltre, l'incertezza può essere utilizzata per valutare le informazioni relative date dai differenti componenti della soluzione: una parte della mappa può essere ben conosciuta, mentre, un'altra può essere affetta da una grande incertezza.

Il filtro di *Bayes* può essere utilizzato per trovare la soluzione dello *SLAM posterior* (2.1) al tempo t, dato il *posterior* dell'istante $t-1$:

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

L'integrale mostrato nella (2.2) non è risolvibile in forma chiusa, se non impiegando delle assunzioni sulla tipologia di distribuzione dello *SLAM posterior*. Tecniche statistiche di stima quali il filtro di *Kalman* e il filtro particellare sono nient'altro che approssimazioni del filtro di *Bayes*.

2.2 Definizione e notazioni dello SLAM

Prima di addentrarsi nella spiegazione dei metodi di SLAM è utile, per la comprensione del testo, soffermarsi sulla spiegazione dello *SLAM posterior* e su alcuni tipi di notazioni. La posa al tempo t del robot è denominata s_t. In robot che operano in mondi bidimensionali essa è costituita da una terna di coordinate che comprende la coppia di coordinate cartesiane x, y, che definiscono la posizione assoluta sul piano di lavoro, e l'orientamento α, o più brevemente l'*heading*, anch'esso rispetto a un riferimento assoluto, si veda la figura 2.1. La traiettoria completa, o *path*, è definita come una collezione di pose del robot ad ogni *time step*, ed è denominata s^t:

$$s^t = \{s_1, s_2, s_3, ..., s_t\} \quad (2.3)$$

In questa tesi viene assunto che la mappa sia costituita da N *landmark* puntuali immobili, e che il robot navighi in un ambiente 2D. Tali tipi di landmark

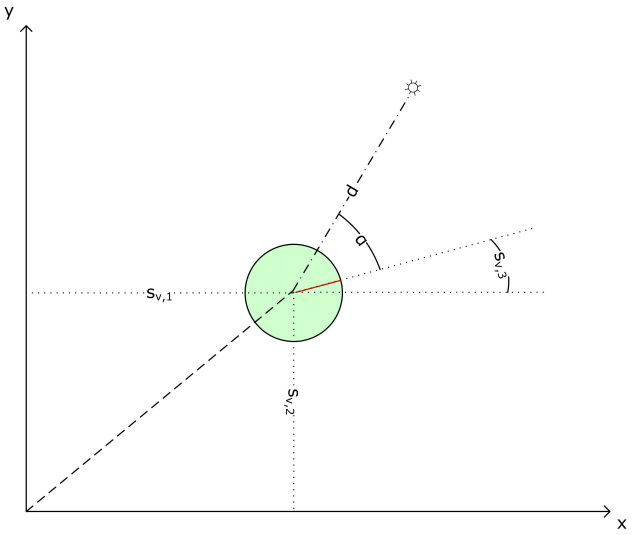


Figura 2.1: *Definizione di un robot in un ambiente planare.*

vengono estratti dalle osservazioni e sono genericamente utilizzati come una rappresentazione sintetica della posizione di *feature* geometriche presenti nell'ambiente, come recessi, angoli, zone peculiarmente evidenti ai sensori. L'insieme Θ, che definisce la mappa, è composto da N elementi che esprimono la posizione reale di ogni *landmark*.

$$\Theta = \{\theta_1, \theta_2, \theta_3, ..., \theta_N\} \tag{2.4}$$

Mentre il robot esplora l'ambiente esso riceve informazioni sui suoi spostamenti. Queste informazioni vengono restituite, come nel nostro caso, da *encoders* montati sull'asse di rotazione delle ruote oppure, da unità di navigazione inerziale, o più semplicemente dai comandi di controllo eseguiti dal robot. Qualunque sia l'origine di queste informazioni sul proprio moto esse possono essere dette, per semplicità, controlli. Analogamente alle altre notazioni definite, si denota il controllo al tempo t come u_t; mentre l'insieme di tutti i controlli eseguiti dal robot:

$$u^t = \{u_1, u_2, u_3, ..., u_t\} \tag{2.5}$$

Nella formulazione comune in un problema di SLAM, le osservazioni sulla posizione dei *landmark* vengono definite con una coppia di variabili costituite da

distanza e orientamento relative alla posa del robot; l'osservazione al tempo t è denominata z_t. Il set di tutte le osservazioni è definito da:

$$z^t = \{z_1, z_2, z_3, ..., z_t\} \tag{2.6}$$

Si assume che ogni osservazione restituisce informazione sulla posizione di un certo landmark θ_n relativa alla posa s_t. Si indicherà con n la variabile che identifica il *landmark* che è stato osservato. In realtà, questa identità non è mai completamente certa, in quanto potrebbe accadere che un *landmark* venga scambiato per un altro. L'identità del landmark corrispondente all'osservazione z_t, verrà scritta come n_t, dove $n_t \in (1, 2, 3, ..., N)$; per chiarezza, $n_4 = 9$ vuol dire che al tempo $t = 4$ è stato osservato il *landmark* numero 9. Genericamente n_t è chiamata anche associazione di dati o corrispondenza. L'insieme di tutte le associazioni di dati è denotata come:

$$n^t = \{n_1, n_2, n_3, ..., n_t\} \tag{2.7}$$

Per semplicità di notazione, viene assunto che il robot riceve una misura z_t, esegue un solo controllo u_t per istante di tempo. Lo scopo principale dello SLAM è quello di restituire la migliore stima della posa s_t e della mappa Θ, dato l'insieme rumoroso di controlli u^t e di osservazioni z^t; in termini probabilistici vuol dire restituire la probabilità di:

$$p(s_t, \Theta | z^t, u^t) \tag{2.8}$$

cioè trovare la soluzione allo *SLAM posterior*. Se è dato anche l'insieme delle associazioni di dati n^t, si può completare la notazione con:

$$p(s_t, \Theta | z^t, u^t, n^t) \tag{2.9}$$

2.3 Lo SLAM e le catene di Markov

Lo SLAM può essere descritto come una catena di Markov (figura 2.2). La posa corrente s_t può essere descritta da una funzione probabilistica della posa al tempo

precedente $t-1$ e del controllo u_t eseguito dal robot. Questa funzione è definita essere il modello di moto, *motion model*, e descrive come il controllo influenzi il movimento del robot; oltretutto il *motion model* descrive come il rumore nel controllo inserisce incertezza nella stima della posa. Il *motion model* è descritto dalla:

$$p(s_t|s_{t-1}, u_t) \tag{2.10}$$

Anche le osservazioni effettuate dai sensori sono governate da una funzione probabilistica. L'osservazione z_t è funzione del landmark θ_n e della posa del robot s_t. Questa funzione è comunemente definita come modello di misurazione, o *measurement model*, e descrive la fisica e il modello di errore del sensore del robot. Essa è definita dalla seguente:

$$p(z_t|s_t, \Theta, n_t) \tag{2.11}$$

Utilizzando il *measurement model* e il *motion model* lo *SLAM posterior* al tempo t può essere calcolato in maniera ricorsiva come funzione del *posterior* al tempo $t-1$. Questa regola di aggiornamento ricorsivo definisce il filtro di Bayes per lo SLAM, che è il principio di funzionamento di base della maggior parte delle tecniche di SLAM.

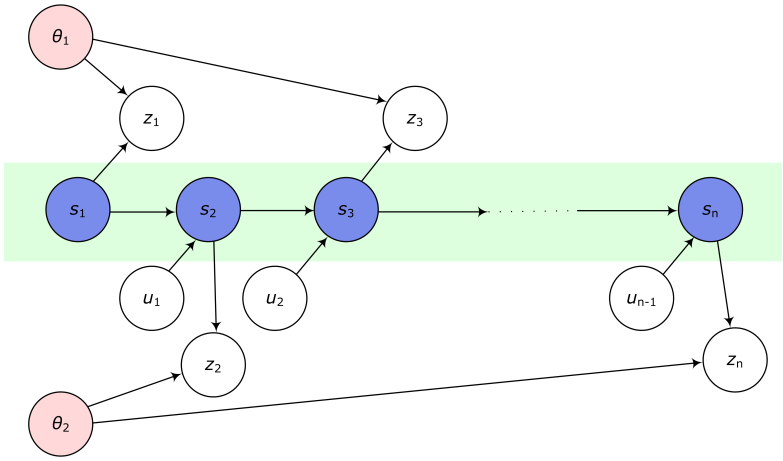


Figura 2.2: *Descrizione dello SLAM come una catena di Markov.*

2.4 Il filtro di Bayes

Il filtro di Bayes può essere dedotto dallo *SLAM posterior* come segue. Lo *SLAM posterior*, descritto nella (2.9), è sviluppato utilizzando la regola di Bayes:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta \, p(z_t | s_t, \Theta, z^{t-1}, u^t, n^t) \, p(s_t, \Theta | z^{t-1}, u^t, n^t) \tag{2.12}$$

Il denominatore della regola di Bayes è una costante normalizzante ed è descritto da η nella precedente (2.12). Sfruttando la peculiarità che z_t è solamente funzione della posa del robot s_t, della mappa Θ, e dell'ultima associazione di dati n_t[4], lo SLAM posterior diviene:

$$p(s_t, \Theta | z^t, u^t, n^t) \; = \eta \, p(z_t | s_t, \Theta, n_t) \, p(s_t, \Theta | z^{t-1}, u^t, n^t) \tag{2.13}$$

Si può quindi utilizzare il Teorema della Probabilità Totale per condizionare il termine più a destra della (2.12) con la posa del robot al tempo $t-1$:

$$\begin{aligned} & p(s_t, \Theta | z^t, u^t, n^t) \; = \\ & \eta \, p(z_t | s_t, \Theta, n_t) \int p(s_t, \Theta | s_{t-1}, z^{t-1}, u^t, n^t) \, p(s_{t-1} | z^{t-1}, u^t, n^t) ds_{t-1} \end{aligned} \tag{2.14}$$

Il termine a sinistra presente dentro l'integrale della (2.14) può essere esteso utilizzando la definizione di probabilità condizionale:

$$\begin{aligned} p(s_t, \Theta | z^t, u^t, n^t) \; &= \; \eta \, p(z_t | s_t, \Theta, n_t) \cdot \\ \cdot \int p(s_t | \Theta, s_{t-1}, z^{t-1}, u^t, n^t) \, p(\Theta | s_{t-1}, z^{t-1}, u^t, n^t) \, p(s_{t-1} | z^{t-1}, u^t, n^t) \, ds_{t-1} & \end{aligned} \tag{2.15}$$

Il primo termine dentro l'integrale può essere semplificato notando che s_t è solo funzione di s_{t-1} e u_t[5]:

$$\begin{aligned} p(s_t, \Theta | z^t, u^t, n^t) \; &= \; \eta \, p(z_t | s_t, \Theta, n_t) \cdot \\ \cdot \int p(s_t | s_{t-1}, u^t \,) \, p(\Theta | s_{t-1}, z^{t-1}, u^t, n^t) \, p(s_{t-1} | z^{t-1}, u^t, n^t) \, ds_{t-1} & \end{aligned} \tag{2.16}$$

[4] Si veda il modello di misura precedentemente descritto.

[5] Si veda il modello di moto precedentemente descritto.

In questo modo è possibile fondere i termini più a destra presenti nell'integrale:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta\, p(z_t | s_t, \Theta, n_t) \int p(s_t | s_{t-1}, u_t)\, p(s_{t-1}, \Theta | z^{t-1}, u^t, n^t) ds_{t-1} \quad (2.17)$$

Considerato che la posa s_t e l'associazione di dati n_t senza l'ultima osservazione z_t non contiene nuove informazioni su s_{t-1} o Θ, è possibile elidere il termine più a destra dell'integrale. Il risultato è una formula ricorsiva per calcolare lo *SLAM posterior* al tempo t dato quello al tempo $t-1$, il *motion model* $p(s_t|s_{t-1}, u_t)$ e il *measurement model* $p(z_t|s_t, \Theta, n_t)$:

$$p(s_t, \Theta | z^t, u^t, n^t) = \eta\, p(z_t | s_t, \Theta, n_t) \int p(s_t | s_{t-1}, u_t)\, p(s_{t-1}, \Theta | z^{t-1}, u^{t-1}, n^{t-1}) ds_{t-1} \quad (2.18)$$

2.5 Associazione di dati

Le esplicite associazioni di dati n^t, nella realtà dello SLAM, raramente sono osservabili. Se, comunque, l'incertezza della posizione dei *landmark* è bassa, una semplice condizione euristica è sufficiente per determinare la corretta associazione di dati. In particolare è un approccio usuale a questo problema l'utilizzo di una regola di massima verosimiglianza per assegnare la corretta osservazione. Questo metodo semplicemente assegna ogni osservazione al più probabile *landmark* che l'ha generata. Se non è possibile associare tale osservazione a nessuno di essi, viene utilizzata per assegnare un nuovo *landmark* nell'ambiente.

Se si considera il caso classico dell'EKF[6], la probabilità dell'osservazione può essere scritta come funzione della differenza tra la osservazione z_t e l'osservazione attesa $\hat{z}_{n_t}$, chiamata anche *innovazione*.

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

[6] Extended Kalman Filter, spiegato nel capitolo 3.

Nella (2.19), Z_t è la matrice di covarianza dell'innovazione al tempo t. Questa associazione di dati euristica è spesso riformulata come *negative log likelihood*:

$$\hat{n}_t = \arg\min_{n_t} \left[\ln |Z| + (z - \hat{z})^T Z^{-1} (z - \hat{z})\right] \tag{2.19}$$

Il secondo termine dell'equazione (2.19) definisce la distanza di Mahalanobis, una metrica normalizzata sulle covarianze dell'osservazione e sulla stima della posizione del *landmark*. Per questo motivo l'associazione di dati che utilizza questa tecnica viene chiamata *nearest neighbor data association*, o *nearest neighbor gating*. L'associazione di dati a massima verosimiglianza ha un buon funzionamento quando l'associazione di dati corretta è significativamente più probabile delle associazioni di dati errate. Se esiste una forte incertezza nel posizionamento dei *landmark* questa tecnica assegnerà a più di una associazione di dati una probabilità alta. L'ambiguità nella corretta associazione di dati è frequente se il robot ha sensori molto rumorosi.

Oltre al metodo *nearest neighbor gating* ne esistono altri più sofisticati, che ottengono una precisione tale da evitare problematici errori nel processo di SLAM[7].

[7] L'EKF SLAM, per esempio, tende a divergere se vengono incluse osservazioni frutto di associazioni di dati errate.

Capitolo 3

SLAM EKF

Nel 1960 Kalman propose un nuovo metodo basato su una soluzione ricorsiva per la risoluzione a tempo discreto al problema del *data filtering* lineare. Il filtro di Kalman è un insieme di equazioni che provvedono, in maniera computazionalmente efficiente, ad effettuare la stima dello stato di un processo minimizzando la media dell'errore quadratico. Viene di seguito spiegato in grandi linee il funzionamento del filtro di Kalman discreto, quindi il filtro di Kalman esteso nel discreto per la soluzione al problema dello SLAM. Lo SLAM basato su EKF viene descritto per completezza, poiché è base di comparazione di tutte le tecniche di SLAM, e del FastSLAM in particolare.

3.1 Il filtro di Kalman discreto

Il filtro di Kalman discreto [24] ha come obiettivo la stima dello stato $x \in \mathbb{R}^n$ di un processo governato da un'equazione alle differenze lineare stocastica a tempo discreto:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{3.1}$$

con la misura $z \in \mathbb{R}^m$ data dalla:

$$z_k = Hx_k + v_k \tag{3.2}$$

dove le variabili w e v rappresentano rispettivamente il rumore del processo e della misura. Questi sono assunti essere rumore bianco, tra loro indipendenti e con distribuzione di probabilità normale:

$$p(w) \sim N(0, Q) \tag{3.3}$$

$$p(v) \sim N(0, R) \tag{3.4}$$

Le matrici Q ed R rappresentano le matrici di covarianza rispettivamente del rumore di processo e del rumore di misura.

Si definisce $\hat{x}_k^- \in \mathbb{R}^n$ lo stato stimato *a priori* al tempo k, data la conoscenza del processo al passo $k-1$, e $\hat{x}_k \in \mathbb{R}^n$ lo stato stimato *a posteriori* all'istante k, date le misure z_k. E' possibile quindi esaminare gli errori della stima *a priori* (3.5) e *a posteriori* (3.6):

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{3.5}$$

$$e_k \equiv x_k - \hat{x}_k \tag{3.6}$$

come pure la matrice di covarianza *a priori* (3.7) e a *a posteriori* (3.8)

$$P_k^- = E[e_k^- e_k^{-T}] \tag{3.7}$$

$$P_k = E[e_k e_k^T] \tag{3.8}$$

E' possibile ricavare l'equazione del filtro di Kalman ricercando una equazione che calcoli la stima *a posteriori* dello stato $\hat{x}_k$ utilizzando una combinazione lineare della stima *a priori* $\hat{x}_k^-$ e una differenza pesata tra la misura attuale z_k con una misura predetta $H\hat{x}_k^-$.

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{3.9}$$

La differenza $z_k - H\hat{x}_k^-$ presente nella (3.9) è chiamata *innovazione* della misura, o più brevemente *residuo*. Esso riflette quanto è stata accurata la misura predetta $H\hat{x}_k^-$ rispetto alla reale misura effettuata z_k. La matrice K, calcolata attraverso diversi passaggi algebrici, rappresenta il guadagno che minimizza l'errore della stima *a posteriori* (3.6).

Il filtro di Kalman discreto stima gli stati del processo utilizzando una sorta di controllo a *feedback*: il filtro stima lo stato in un certo istante e riceve un *feedback* tramite le misure rumorose effettuate. Per questo motivo è infatti possibile separare idealmente le equazioni del filtro di Kalman in due gruppi funzionali: equazioni di aggiornamento dello stato *a priori* e equazioni di aggiornamento tramite misure. Le equazioni di aggiornamento dello stato *a priori* sono responsabili della predizione un passo avanti dello stato corrente e della matrice di covarianza degli errori di stima, per produrre una stima *a priori* degli stati per il prossimo istante di tempo. Le equazioni di aggiornamento tramite misure, invece, sono responsabili del *feedback*, cioè di incorporare le misure nello stato stimato *a priori* per produrre una stima corretta dello stato *a posteriori*.

Equazioni di aggiornamento dello stato *a priori* del filtro di Kalman discreto:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.10}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3.11}$$

Equazioni di aggiornamento tramite misure del filtro di Kalman discreto:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{3.12}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.13}$$

$$P_k = (I - K_k H)P_k^- \tag{3.14}$$

Le equazioni di aggiornamento dello stato *a priori* si possono pensare come equazioni di un predittore discreto, mentre le equazioni di aggiornamento tramite misure, come equazioni di un correttore dinamico che rifiniscono la predizione effettuata.

3.2 Il filtro di Kalman Esteso

Nella sezione precedente, 3.1, si è descritto come il filtro di Kalman discreto fosse applicabile a processi stocastici a tempo discreto, in questo paragrafo invece ci

soffermeremo sulla possibilità di utilizzare il filtro di Kalman per un processo non lineare stocastico e/o per un processo che ha relazioni non lineari tra le misure e il processo. E' necessario effettuare delle modifiche sostanziali all'originale filtro di Kalman per poterlo applicare. Il filtro di Kalman esteso [24], o più brevemente *EKF* (Extended Kalman Filter), è un filtro di Kalman discreto linearizzato sulla corrente media e covarianza.

Similmente alle approssimazioni in serie di Taylor, è possibile linearizzare il processo di stima sulla stima corrente utilizzando le derivate parziali del processo e della dinamica delle misure, anche se esse sono legate da relazioni non lineari.

La stima dello stato $x \in \mathbb{R}^{\mathrm{n}}$ rimane l'obiettivo dell'EKF ma adesso il processo è governato da una equazione alle differenze stocastica non lineare:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \tag{3.15}$$

con la misura $z \in \mathbb{R}^{\mathrm{m}}$:

$$z_k = h(x_k, v_k) \tag{3.16}$$

dove w_k e v_k rappresentano il rumore sul processo e sulle misure. La funzione alle differenze non lineare in (3.15) lega lo stato corrente con quello al tempo $k-1$, relazionato al controllo al tempo $k-1$ e al rumore al tempo $k-1$. La funzione non lineare h della (3.16) invece lega la misura al tempo k con lo stato al tempo k.

Visto che nella pratica non è possibile quantificare le quantità w_k e v_k per ogni intervallo di tempo, possiamo approssimare la (3.15) e la (3.16) con $w_k = 0$ e $v_k = 0$, cioè considerandole prive di rumore

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.17}$$

$$\tilde{z}_k = h(\hat{x}_k, 0) \tag{3.18}$$

Nella (3.17) la $\hat{x}_{k-1}$ rappresenta lo stato stimato *a posteriori* al tempo k. E' molto importante sottolineare inoltre che le distribuzioni delle variabili casuali non risultano più coincidere con le distribuzioni normali dopo le trasformazioni non lineari, questo è difatti un problema strutturale del filtro di Kalman esteso.

Esistono alcune estensioni dell'EKF in cui vengono preservate le distribuzioni normali attraverso le trasformazioni non lineari [9].

Per stimare un processo non lineare bisogna iniziare con delle equazioni che linearizzano una stima sulle (3.17) e (3.18),

$$x_k = \tilde{x_k} + A_k(x_{k-1} - \hat{x}_{k-1}) + W_k w_{k-1} \tag{3.19}$$

$$z_k = \tilde{z_k} + H_k(x_k - \tilde{x}_k) + V_k v_k \tag{3.20}$$

dove:

- x_k e z_k sono i vettori dello stato e delle misurazioni correnti.
- $\tilde{x}_k$ e $\tilde{z}_k$ sono i vettori dello stato approssimato e delle misurazioni approssimate correnti, descritti nella (3.17) e (3.18).
- $\hat{x}_k$ è la stima *a posteriori* dello stato al tempo k.
- w_k e v_k rappresentano il rumore di processo e di misura.
- A_k è lo jacobiano di f rispetto a x calcolato al tempo k, i cui elementi vengono così calcolati:

$$A_k^{[i,j]} = \frac{\delta f_{[i]}}{\delta x_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.21}$$

- W_k è lo jacobiano di f rispetto a w calcolato al tempo k, i cui elementi vengono così calcolati:

$$W_k^{[i,j]} = \frac{\delta f_{[i]}}{\delta w_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.22}$$

- H_k è lo jacobiano di h rispetto a x calcolato al tempo k, i cui elementi vengono così calcolati:

$$H_k^{[i,j]} = \frac{\delta h_{[i]}}{\delta x_{[j]}}(\tilde{x}_k, 0) \tag{3.23}$$

- V_k è lo jacobiano di h rispetto a v calcolato al tempo k, i cui elementi vengono così calcolati:

$$V_k^{[i,j]} = \frac{\delta h_{[i]}}{\delta v_{[j]}}(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.24}$$

Si definiscono gli errori di predizione come

$$\tilde{e}_{x_k} \equiv x_k - \tilde{x}_k \tag{3.25}$$

e il *residuo* dell'errore di misura

$$\tilde{e}_{z_k} \equiv z_k - \tilde{z}_k \tag{3.26}$$

Bisogna tenere a mente che non si conosce il reale valore del vettore degli stati x_k nella (3.25), ma si può conoscere la misura z_k della (3.26) che è quella utilizzata per stimare x_k. Da queste considerazioni possiamo scrivere le equazioni che definiscono la dinamica di errore come:

$$\tilde{e}_{x_k} \approx A_k(x_{k-1} - \hat{x}_{k-1}) + \varepsilon_k \tag{3.27}$$

$$\tilde{e}_{z_k} \approx H_k\tilde{e}_{x_k} + \eta_k \tag{3.28}$$

dove ε_k ed η_k sono nuove variabili casuali indipendenti a media nulla, con le rispettive matrici di covarianza calcolate come WQW^T e VRV^T. E' importante notare come la (3.27) e la (3.28) sono equazioni lineari e sono simili alle (3.1) e (3.2). Usando il residuo di misurazione della (3.26) e un secondo ipotetico filtro di Kalman per stimare $\tilde{e}_{x_k}$, è possibile ottenere $\hat{e}_k$, utilizzabile poi per calcolare la stima dello stato *a posteriori* per l'originale processo non lineare:

$$\hat{x}_k = \tilde{x}_k + \hat{e}_k \tag{3.29}$$

L'equazione del filtro di Kalman utilizzata per stimare $\hat{e}_k$ risulta:

$$\hat{e}_k = K_k\tilde{e}_{z_k} \tag{3.30}$$

Sostituendo la (3.30) nella (3.29) e considerando la (3.26), la (3.29) diventa:

$$\hat{x}_k = \tilde{x}_k + K_k(z_k - \tilde{z}_k) \tag{3.31}$$

In questo modo possiamo utilizzare la (3.31) come aggiornamento tramite misure nell'EKF, con $\tilde{x}_k$ e $\tilde{z}_k$ che provengono dalle (3.17) e (3.18) e il guadagno K_k calcolato con le stesse modalità del filtro di Kalman discreto.

Analogamente è possibile ricondurci all'analisi fatta nella sezione 3.1 delle equazioni di aggiornamento dello stato *a priori* e delle equazioni di aggiornamento tramite misure. Le equazioni di aggiornamento dello stato *a priori* per un EKF sono costituite dalle:

$$\hat{x}_k^- \equiv \tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.32}$$

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \tag{3.33}$$

che proiettano lo stato dal tempo $k-1$ al tempo k. La matrice Q_k, ha il pedice k per sottolineare il fatto che essa è libera di variare per ogni istante, ma di solito viene mantenuta costante.

Le equazioni di aggiornamento tramite misure per un EKF sono costituite dalle:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T) \tag{3.34}$$

$$P_k = (I - K_k H_k) P_k^- \tag{3.35}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - h(\hat{x}_k^-, 0)) \tag{3.36}$$

che correggono, esattamente come le (3.14), lo stato e la covarianza stimata con le misure z_k. La matrice R_k, ha il pedice k per sottolineare il fatto che essa è libera di variare per ogni istante, ma di solito viene mantenuta costante.

Nella (3.35) è importante notare che la matrice H_k modula significativamente il valore del guadagno K_k, amplificando soltanto le componenti rilevanti del processo di misura. Se non c'è corrispondenza uno ad uno tra z_k e lo stato attraverso h, lo jacobiano H_k influenza il guadagno di Kalman K_k incrementando soltanto la porzione di residuo $z_k - h(\hat{x}_k^-, 0)$ che interessa lo stato. Se non esiste mai corrispondenza uno ad uno tra le misure z_k e lo stato, attraverso h, allora il filtro diverge a causa della non osservabilità del processo.

3.2.1 L'utilizzo dell'EKF nel problema dello SLAM

L'approccio dominante nell'utilizzo del filtro di Kalman esteso per il problema dello SLAM venne proposto per la prima volta da R. C. Smith e P. Cheeseman [23], e poi la prima volta applicato in un sistema reale da P. Moutarlier e R. Chatila [14]. Il filtro di Kalman esteso viene applicato per la risoluzione dello *SLAM posterior* per la stima della posa del robot e di tutte le posizioni delle *feature* della mappa. Lo *SLAM posterior* viene approssimato con una distribuzione di probabilità gaussiana su tutte le posizioni delle *feature* e la posa del robot. Lo stato complessivo è quindi costituito dalla posa del robot e dalla posa delle *feature* individuate. Gli elementi presenti fuori della diagonale della distribuzione di probabilità gaussiana rappresentano le correlazioni tra le varie variabili di stato, per questo motivo l'EKF può essere un metodo che ben rappresenta il problema dello SLAM. Questo metodo presenta però dei problemi quali la complessità computazionale e l'estrema sensibilità nelle associazioni di dati.

Problemi dell'EKF applicato allo SLAM

Il primo problema del metodo che si basa sull'EKF per la risoluzione dello SLAM risulta risiedere nella complessità computazionale; sia l'occupazione di memoria sia il tempo di computazione aumentano in maniera quadratica con il numero di *landmark* trovati nell'ambiente. A meno di non aumentare le risorse computazionali a dismisura, l'EKF limita in un ambiente circoscritto l'utilizzo di questa tecnica di SLAM. La complessità quadratica è una conseguenza della rappresentazione della gaussiana implementata dall'EKF. L'incertezza dello *SLAM posterior* è rappresentata da una matrice di covarianza di grandezza $2N + 3$, in un mondo bidimensionale, dove N sono il numero di *landmark* e 3 gli stati che descrivono la posa del robot. Risulta facile analizzare che il contributo di memoria cresce come N^2. Gli elementi di questa matrice di covarianza rappresentano le correlazioni tra le diverse variabili di stato, per cui ogni volta che viene aggiunto un *landmark* esso influenza tutte le altre variabili di stato, per cui è necessario riprocessare tutti gli altri elementi della matrice di covarianza con un costo com-

putazionale decisamente alto, anch'esso N^2. Vista l'estrema richiesta di risorse l'EKF viene utilizzato raramente e vengono in realtà utilizzate delle sue modifiche più efficienti e flessibili nell'aggiornamento come il CEKF.

Il secondo problema di cui è affetto il filtro EKF è costituito dall'associazione di dati, cioè nel relazionare le osservazioni effettuate con le corrette *feature* nella mappa, il cosidetto *mapping* tra osservazione e *landmark*. L'associazione dei dati è un processo complesso e molto sensibile che può generare gravi errori sul posizionamento della posa del robot e dei *landmark* della mappa. L'approccio standard per l'associazione di dati nell'EKF avviene assegnando ogni osservazione a un *landmark* utilizzando una regola di massima verosimiglianza, cioè ogni osservazione è associata al *landmark* che più probabilmente l'ha generata. Se questa probabilità risulta più bassa di una certa soglia allora viene creato un nuovo landmark. Una falla dell'EKF è che non esiste nessun meccanismo per rappresentare l'incertezza nell'associazione di dati e l'effetto di incorporare nello stato un nuovo *landmark*, causato da un'associazione errata, non può venire corretto. Se un gran numero di letture vengono incorporate in modo sbagliato il filtro EKF divergerà. L'indecisione nell'associazione di dati può venire in qualche modo migliorata effettuando più osservazioni simultaneamente, ma non risolve di certo il problema sostanziale. Oltretutto l'associazione corretta di una certa osservazione non è sempre la più probabile se considerata per la prima volta, più osservazioni future sono necessarie per avere un buon livello di informazione su cui basare l'identificazione della corretta associazione di dati.

Un altro problema dell'EKF riguarda la sparsità strutturale della matrice di covarianza degli stati stimati. I controlli prodotti dal robot e le osservazioni vincolano soltanto un piccolo sottoinsieme di tutto lo stato in un certo istante k. Altre peculiarità sono che due *landmark* lontani sono poco correlati e ancora, coppie di *landmark* lontani sono correlati similmente. Fortunatamente esistono diversi studi che sfruttano queste peculiarità per produrre versioni di EKF più efficienti, ma non risolvono l'ambiguità delle associazioni di dati.

Descrizione analitica dell'EKF applicato allo SLAM

Come precedentemente spiegato, lo *SLAM posterior*, descritto nella (2.2), non può venir risolto in forma chiusa. L'utilizzo di un filtro EKF è un metodo comune per far fronte a questa difficoltà e stimare lo *SLAM posterior*. Il filtro EKF rappresenta lo *SLAM posterior* come una distribuzione di probabilità multivariata di gauss di grandi dimensioni parametrizzata dalla media μ e dalla matrice di covarianza Σ_t. La media descrive la più possibile posa del robot e dei landmark, mentre la matrice di covarianza codifica le correlazioni delle coppie delle variabili di stato. Lo stato è composto dalla posa del robot e dalla posa dei *landmark*.

$$
\begin{aligned}
p(s^t, \Theta | u^t, z^t, n^t) &= N(x_t; \mu_t, \Sigma_t) \\
x_t &= \{s_t, \theta_{1t}, \theta_{2t}, ..., \theta_{Nt}\} \\
\mu_t &= \{\mu_{s_t}, \mu_{\theta_{1t}}, \mu_{\theta_{2t}}, ..., \mu_{\theta_{Nt}}\} \\
\Sigma &= \begin{bmatrix}
\Sigma_{s_t} & \Sigma_{s_t,\theta_1} & \cdots & \Sigma_{s_t,\theta_N} \\
\Sigma_{\theta_1,s_t} & \Sigma_{\theta_1} & \Sigma_{\theta_1,\theta_2} & \vdots \\
\vdots & \Sigma_{\theta_2,\theta_1} & \ddots & \vdots \\
\Sigma_{\theta_N,s_t} & \cdots & \cdots & \Sigma_{\theta_N}
\end{bmatrix}
\end{aligned}
$$

Come già anticipato per robot che si muovono in un piano, il vettore μ_t ha dimensioni $2N + 3$, dove N è il numero di *landmark*. Tre dimensioni sono utilizzate per rappresentare la posa del robot, e due per ogni landmark. La matrice di covarianza ha quindi dimensioni $2N+3 \times 2N+3$. Per questo motivo il numero di parametri necessari a descrivere un EKF è il quadrato del numero di *landmark* presenti nella mappa. Il filtro di Kalman è uno stimatore ottimo per sistemi lineari con rumore gaussiano, l'EKF, come descritto, è un estensione del filtro di Kalman classico per sistemi non lineari. Per essere efficace su questi tipi di processi esso linearizza il *motion model* e il *measurement model* attorno allo stato stimato più probabile del sistema. Questa risulta essere una buona approssimazione se i veri modelli sono approssimativamente lineari e se il passo di campionamento del filtro è piccolo.

Capitolo 4

Il FastSLAM

In questo capitolo verrà presentato un diverso approccio per la soluzione del problema dello SLAM. Questo metodo risulta decisamente innovativo e allo stesso tempo maturo rispetto al classico metodo del filtro di Kalman esteso. Il FastSLAM è stato sviluppato alla Stanford University e successivamente perfezionato da M.Montemerlo [13]. Esso è frutto di nuovi studi sui filtri particellari, riprendendo anche alcune caratteristiche importanti dal metodo basato sull'EKF.

Ogni controllo o osservazione effettuata dal robot vincola poche variabili di stato. I controlli effettuati dal robot vincolano probabilisticamente la posa precedente a quella attuale, mentre le osservazioni vincolano la posizione del *landmark* relativa a quella del robot. Solo dopo un gran numero di questi vincoli probabilistici la mappa diventa completamente correlata e lo SLAM riesce ad essere risolto a dovere. Lo SLAM basato su EKF non sfrutta totalmente questo tipo di sparsità nel tempo bensì ne risulta sofferente.

Il FastSLAM, che viene di seguito presentato, è un approccio alternativo ed efficiente al problema dello SLAM ed è basato su un filtro particellare. Esso sfrutta la sparsità strutturale del problema dello SLAM per fattorizzare lo *SLAM posterior* in un prodotto di stime di bassa dimensione. Il risultato è un algoritmo efficace anche in mappe molto grandi e robusto alle ambiguità sulle associazioni di dati.

4.1 Il filtro particellare Rao-Blackwellized

Il filtro di Kalman e il filtro di Kalman esteso rappresentano le distribuzioni di probabilità utilizzando un modello parametrizzato, una distribuzione multivariata di gauss; il filtro particellare, invece, rappresenta una distribuzione attraverso un insieme finito di stati campionati, anche detti *particelle*. Le particelle sono distribuite nello spazio in maniera non uniforme: regioni in cui esiste un'alta probabilità contengono un'alta densità di particelle, regioni con bassa probabilità ne contengono poche o addirittura nessuna. Questa struttura non parametrizzata può modellare, a patto di utilizzare un ragionevole numero di particelle, una qualunque distribuzione multimodale di arbitraria complessità. Se si utilizzasse un numero infinito di particelle si potrebbe ricostruire esattamente una vera distribuzione, per cui l'equazione di aggiornamento del filtro di Bayes potrebbe essere calcolata semplicemente con una procedura di *resampling*.

Diverse sono le applicazioni del filtro particellare nel campo della stima nella robotica mobile, una delle più note è costituita dalla *Montecarlo localization*, più brevemente MCL. Data una mappa nota a priori, l'incertezza della posa del robot è rappresentata da un insieme di particelle distribuite all'interno di essa. Il robot, nel caso più significativo, non è a conoscenza della sua posizione iniziale, e l'incertezza è costituita dalla distribuzione uniforme delle particelle dentro la mappa. Con il passare del tempo, cioè dopo aver assunto e incorporato un certo numero di controlli ed osservazioni, il *posterior* è arrivato a convergere a una distribuzione unimodale[1], quindi il robot è riuscito a localizzare con certezza la propria posizione. Proprio per la sua costituzione non strutturata, il filtro particellare ha capacità di essere performante a dinamiche di moto e di misura non lineari, ciò lo rende particolarmente robusto ed adatto al funzionamento nel mondo reale e quindi alla risoluzione del problema dello SLAM.

I filtri particellari standard sono dimensionati per risolvere problemi di bassa dimensionalità, fatto particolarmente non consona alle esigenze della risoluzione dello SLAM che può avere dimensioni enormi, che crescono al crescere della

[1]Una distribuzione di dati statistici è detta unimodale se ha una sola moda.

mappa. Questo problema può essere superato fattorizzando il problema dello SLAM in diversi sottoproblemi di bassa dimensione, cioè in un insieme di problemi indipendenti di stima dei *landmark* condizionati alla stima del percorso del robot. Il percorso del robot a posteriori ha dimensionalità bassa e può essere stimato efficientemente con un filtro a particelle. Il FastSLAM è una valida applicazione di un filtro particellare di Rao-Blackwellized [5].

4.2 Lo SLAM posterior nel FastSLAM

Il metodo dell'EKF, come pure la maggioranza degli altri metodi di SLAM, sono basati sulla stima del *posterior* sulla mappa e sulla posa del robot:

$$p(s_t, \Theta | z^t, u^t, n^t) \tag{4.1}$$

Il FastSLAM, tuttavia, stima il *posterior* sulla mappa e sul percorso, cioè:

$$p(s^t, \Theta | z^t, u^t, n^t) \tag{4.2}$$

Questa sottile differenza permette di fattorizzare lo *SLAM posterior* in un prodotto di termini meno complessi. Per arrivare a una tale conclusione bisogna effettuare alcune considerazioni. Si immagini un robot, figura 4.1, che effettua un percorso e che osservi in tre momenti diversi, tre *landmark* differenti. Se il reale percorso del robot fosse conosciuto, la posizione di ogni *landmark* sarebbe indipendente dagli altri. Questa indipendenza ha importanti conseguenze: data la conoscenza del reale percorso del robot, l'osservazione di un *landmark* non restituisce alcuna informazione sulla posizione degli altri *landmark* presenti nella mappa. Se si riesce ad ottenere il reale percorso del robot possiamo stimare la posizione di ogni *landmark* come una quantità indipendente. Questo significa che è quindi possibile fattorizzare lo *SLAM posterior* (4.2) in prodotto di termini più semplici. Murphy e Russell [15] sono riusciti per primi a fattorizzare in modo esatto, e non approssimato, lo *SLAM posterior*, il loro lavoro affermava che è possibile fattorizzarlo in un prodotto del *robot path posterior*, definito da $p(s^t | z^t, u^t, n^t)$ e da N *landmark posterior* condizionati sul percorso del robot.

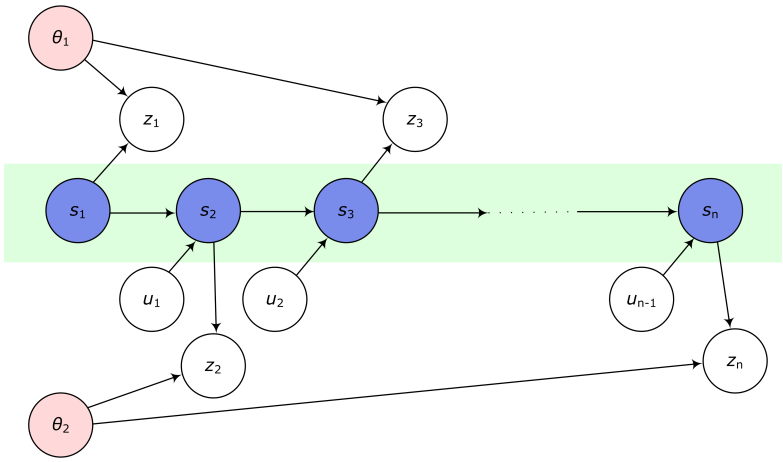


Figura 4.1: *Fattorizzarione del problema dello SLAM: se il reale percorso del robot è conosciuto (in verde), le posizioni dei landmark θ_1 e θ_2 sono condizialmente indipendenti.*

La fattorizzazione così ottenuta è la seguente:

$$p(s^t, \Theta | z^t, u^t, n^t) = \underbrace{p(s^t | z^t, u^t, n^t)}_{path\ \ posterior} \underbrace{\prod_{n=1}^{N} p(\theta | s^t, z^t, u^t, n^t)}_{n-landmark\ \ posterior} \tag{4.3}$$

4.2.1 Dimostrazione della fattorizzazione del FastSLAM

La fattorizzazione del FastSLAM deriva direttamente dallo *SLAM posterior* illustrato nella (4.2). Utilizzando la definizione di probabilità condizionata, lo *SLAM posterior* può essere riscritto come:

$$p(s^t, \Theta | z^t, u^t, n^t) \ = \ p(s^t | z^t, u^t, n^t)\, p(\Theta | s^t, z^t, u^t, n^t) \tag{4.4}$$

Per arrivare a dimostrare la (4.3) è sufficiente la seguente per tutti i valori non negativi di t:

$$p(\Theta | s^t, z^t, u^t, n^t) = \prod_{n=1}^{N} p(\theta_n | s^t, z^t, u^t, n^t) \tag{4.5}$$

La (4.5) può essere dimostrata per induzione. Bisogna prima raggiungere due risultati intermedi per raggiungere questo risultato. La prima quantità da ricavare è la probabilità condizionata del *landmark* θ_{n_t}. Questa quantità può essere riscritta utilizzando la regola di Bayes:

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Bayes}{=} \frac{p(z_t|\theta_{n_t}, s^t, z^{t-1}, u^t, n^t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} p(\theta_{n_t}|s^t, z^{t-1}, u^t, n^t) \quad (4.6)$$

L'osservazione corrente z_t dipende solamente dallo stato del robot e dai *landmark* osservati, per cui nel termine più a destra della (4.6) possiamo osservare che la posa corrente s_t, il controllo corrente u_t e la corrente associazione di dati n_t non hanno effetto su θ_{n_t} senza l'osservazione corrente z_t. Per cui si riscrive la (4.6) in:

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Markov}{=} \frac{p(z_t|\theta_{n_t}, s_t, n_t)}{p(z_t|s^t, z^{t-1}, u^t, n^t} p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (4.7)$$

Risolvendo rispetto al termine più a destra si ottiene:

$$p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) = \frac{p(z_t|s^t, z^{t-1}, u^t, n^t)}{p(z_t|\theta_{n_t}, s^t, n^t)} p(\theta_{n_t}|s^t, z^t, u^t, n^t) \quad (4.8)$$

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

$$p(\theta_{n\neq n_t}|s^t, z^t, u^t, n^t) \overset{Markov}{=} p(\theta_{n\neq n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (4.9)$$

Dati questi risultati intermedi possiamo dimostrare per induzione la (4.3). Assumiamo per ipotesi per l'induzione al tempo $t-1$:

$$p(\Theta|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) = \prod_{n=1}^{N} p(\theta_n|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (4.10)$$

Per $t = 0$ nessuna osservazione è stata incorporata nello *SLAM posterior* e la fattorizzazione (4.3) risulta essere banalmente verificata. In generale quando $t > 0$ dobbiamo espandere la parte sinistra della (4.3) con la regola di Bayes:

$$p(\Theta|s^t, z^t, u^t, n^t) \overset{Bayes}{=} \frac{p(z_t|\theta, s^t, z^{t-1}, u^t, n^t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} p(\theta|s^t, z^{t-1}, u^t, n^t) \quad (4.11)$$

Si può di nuovo semplificare con lo stesso metodo utilizzato nella (4.7), poiché z_t dipende solo da Θ, s_t e n_t il numeratore della (4.11) può essere semplificato, inoltre la posizione dei *landmark* presenti in Θ non dipendono da s_t, u_t, n_t senza l'osservazione corrente z_t, e quindi la (4.11) diviene:

$$p(\Theta|s^t, z^t, u^t, n^t) \overset{Markov}{=} \frac{p(z_t|\theta_{n_t}, s_t, n_t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} p(\Theta|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (4.12)$$

Per induzione è quindi possibile sostituire il termine a destra della (4.12),

$$p(\Theta|s^t, z^t, u^t, n^t) \overset{Induzione}{=} \frac{p(z_t|\theta_{n_t}, s_t, n_t)}{p(z_t|s^t, z^{t-1}, u^t, n^t)} \prod_{n=1}^{N} p(\theta_n|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \quad (4.13)$$

utilizzando quindi i risultati intermedi ottenuti dalle (4.8) e (4.9) si ottiene:

$$p(\Theta|s^t, z^t, u^t, n^t) = p(\theta_{n_t}|s^t, z^t, u^t, n^t) \prod_{n \neq n_t}^{N} p(\theta_n|s^t, z^t, u^t, n^t) \quad (4.14)$$

che è proprio il prodotto del *landmark posterior* della (4.5), ciò che si voleva dimostrare:

$$p(\Theta|s^t, z^t, u^t, n^t) = \prod_{n=1}^{N} p(\theta_n|s^t, z^t, u^t, n^t) \quad (4.15)$$

4.3 L'algoritmo del FastSLAM

La fattorizzazione dello *SLAM posterior*, esposta nella (4.3), evidenzia un importante struttura che è ignorata nel metodo dello SLAM che stima un *posterior* non strutturato, come per esempio quello basato su EKF. Questa struttura suggerisce che, sotto determinate condizioni, non è necessario mantenere le correlazioni tra i *landmark*. Il FastSLAM sfrutta appieno la fattorizzazione, dimostrata nella sottosezione 4.2.1, utilizzando $N + 1$ filtri di bassa dimensionalità, ognuno per ogni termine di essa.

Il FastSLAM, a differenza di altri metodi di SLAM, utilizza un filtro particellare per la stima del primo termine della (4.3), cioè per la stima del *path posterior*. I rimanenti N termini, i *landmark posterior*, vengono stimati tramite dei filtri

EKF. Ogni EKF, però, viene utilizzato per la stima di un solo *landmark*, è di bassa dimensionalità, e, soprattutto, non aumenta di dimensione. Gli stimatori EKF dei *landmark* sono condizionati sul percorso del robot, ogni particella componente del filtro particellare possiede il proprio insieme di filtri EKF. In totale esistono quindi $N \cdot M$ EKF, ove N è il numero di *landmark* ed M il numero di particelle. La struttura del filtro particellare del FastSLAM è descritta in fig.4.2, essa è in tutto assimilabile a una applicazione del filtro particellare di Rao-Blackwellized. Ogni particella ha la seguente struttura:

$$S_t^{[m]} = \left\langle s^{t,[m]}, \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, ..., \mu_{N,t}^{[m]}, \Sigma_{N,t}^{[m]} \right\rangle \tag{4.16}$$

- L'apice $[m]$ indica l'indice della particella.
- $s^{t,[m]}$ è la stima del percorso della particella *m*-esima.
- $\mu_{n,t}^{[m]}$ e $\Sigma_{n,t}^{[m]}$ rappresentano la media e la covarianza della distribuzione di probabilità gaussiana della posizione del *landmark* *n*-esimo condizionato sul percorso del robot $s^{t,[m]}$.

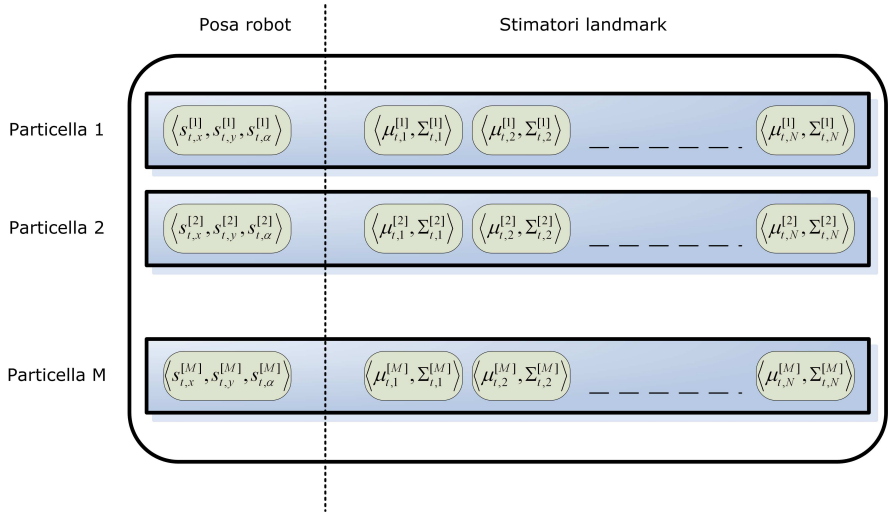


Figura 4.2: *Struttura del Particle filter.*

Tutte queste quantità formano la struttura della particella *m*-esima $S_t^{[m]}$, complessivamente M nello *SLAM posterior*. Calcolare il *posterior* al tempo t

dal tempo $t-1$, vuol dire generare un nuovo insieme di particelle S_t dal set dell'istante precedente S_{t-1}, questo processo viene chiamato *filtering*. Il nuovo insieme di particelle comprende i nuovi controlli u_t come pure le nuove misure z_t, con le corrispondenti associazioni di dati n_t. Il *filtering* viene effettuato con una sequenza di quattro passaggi:

1. Viene dedotta una nuova posa per ogni particella partendo dall'ultimo controllo u_t. Ogni posa è aggiunta alla stima del percorso del robot $s^{t,[m]}$.

2. L'EKF di ogni particella, corrispondente al *landmark* osservato viene aggiornato con la nuova osservazione.

3. Si calcola un peso, *importance weight*, per ogni particella; visto che le particelle non sono costruite sul reale *path posterior*, ma su una stima, a ogni particella viene assegnato un peso che riflette questa differenza.

4. Si crea un nuovo insieme di particelle S_t, basandosi su un criterio di *importance resampling* sull'insieme appena pesato. L'*importance resampling* è un passo cruciale per assicurare che le particelle siano distribuite sul vero *path posterior*[2].

Questi quattro passi verranno nell'ordine descritti in dettaglio nelle sottosezioni che seguono.

4.3.1 Deduzione della nuova posa

Il primo dei quattro passi del metodo del FastSLAM è quello di generare probabilisticamente un'ipotetica nuova posa al tempo t sulle basi di quella al tempo $t-1$. Questa procedura è effettuata campionando probabilisticamente il modello di moto:

$$s_t^{[m]} \sim p(s_t \,|\, u_t,\, s_{t-1}^{[m]}) \tag{4.17}$$

[2]Presupponendo un numero infinito di particelle.

Questa ipotesi di posa è aggiunta ad un insieme temporaneo di particelle lungo il percorso $s^{t-1,[m]}$. Sotto l'assunzione che l'insieme particellare S_{t-1} sia distribuito secondo $p(s^{t-1} \mid u^{t-1}, s^{t-1})$[3], allora le nuove particelle vengono distribuite secondo la seguente:

$$p(s^t \mid n^{t-1}, u^t, n^{t-1}) \tag{4.18}$$

che è comunemente denominata come *proposal distribution*. È importante sottolineare che il *motion model* può essere una qualsiasi funzione non lineare, in contrasto con il metodo EKF in cui essa deve essere necessariamente linearizzata. Un passaggio chiave è che qualunque sia la *proposal distribution*, la nuova posa viene calcolata in un tempo costante per ogni particella, qualunque sia la dimensione della mappa.

Viene proposto un semplice modello di moto a quattro variabili, per un robot che opera in un piano. Il modello assume che il robot mantiene velocità costante nell'intervallo di campionamento. Ogni controllo u_t consta di due variabili, una velocità traslazionale v_t e una velocità angolare ω_t, affette da rumore gaussiano. Gli errori su v_t ed ω_t sono di tipo moltiplicativo ed additivo. La notazione $\mathcal{N}(x; \mu, \Sigma)$ denota nel seguito del testo una distribuzione normale, della variabile x, a media μ e covarianza Σ. Per cui si scrive:

$$v_t^{'} \sim \mathcal{N}(v; v_t, \alpha_1 v_t + \alpha_2)$$
$$\omega_t^{'} \sim \mathcal{N}(\omega; \omega_t, \alpha_1 \omega_t + \alpha_2)$$

Un tale modello di moto riesce a descrivere i caratteristici errori di slittamento e pattinamento di un tipico robot su ruote. Esistono, inoltre, studi più approfonditi del modello di moto che modellizzano tali errori sistematici, in maniera nettamente più efficace come [12], che ha modellato in robot olonomi l'errore sistematico e l'errore non sistematico come parte complessa e parte reale di una variabile complessa ed eseguito un analisi più approfondita del problema realizzando un modello efficace anche in prove sperimentali, e altri [2] che hanno proposto altri metodi di stima dell'errore sistematico basandosi su particolari percorsi da fare

[3] Questa assunzione è valida in un insieme infinito di particelle.

eseguire al robot. Il primo passo per calcolare la nuova posa s_t è quello di simulare una nuova velocità traslazionale e rotazionale dal controllo corrente u_t, per poi utilizzarlo per aggiornare la posa $s_{t-1}^{[m]}$. Nella figura 4.3 è mostrato la disposizione di 250 particelle applicando questo *motion model* su una traiettoria curva.

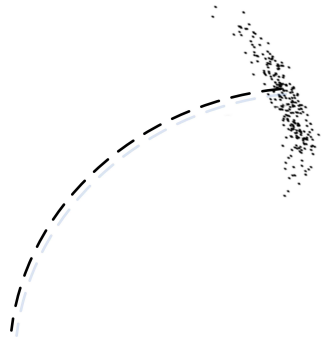


Figura 4.3: *Samples dedotti dal motion model, in questo esempio l'errore traslazionale è basso, mentre quello rotazionale è alto.*

Impoverimento dei campioni

I due passaggi chiave del *particle filtering* è campionare dalla *proposal distribution* e fare *importance resampling*, queste procedure possono essere pensate come un processo incrementale di costruzione ed eliminazione di un albero di traiettorie. Il campionamento dalla *proposal* genera nuove traiettorie, e il *resampling* cancella tutte quelle traiettorie improbabili per mantenere un numero costante di particelle. Risulta chiaro che un filtro particellare con molte traiettorie distinte ha un *sampling* molto vario rispetto a un *posterior* in cui il filtro particellare ha molte traiettorie duplicate. Il punto chiave è che, in generale, la diversità nell'insieme dei campioni porta a una migliore accuratezza di stima.

Il processo di *resampling* diminuisce questa diversità cancellando alcune particelle e duplicandone altre. Dopo che un certo numero di controlli e osservazioni sono stati incorporati nel filtro, è inevitabile che la traiettoria di ogni particella

condividerà parte della storia del suo percorso con le altre. Andando indietro nel tempo, per seguire la storia della traiettoria di ogni particella, si scoprirebbe che ad un certo punto tutte le particelle condivideranno tutte un comune antenato. La quantità di passi necessaria a trovare questo comune antenato è un valido indice per il filtro. Ogni particella nel FastSLAM rappresenta un percorso ipotetico del robot; le posizioni dei *landmark* sono indipendenti tra loro e sono condizionate su ognuno dei percorsi descritti dalle particelle. Più è vario l'insieme particellare, più accuratamente il FastSLAM può correggere la traiettoria del robot e le posizioni dei *landmark* data una nuova osservazione. La distanza del percorso fino al punto in cui si trova la traiettoria comune, si può considerare come la distanza prima di cui il filtro non può riconsiderare le ipotesi passate.

La quantità di diversità nell'insieme particellare è governata dal bilanciamento tra la *proposal* e il processo di cancellazione. Più ipotesi sono cancellate durante ogni aggiornamento, meno esisterà diversità nel set. La diversità può essere massimizzata se la *proposal distribution* e la *SLAM posterior distribution* sono identiche. Tuttavia se i *samples* fossero dedotti direttamente dallo *SLAM posterior*, non esisterebbe alcuna necessità di un filtro particellare. Quando si ha un robot con un più alto rumore di moto rispetto al rumore di misura, questo disallineamento delle due distribuzioni risulta essere più pronunciato: il modello di moto sparge le particelle su uno spazio ampio e solo una piccola parte di esse riceve pesi validanti. Nella figura 4.4 è mostrato un esempio di un *proposal* e *posterior* non coincidenti. Le particelle sono calcolate dal modello di moto molto rumoroso; quindi una osservazione successiva vincola la posa del robot di risiedere nell'ellisse, tutte le altre particelle fuori dall'ellisse riceveranno un peso molto basso e verranno eliminate nel processo di *resampling*, mentre le altre verranno da esso duplicate molte volte. se le osservazioni diventano più accurate, sempre meno particelle avranno una storia comune diversa. Ad un certo punto, successive osservazioni accurate porteranno il FastSLAM alla divergenza. Questo risultato non intuitivo dell'impoverimento dei campioni è un tipico caso di fallimento dei filtri particellari standard. Per questo è necessario modificare la *proposal distribution* in modo che il FastSLAM risulti avere una diversità più

ricca nel filtro particellare.

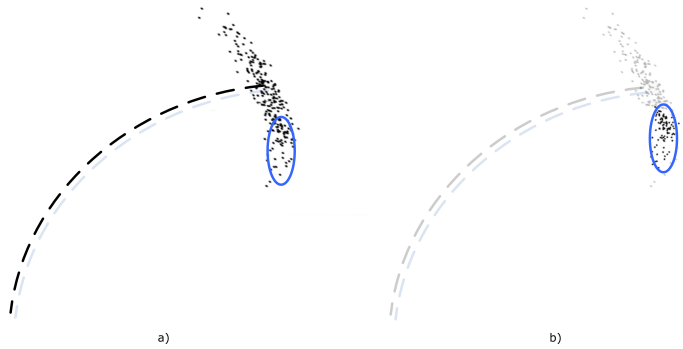


Figura 4.4: *Mismatching tra proposal distribution (a) e posterior distribution (b).*

La nuova *proposal distribution*

Si rappresentano gli standard modelli di moto e di misura come funzioni non lineari con rumore gaussiano indipendente:

$$p(z_t|s_t, \Theta, n_t) = g(s_t, \theta_{n_t}) + \epsilon_t \quad (4.19)$$

$$p(s_t|u_t, s_{t-1}) = h(s_{t-1}, u_t) + \delta_t \quad (4.20)$$

Dove h e g sono non lineari e δ_t e ϵ_t sono rumori gaussiani con covarianza rispettivamente R_t e P_t. Per dedurre la posa non si utilizza più solamente il modello di moto, bensì si utilizza un modello di moto che include le più recenti osservazioni z_t:

$$s_t^{[m]} \sim p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \quad (4.21)$$

Nella (4.21), $s^{t-1,[m]}$ è il percorso fino al tempo $t-1$ che ha eseguito la particella *m*-esima. Il meccanismo di *sampling* dalla (4.21) richiede però una analisi più accurata. Il nuovo modello di moto può essere riscritto in termini di distribuzioni conosciute, che includono il modello di moto e di misura standard, e gli stimatori gaussiani dei *landmark*. La *proposal distribution* viene espansa secondo la regola

di Bayes:

$$p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \overset{Bayes}{\propto}$$
$$\eta\, p(z_t|s_t, s^{t-1,[m]}, u^t, z^{t-1}, n^t)\, p(s_t|s^{t-1,[m]}, u^t, z^{t-1}, n^t) \tag{4.22}$$

La posa del robot s_t del secondo termine dipende solamente dalla posa precedente s_{t-1} e dal controllo corrente u_t; le restanti variabili nella probabilità condizionale possono essere eliminate, e si ottiene il modello di misurazione standard:

$$\overset{Bayes}{=} \eta\, p(z_t|s_t, s^{t-1,[m]}, u^t, z^{t-1}, n^t)\, p(s_t|s^{t-1,[m]}, u^t) \tag{4.23}$$

Si utilizza il teorema della probabilità totale per condizionare il primo termine del prodotto della precedente, sul *landmark* correntemente osservato θ_{n_t}. Quindi si utilizza la regola di Markov e si ottiene:

$$= \eta \int \underbrace{p(z_t|\theta_{n_t}, s_t, n_t)}_{\sim\mathcal{N}(z_t; g(s_t,\theta_{n_t}), R_t)} \underbrace{p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1})\, d\theta_{n_t}}_{\sim\mathcal{N}(\theta_{n_t}; \mu_{n_t,t-1}, \Sigma_{n_t,t-1})} \underbrace{p(s_t|s^{t-1,[m]}, u_t)}_{\sim\mathcal{N}(s_t; h(s^{[m]}_{t-1}, u_t), P_t)} \tag{4.24}$$

L'espressione (4.24) mostra che la distribuzione di *sampling* è una convoluzione di due distribuzioni di probabilità gaussiane moltiplicate per una terza. In forma generale questa distribuzione non possiede una soluzione in forma chiusa da cui è possibile effettuare campionamenti. Se la funzione g fosse lineare questa distribuzione sarebbe una distribuzione di probabilità gaussiana. Per questo motivo si linearizza g con una espansione in serie di Taylor al primo ordine:

$$g(s_t, \theta_{n_t}) \approx \hat{z}_t + G_\theta(\theta_{n_t} - \mu_{N_t,t-1}) + G_s(s_t - \hat{s}_t) \tag{4.25}$$

Dove $\hat{s}_t$ è la posa predetta del robot al tempo t, $\hat{z}_t$ la predetta osservazione per la data particella. Le matrici G_θ e G_s i jacobiani di g rispetto a θ e s, valutati al valore atteso dei loro argomenti. Possiamo quindi determinare una forma approssimata della distribuzione della (4.24):

$$\mathcal{N}(z_t;\, \hat{z}_t + G_s s_t - G_s \hat{s}_t,\, R_t + G_\theta \Sigma_{n_t,t-1} G\theta_t^T) \tag{4.26}$$

Da cui attraverso alcuni passaggi è possibile ottenere che la *proposal distribution* risulta:

$$p(s_t|s^{t-1,[m]}, z^t, u^t, n^t) = \xi e^{-y_t} \tag{4.27}$$

Dove y_t è funzione di $y_t = f(z, \hat{z}_t, s, \hat{s}_t)$. La media e la covarianza inoltre risultano essere:

$$\Sigma_{S_t}^{[m]} = \left[G_S^T Z_t^{-1} G_s + P_t^{-1}\right]^{-1} \tag{4.28}$$

$$\mu_{S_t}^{[m]} = \Sigma_{S_t}^{[m]} G_s^T Z_t^{-1}(z_t - \hat{z}_t) + \hat{s}_t^{[m]} \tag{4.29}$$

4.3.2 Aggiornamento della stima dei *landmark*

Il FastSLAM rappresenta le stime dei *landmark* utilizzando uno stimatore EKF di bassa dimensione $p(\theta_n|s^t, z^t, u^t, n^t)$, già descritto nella (4.3). Per semplicità in questa sezione l'associazione di dati n_t viene presupposta conosciuta, in seguito questa restrizione verrà rimossa.

Per la struttura del FastSLAM, le pose dei *landmark* sono condizionate sul percorso del robot utilizzando N filtri EKF allegati a ogni particella S_t, per cui il *posterior* sulla posizione dell'*n*-esimo *landmark* θ_n è facilmente ottenibile. Come già accennato, il tempo di computazione dipende solamente da quando accade che $n = n_t$, cioè quando l'associazione di dati restituisce l'informazione che è stato osservato il *landmark n*; in questo modo solo un *posterior* del *landmark n* dovrà essere modificato. Per il *landmark n* osservato viene eseguito il consueto sviluppo:

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Bayes}{=} \eta\, p(z_t|\theta_{n_t}, s^t, z^{t-1}, u^t, n^t)\, p(\theta_{n_t}|s^t, z^{t-1}, u^t, n^t) \tag{4.30}$$

Si utilizza la proprietà di Markov per semplificare i termini che compaiono nella precedente. L'osservazione z_t dipende solamente da θ_{n_t}, s_t ed n_t, mentre θ_{n_t} non è influenzata da s_t, u_t ed n_t senza l'osservazione z_t:

$$p(\theta_{n_t}|s^t, z^t, u^t, n^t) \overset{Markov}{=} \eta\, p(z_t|\theta_{n_t}, s_t, n_t)\, p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{4.31}$$

per le restanti *landmark* $n \neq n_t$ il *posterior* rimane intatto:

$$p(\theta_{n\neq n_t}|s^t, z^t, u^t, n^t) = p(\theta_{n\neq n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{4.32}$$

Il FastSLAM implementa la (4.31) con un EKF. Il filtro di Kalman esteso utilizza una approssimazione lineare gaussiana del modello misura, presupponendo ciò la

distribuzione risultante $p(\theta_{n_t}|s^t, z^t, u^t, n^t)$ è esattamente gaussiana, anche se il modello di moto non è lineare. È una tipica conseguenza di campionare sulla posa del robot. Il modello di misurazione non lineare $g(s_t, \theta_{n_t})$ è approssimato con una espansione in serie di Taylor del primo ordine. Lo stimatore del *landmark* è vincolato su un percorso fisso, per cui l'espansione di Taylor si effettua solamente sul θ_{n_t}. Si assume che l'errore di misura sia gaussiano con matrice di covarianza R_t. L'espansione in serie di Taylor è data da:

$$\begin{aligned} \hat{z}_t &= g(s_t^{[m]}, \mu_{n_t,t-1}) \\ G_{\theta_{n_t}} &= \nabla_{\theta_{n_t}} g(s_t, n_t)|_{s_t=s_t^{[m]};\theta_{n_t}=\mu_{n_t,t-1}} \\ g(s_t, n_t) &\approx \hat{z}_t + G_{\theta_{n_t}}(\theta_{n_t} - \mu_{n_t,t-1}) \end{aligned}$$

Utilizzando questa approssimazione il primo termine a destra dell'ugualianza del prodotto della (4.31) diventa:

$$p(z_t|\theta_{n_t}, s_t, n_t) \sim \mathcal{N}(z_t;\, \hat{z}_t + G_{\theta_{n_t}}(\theta_{n_t} - \mu_{n_t,t-1}, R_t) \tag{4.33}$$

Il secondo termine a destra dell'ugualianza della (4.31) è anch'esso gaussiano e risulta essere lo stato dell'EKF al tempo $t-1$:

$$p(\theta_{n_t}|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \sim \mathcal{N}(\theta_{n_t};\, \mu_{n_t,t-1}, \Sigma_{n_t,t-1}) \tag{4.34}$$

È possibile, infine, ottenere la media e la covarianza del prodotto presente nella (4.31) utilizzando le equazioni di aggiornamento dell'EKF.

Aggiornare i filtri di Kalman estesi di ogni particella è un operazione che richiede tempo costante poiché tali filtri sono a dimensione fissa; il tempo di computazione per incorporare un osservazione non dipende dal numero totale di *landmark*.

In particolare in un caso di SLAM planare, figura 4.5, con *landmark* completamente osservabili, i sensori del robot preposti all'osservazione effettuano misure di distanza ed orientamento dei *landmark* che lo circondano.

Assumendo che lo stato del robot sia descritto dalle tre variabili $\langle s_{t,x}, s_{t,y}, s_{t,\theta}\rangle$, e le correnti posizioni dei *landmark* da $\langle \theta_{n_t,x}, \theta_{n_t,y}\rangle$. La funzione

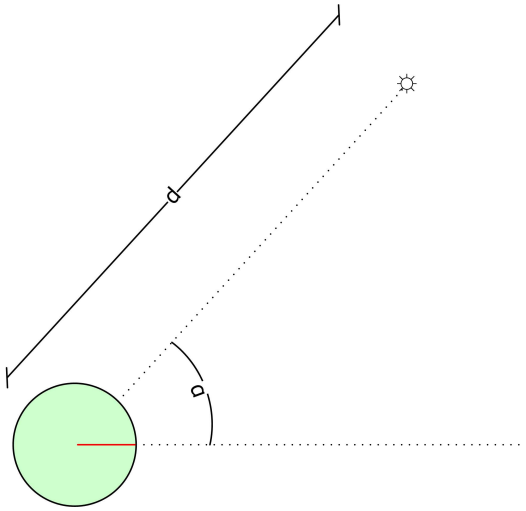


Figura 4.5: *I sensori del robot preposti all'osservazione effettuano misure di distanza ed orientamento dei landmark che lo circondano.*

non lineare di misura $g(s_t, \theta_{n_t})$ è descritta dalla:

$$g(s_t, \theta_{n_t}) = \begin{bmatrix} r(s_t, \theta_{n_t}) \\ \Phi(s_t, \theta_{n_t}) \end{bmatrix} = \begin{bmatrix} \sqrt{(\theta_{n_t,x} - s_{t,x})^2 + (\theta_{n_t,y} - s_{t,y})^2} \\ \tan^{-1}(\frac{\theta_{n_t,y} - s_{t,y}}{\theta_{n_t,x} - s_{t,x}}) \end{bmatrix} \quad (4.35)$$

Lo jacobiano $G_{\theta_{n_t}}$ è calcolato come:

$$G_{\theta_{n_t}} = \begin{bmatrix} \frac{\theta_{n_t,x} - s_{t,x}}{\sqrt{q}} & \frac{\theta_{n_t,y} - s_{t,y}}{\sqrt{q}} \\ -\frac{\theta_{n_t,y} - s_{t,y}}{\sqrt{q}} & \frac{\theta_{n_t,x} - s_{t,x}}{\sqrt{q}} \end{bmatrix} \quad (4.36)$$

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

4.3.3 Calcolo del peso della particella

Le particelle dedotte dal *motion model* sono distribuite secondo la $p(s^t|z^{t-1}, u^t, n^{t-1})$ che non coincide con il *posterior* desiderato $p(s^t|z^t, u^t, n^t)$. Questa differenza è corretta da un processo chiamato *importance resampling*. In generale questa procedura è una tecnica per dedurre *sampling* da funzioni in cui non è possibile effettuare una diretta procedura di *sampling*. Invece di

campionare direttamente da una funzione di *target*, i *samples* sono presi da una funzione più semplice (*proposal distribution*).

A ogni campione è assegnato un peso, uguale al rapporto della funzione di *target* sulla *proposal* proposta di quel punto nell'insieme di campionamento. Con questo procedimento viene creato un nuovo insieme di particelle non pesate, campionando secondo probabilità in proporzione ai pesi assegnati dell'insieme di particelle con peso. Invece di campionare direttamente dalla distribuzione *target*, si campiona da una distribuzione più semplice, *sample proposal*, una gaussiana.

Nelle regioni dove la distribuzione di *target* è più grande della *proposal*, alle particelle viene assegnato un peso grande; viceversa, nelle regioni in cui distribuzione di *target* è più piccola della *proposal* venono assegnati alle particelle pesi inferiori. Il risultato è che le particelle con più peso vengono scelte più spesso. Se si avesse un numero infinito di particelle questa procedura assicurerebbe *samples* distribuite secondo la reale distribuzione *target*.

Sotto l'assunzione asintotica che i percorsi in $s^{t-1,[m]}$ siano generati in accordo alla *target distribution* al passo precedente, cioè $p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})$, si nota che la *proposal distribution* è data dal prodotto:

$$p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1}) \cdot p(s_m^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t) \tag{4.37}$$

Nel FastSLAM, l'*importance weight* di ogni particella $w_t^{[i]}$ è uguale al rapporto dello *SLAM posterior* sulla *proposal distribution* descritta in precedenza:

$$\begin{aligned} w_t^{[i]} &= \frac{target\ distribution}{proposal\ distribution} \\ &= \frac{p(s^{t,[m]}|z^t, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})\, p(s_m^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t)} \end{aligned} \tag{4.38}$$

Il numeratore più essere espanso secondo la definizione di probabilità condizionale:

$$\begin{aligned} &= \frac{p(s_t^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t)\, p(s^{t-1,[m]}|z^t, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1}\, p(s_t^{[m]}|s^{t-1,[m]}, z^t, u^t, n^t))} \\ &= \frac{p(s^{t-1,[m]}|z^t, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1})} \end{aligned} \tag{4.39}$$

È possibile espandere ancora in numeratore con l'uso della regola di Bayes:

$$\overset{Bayes}{=} \eta \frac{p(z_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, p(s^{t-1,[m]}|z^{t-1}, u^t, n^t)}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1})} \tag{4.40}$$

Per la proprietà di Markov u_t ed n_t possono essere eliminati dal secondo termine del numeratore:

$$\begin{aligned} &\overset{Markov}{=} \quad \eta \frac{p(z_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t)\, p(s^{t-1,[m]}|z^{t-1}, u^{t-1}, n^{t-1})}{p(s^{t-1,[m]}|z^{t-1}, u^{t-1})} \\ &= \quad \eta\, p(z_t|s^{t-1,[m]}, z^{t-1}, u^t, n^t) \end{aligned} \tag{4.41}$$

Ora verrà applicato due volte il teorema della probabilità totale in modo da condizionare la (4.41) su s_t e θ_{n_t}:

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

$$\overset{Markov}{=} \eta \iint \underbrace{p(z_t|\theta_{n_t}, s_t, n_t)}_{\sim\mathcal{N}(z_t; g(\theta_{n_t}, s_t), R_t)} \underbrace{p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, u^{t-1}, n^{t-1})}_{\sim\mathcal{N}(\theta_{n_t}; \mu_{n_t,t-1}), \Sigma_{n_t,t-1})}\, d\theta_{n_t} \cdot \cdot \underbrace{p(s_t|s^{t-1,[m],u^t})ds_t}_{\sim\mathcal{N}(s_t; \hat{s}_t, P_t)} \tag{4.42}$$

I tre termini in questa espressione sono tutte distribuzioni di probabilità gaussiane, corrispondenti rispettivamente, al modello di misura, alla stima del *landmark* al tempo $t-1$ e al modello di moto. Utilizzando la linearizzazione di g l'espressione può essere calcolata in forma chiusa. Dopo l'applicazione del teorema di convoluzione si ottiene:

$$w_t^{[m]} \sim \mathcal{N}(z_t;\, \hat{z}_t, \underbrace{G_s P_t G_s^T + G_\theta \Sigma_{n_t,t-1}^{[m]} G_\theta^T + R_t}_{L_t}) \tag{4.43}$$

Montemerlo [13] è riuscito, da questa, ad ottenere la forma matriciale di $w_t^{[m]}$, cioè l'*importance weight* per la particella *m*-esima:

$$L_t = G_s P_t G_s^T + G_\theta \Sigma_{n_t,t-1}^{[m]} G_\theta^T + R_t \quad (4.44)$$

$$w_t^{[m]} = \sqrt{|2\pi L_t^{[m]}|} e^{\frac{1}{2}(z-\hat{z}_t^{[m]})^T L_t^{[m],-1} (z-\hat{z}_t^{[m]})} \quad (4.45)$$

Calcolare l'*importance weight* è un operazione a tempo costante per ogni particella.

4.3.4 Importance resampling

Una volta che alle particelle temporanee sono stati assegnati i pesi, viene creato un nuovo set di particelle S_t da questo set di particelle temporaneo, campionando con probabilità proporzionale ai pesi del set. Esistono diverse tecniche di *sampling* con cui creare S_t, come per esempio il *Madow's systematic sampling algorithm* [10], di particolare semplicità implementativa.

Questo *resampling* richiede un tempo lineare per il numero di *landmark* N, se implementato nella sua forma tecnica più basica. Questo è dovuto al fatto che ogni particella deve essere copiata nel nuovo insieme, e la lunghezza di ogni particella è proporzionale a N. In generale solo una piccola quantità di tutti i *landmark* viene osservata durante un singolo instante di osservazione, per cui copiare l'intera particella può essere abbastanza inefficiente. Una rappresentazione più sofisticata del FastSLAM riesce a ridurre i costi computazionali a un $O(\ln N)$, riducendo dispendiose copie inutili.

4.4 Associazione di dati sconosciuti

Fin qui si è descritto l'algoritmo del FastSLAM sotto la limitazione che l'associazione di dati n^t era considerata conosciuta. Nella realtà dei fatti questa è una assunzione quasi sempre violata. In questa sezione verrà esteso il FastSLAM a

domini in cui il *mapping* tra osservazioni e *landmark* non è conosciuto. Classicamente nello SLAM la soluzione a questo problema viene ovviata scegliendo n_t tale che massimizza un criterio di massima verosimiglianza, funzione delle misure sensoriali z_t e di tutti i possibili dati.

Si definisce:

$$\hat{n}_t = \arg\min_{n_t} p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t) \tag{4.46}$$

Il termine $p(z_t|n_t, \hat{n}^{t-1}, s^t, z^{t-1}, u^t)$ è denominato *likelihood*, e la (4.46) definisce un'istanza di uno stimatore di massima verosimiglianza, in inglese *maximum likelihood* (ML). Le associazioni di dati basati su ML sono anche chiamate associazioni di dati *nearest neighbor*, interpretando il logaritmo negativo del termine di *likelihood* come una funzione di distanza. Più semplicemente per modelli gaussiani, il *negative log likelihood* è nient'altro che la distanza di Mahalanobis, e lo stimatore seleziona le associazioni di dati minimizzando questa distanza.

Come spiegato nella sezione 3.2.1, l'EKF SLAM utilizza una singola associazione di dati per tutto il filtro, e questo risulta essere uno dei punti di debolezza dell'intera tecnica. Tale associazione di dati può causare errori significativi della mappa, con conseguenze fatali dal punto di vista della convergenza. È da considerare anche il fatto che associazioni di dati non corrette portano ad altre future associazioni di date errate, con conseguenze infauste sulla buona riuscita dello SLAM. Il FastSLAM, invece propone un'altra strada, più efficiente, per far fronte a queste difficoltà.

4.4.1 L'incertezza nell'associazione di dati

Due sono i principali fattori che contribuiscono all'incertezza nello *SLAM posterior*: il rumore di misura, e il rumore di moto. Se il rumore di misura aumenta, la distribuzione delle possibili osservazioni di ogni *landmark* diventa molto incerta; se il rumore di misura è sufficientemente alto le distribuzioni delle osservazioni di *landmark* vicini potrebbero sovrapporsi. Ovviamente questa sovrapposizione

genera un ambiguità sull'identificazione dei *landmark*; questa ambiguità nell'associazione causata da rumore di misura viene definita come ambiguità di misura. In figura 4.6 è mostrato un tipico caso di ambiguità di misura, le due ellissi mostrano la distanza delle probabili osservazioni di due *landmark* diversi; l'osservazione, evidenziata da un cerchio nero, potrebbe plausibilmente provenire da uno qualunque dei due *landmark*.

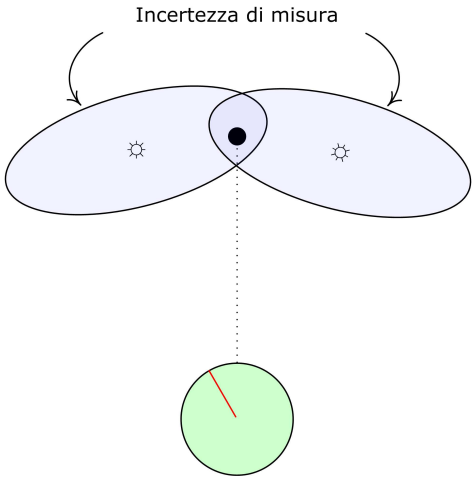


Figura 4.6: *Raffigurazione dell'incertezza di misura. Alti errori di misura possono provocare ambiguità tra landmark vicini.*

L'attribuire un'osservazione al *landmark* errato, causata dall'ambiguità di misura, incrementa l'errore della mappa e la posa del robot, ma ha un effetto sostanziale relativamente basso. L'osservazione potrebbe essere generata da qualunque dei due *landmark* con alta probabilità, per cui l'effetto sulla posa e sulla stima della posizione del *landmark* è basso; quel che succede è che la covarianza associata alla stima della posa di un *landmark* sarà leggermente sovrastimata, mentre la covarianza dell'altro sarà leggermente sottostimata. Spesso questa ambiguità di misura viene superata facendo uso di osservazioni multiple. L'ambiguità di una singola osservazione viene ad avere un piccolo impatto sul totale delle altre.

Un problema di incertezza nell'associazione di dati, notevolmente più importante risiede nell'ambiguità di moto. Esso ha conseguenze decisamente più gravi

sull'accuratezza della stima. Un rumore di moto alto porterà, dopo l'esecuzione di un controllo, a una incertezza della posa elevata. Se è presente un rumore di moto sufficientemente elevato, esso porterà ad ipotesi sulle associazioni di dati drasticamente differenti in osservazioni successive. Una ambiguità sul moto è frequente è facilmente introdotta se esiste un errore sul moto di rotazione del robot. Oltretutto se sono effettuate più osservazioni per istante, a causa della posa del robot fortemente incerta, l'associazione di dati avrà ipotesi ambigue per ogni osservazione. Se l'algoritmo di SLAM sceglie una associazione di dati errata per una singola osservazione, a causa dell'incertezza di moto, le restanti associazioni di dati saranno errate con alta probabilità. Scegliere un gran numero di associazioni di dati errate conduce inevitabilemente alla divergenza dell'EKF SLAM.

4.4.2 Associazione di dati particellare

Diversamente dall'EKF SLAM, il FastSLAM utilizza un approccio multi-ipotesi per il problema dell'associazione di dati. Ogni particella rappresenta un percorso differente del robot, per cui può essere effettuata un associazione di dati su base particellare. Alle particelle che selezionano associazioni di dati non corrette verranno assegnati pesi bassi e saranno rimosse nei futuri passaggi di *resampling*.

Un tale approccio al problema ha significativi vantaggi sull'associazione di dati standard basata su ML. Innanzi tutto si fattorizza l'incertezza della posa del robot fuori del problema dell'ambiguità di dati. L'ambiguità sul moto è una forma più grave di ambiguità di associazione di dati, per cui condizionare le ipotesi di associazione di dati sul percorso del robot risulta essere una strategia efficace. In figura 4.7 alcune particelle avranno una nuova posa consistente all'associazione di dati di sinistra o di destra. Sarà compito dei futuri passi di *importance resampling* determinare il comportamento corretto della vita della particella.

Effettuare associazione di dati su base particellare rende il problema dell'associazione di dati più facile. Nel metodo di SLAM che fa uso dell'EKF, l'incertezza della posizione di un *landmark* è composizione sia dell'incertezza della posa del

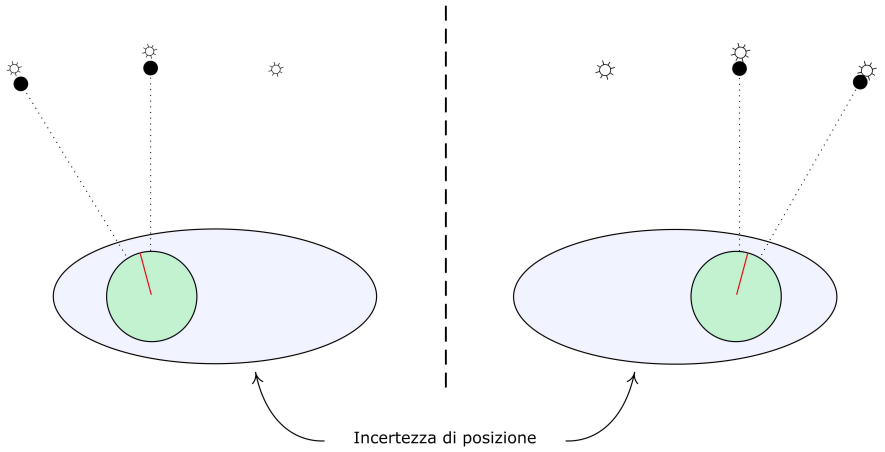


Figura 4.7: *L'incertezza della posa, area evidenziata in grigio, può causare significativi errori sulla associazione di dati.*

robot sia dell'incertezza della misura del *landmark*. Nel FastSLAM invece l'incertezza della posa del robot è rappresentata dall'intero insieme delle particelle. I filtri per la stima dei *landmark* in un filtro particellare non sono affetti da rumore di moto, perché sono condizionati su uno specifico percorso del robot. È come se ogni particella rappresentasse una completa probabile istanza di un robot. Questo metodo evidenzia i suoi vantaggi se il robot ha un movimento rumoroso e una accurata apparecchiatura sensoriale di misurazione. Un altro importante vantaggio è rappresentato dal *delayed decision making*. A ogni istante una parte delle particelle riceverà una plausibile, ma errata, associazione di dati. Nel futuro, il robot potrebbe ricevere una nuova osservazione che rifiuterebbe con sicurezza le assegnazioni precedenti. In quel momento le particelle con associazione di dati non corrette riceverebbero un peso basso e saranno candidate ad essere rimosse dal filtro. L'effetto di una associazione di dati errata, effettuata in passato, può essere quindi corretta. Oltretutto, non ci sono regole euristiche per rimuovere dal filtro particellare precedenti associazioni di dati non corrette; questo passaggio è effettuato in una valida maniera statistica, come conseguenza del processo di *resampling*.

4.4.3 Procedure di associazione di dati per il FastSLAM

Associazione di dati ML particellare

L'approccio più immediato di un associazione di dati particellare è applicare il principio della associazione di dati ML, con l'unica modifica che la valutazione deve essere effettuata indipendentemente per ogni particella. Tenendo conto che gli stimatori di posizione sono costituiti da EKF, la *likelihood* della (4.46) può essere calcolata utilizzando le innovazioni. Questa *likelihood* è in tutto simile al calcolo dell'*importance weight* calcolato per il FastSLAM. Se il valore di questa *likelihood* scende sotto una certa soglia p_0 un nuovo *landmark* viene aggiunto alla particella. L'associazione di dati ML tende a funzionare molto meglio nel FastSLAM piuttosto che nell'approccio SLAM EKF. La ragione principale risiede nel fatto che la componente più sensibile dell'ambiguità dell'associazione di dati proviene direttamente dall'incertezza della posa del robot. Una parte di queste particelle avrà una nuova posa che è consistente con la vera posa del robot. Queste pose riceveranno associazioni di dati corrette e saranno esplicative delle osservazioni. Invece, particelle che avranno pose del robot lontane dalla posizione effettiva riceveranno associazioni di dati non corrette, che saranno poco esplicative dei dati ricevuti.

Associazione di dati Monte Carlo

Mentre l'associazione di dati ML particellare si focalizza sull'ambiguità di moto non tiene conto dell'ambiguità di misura. Ogni osservazione è accoppiata con il *landmark* che con più probabilità l'ha generato, se comunque il rumore di misura è alto potrebbero esserci diverse associazioni di dati plausibili per ogni osservazione. Partendo da queste considerazioni è possibile menzionare un altro approccio all'associazione di dati, assegnando probabilisticamente le corrispondenze *landmark*-osservazione in accordo alla loro verosimiglianza. Questo approccio è descritto come *Monte Carlo Data Association*. L'approccio ML non considera mai la possibilità che associazioni di dati di *landmark* vicini potrebbero

essere confusi a causa dell'errore di misura. Il metodo Monte Carlo genererà esponenzialmente più possibili associazioni di dati al crescere dell'errore di misura; questo metodo richiede un numero più grande di particelle per mantenere la stessa accuratezza di un metodo ML.

Associazione di dati Niento

Un altro approccio per l'associazione di dati è stato proposto da Niento [17], questa procedura elenca tutte le possibili K associazioni di dati per una data osservazione e particella. Le osservazioni plausibili sono definite come corrispondenze con una verosimiglianza sotto una soglia minima. Ogni particella è quindi clonata K volte e l'osservazione è incorporata in ogni particella con la corrispondente associazione di dati. L'insieme particellare è quindi nuovamente ridotto a M elementi grazie al processo di *resampling*. Questo metodo è in tutto assimilabile al procedimento di associazione di dati con il metodo Monte Carlo poiché la particella sopravviverà con probabilità proporzionale alla verosimiglianza dell'osservazione.

4.4.4 Aggiunta di un *landmark*

Come si è valutato nelle sezioni precedenti aggiungere un nuovo *landmark* è un'operazione delicata, anche nel FastSLAM. Se la funzione di misura $g(\theta_{n_t}, s_t)$ è invertibile, una singola misura è sufficiente ad inizializzare un nuovo *landmark*. Ogni osservazione definisce una distribuzione di probabilità gaussiana:

$$\mathcal{N}(z_t;\, \hat{z}_t + G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t}), R_t) \tag{4.47}$$

Il gaussiano può essere scritto esplicitamente come:

$$\frac{1}{\sqrt{|2\pi R_t|}} e^{-\frac{1}{2}(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))^T R_t^{-1} (z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))} \tag{4.48}$$

Si definisce una funzione J uguale all'argomento, con il segno cambiato, dell'esponente della precedente (4.48):

$$J = \frac{1}{2}(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))^T R_t^{-1} (z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1})) \tag{4.49}$$

La derivata seconda di J rispetto a θ_{n_t} risulterà essere l'inversa della matrice di covarianza del gaussiano nelle coordinate del *landmark*:

$$\frac{\partial J}{\partial \theta_{n_t}} = -(z_t - \hat{z}_t - G_{\theta_{n_t}}(\theta_{n_t} - \mu^{[m]}_{n_t,t-1}))^T R_t^{-1} G_{\theta_{n_t}} \quad (4.50)$$

$$\frac{\partial^2 J}{\partial \theta^2_{n_t}} = G^T_{\theta_{n_t}} R_t^{-1} G_{\theta_{n_t}} \quad (4.51)$$

Conseguentemente una osservazione invertibile può essere utilizzata per creare un nuovo *landmark* come segue:

$$\mu^{[m]}_{n_t,t} = g^{-1}(s^{[m]}_t, z_t) \quad (4.52)$$

$$\Sigma^{[m]}_{n_t,t} = (G^T_{\theta_{n_t}} R_t^{-1} G_{\theta_{n_t}})^{-1} \quad (4.53)$$

$$w^{[m]}_t = p_0 \quad (4.54)$$

In termini più pratici una procedura di inizializzazione più semplice è ugualmente efficace. Invece di calcolare l'esatta covarianza iniziale di un nuovo *landmark*, la covarianza può essere calcolata assegnando la varianza di ogni variabile del *landmark* a un valore alto, e incorporare nello stimatore la prima osservazione. Si riscrive la (4.54):

$$\mu^{[m]}_{n_t,t} = g^{-1}(s^{[m]}_t, z_t) \quad (4.55)$$

$$\Sigma^{[m]}_{n_t,t} = K \cdot I \quad (4.56)$$

Più K è grande più si approssima la vera covarianza, ma è causa anche di stabilità numerica. Esistono anche tecniche di inizializzazione per situazioni in cui g non è invertibile, come nel caso del *bearing only SLAM*[4], in cui vengono effettuate osservazioni multiple per stimare la locazione del *landmark*.

4.5 Considerazioni sull'algoritmo del FastSLAM

Fattorizzare il problema dello SLAM utilizzando il percorso del robot sembrerebbe una cattiva idea visto che nel tempo la lunghezza delle particelle del FastSLAM

[4]SLAM in cui il *motion model* è funzione solo dell'orientamento e del suo errore.

aumenta. Tuttavia nessuna delle equazioni di aggiornamento dipende dal percorso totale t, soltanto la posa più recente $s_{t-1}^{[m]}$ è utilizzata per aggiornare l'intero *particle set*. Di conseguenza possiamo dimenticare tutte le pose precedenti meno che quella più recente nella parametrizzazione delle particelle. Questo evita il grande problema computazionale che risulterebbe se la dimensionalità di ogni particella crescesse nel tempo. Le particelle nell'algoritmo completo del FastSLAM hanno la forma:

$$S[m]_t = \left\langle s_t^{[m]} | N_t^{[m]}, \mu_{1,t}^{[m]}, \Sigma_{1,t}^{[m]}, \ldots, \mu_{N_t^{[m]},t}^{[m]}, \Sigma_{N_t^{[m]},t}^{[m]} \right\rangle \tag{4.57}$$

Oltre all'ultima posa $s_t^{[m]}$ e alla stima dei *landmark* $\mu_{n,t}^{[m]}$ e $\Sigma_{n,t}^{[m]}$, ogni particella mantiene il numero di *feature* $N_t^{[m]}$ della sua mappa. È molto importante notare che ogni particella può avere un diverso numero di *landmark*, proprio grazie alla sua particolare indipendenza. Fin qui si è trattata una singola osservazione per ogni controllo per semplicità computazionale, ma letture multiple possono essere implementate per istante temporale, processando ogni osservazione sequenzialmente. Il peso di ogni particella è uguale al prodotto dei pesi dovuti a ogni singola osservazione. Utilizzare osservazioni multiple aumenta l'accuratezza dell'associazione di dati e la precisione della mappa risultante.

4.6 Estensioni del FastSLAM

Vengono qui spiegate alcune delle estensioni del FastSLAM per aumentare la sua precisione o la sua *performance* computazionale.

4.6.1 Eliminazione di *feature* con *Negative evidence*

La rappresentazione del *landmark* puntuale nella mappa è una rappresentazione affermativa dell'esistenza di un *landmark*, la mappa descrive dove sono i *landmark* ma non ci dice dove non sono presenti. Come pure le osservazioni vengono tipicamente utilizzate per provare l'esistenza di una *feature* del mondo esplorato.

Le osservazioni possono però essere utilizzate per verificare se un *landmark* esiste o no. Se il robot predice che dalla sua posizione dovrebbe vedere un *landmark*, ed invece effettivamente questo non accade, la mancanza di informazione fornita da ciò prova che tale *landmark* in realtà non esiste. Questi falsi *landmark* possono occorrere soprattuto se il sistema sensoriale di misura genera misure spurie. La mancata osservazione di un *landmark*, denominata *negative evidence*, può essere utilizzata per rimuovere falsi *landmark* causati da osservazioni molto rumorose.

Questa tecnica può essere facilmente implementata nel FastSLAM. Sia $\tau_n^{[m]}$ una variabile binaria che indica l'esistenza di una *feature* $\theta_n^{[m]}$. Osservare il *landmark* $\theta_n^{[m]}$ fornisce un prova positiva della sua esistenza, mentre non osservare tale *landmark* quando esso è nell'area percettuale del sistema sensoriale del robot, fornisce una prova negativa della sua esistenza. È formulabile un *posterior* di distribuzione probabilistica:

$$p(\tau_n^{[m]}|\hat{n}^{t,[m]}, s^{t,[m]}, z^t) \tag{4.58}$$

Il calcolo è rapidamente effettuabile in tempo reale. Appena $\tau_n^{[m]}$ scende sotto una certa soglia fissata, che corrisponde alla minima probabilità di esistenza, allora tale *landmark* viene eliminato dal filtro. Questo procedimento permette al FastSLAM di sopprimere dalle particelle falsi *landmark* causati da misure particolamente rumorose.

4.6.2 Riduzione della complessità a Log(N)

La complessità computazionale dell'algoritmo del FastSLAM qui presentato richiede $O(M \cdot N)$ dove M è il numero di particelle ed N il numero di *landmark* della mappa. La complessità lineare M è fissata dal fatto che è necessario processare M particelle per ogni aggiornamento del filtro, la complessità lineare N invece è attribuibile alla tecnica di *importance resampling*. Il processo di *importance resampling* però può causare che una singola particella dell'insieme pesato può essere duplicata diverse volte nel *particle set* S_t. Il metodo più semplice di implementazione è quello di copiare ripetutamente l'intero contenuto della particella

nel nuovo insieme. È importante sottolineare che la lunghezza della particella dipende linearmente da N, così un'operazione di copia richiede un costo lineare rispetto alla grandezza della mappa. Inoltre la maggior parte dei filtri EKF per i *landmark* non vengono cambiati ad ogni intervallo di tempo, se si utilizza un così conservativo metodo di *resampling* delle particelle, verranno inutilmente copiati una gran quantità di dati.

Da queste considerazioni è possibile suggerire una migliore rappresentazione del filtro particellare; ogni particella invece di essere composta da un *array* di filtri per i *landmark*, diventa costituita da un albero binario ove media e covarianza di ogni *landmark* vengono rappresentate nelle sue foglie. Ogni sottoalbero, contenente medie e covarianze, può essere condiviso da diverse particelle. Condividere sottoalberi rende le funzioni di aggiornamento più complesse ma dal punto di vista computazionale enormemente più efficienti, soprattutto quando sono presenti molti *landmark*. Se si fa l'assunzione che l'albero sia bilanciato allora accedere a una foglia richiede una ricerca binaria, che in media richiede un tempo $\ln(N)$, la stessa complessità computazionale è richiesta per la creazione di sottoalberi. In generale l'intero processo di aggiornamento di M particelle richiede un tempo di computazione pari a $O(M \cdot \ln(N))$. Si ricorda al lettore che il metodo di SLAM EKF, invece, aveva complessità computazionale $O(N^2)$

Organizzare le particelle come alberi binari solleva altre difficoltà. I sottoalberi sono continuamente condivisi tra particelle. Quando però esiste un sottoalbero che non è più assegnato ad alcuna particella, a causa di un processo di *resampling*, esso deve essere eliminato. Tale processo è chiamato *garbage collection* ed è un metodo che si applica al FastSLAM senza particolari difficoltà con tecniche ottime.

Quando il *mapping* tra *landmark* e osservazione non è conosciuto, nel caso cioè delle associazioni di dati sconosciute, gli alberi contenenti i dati del *landmark* associati ad ogni particella devono essere organizzati spazialmente. Si devono quindi implementare tecniche di ricerca e costruzione bilanciata per tali alberi con metodi avanzati e complessi, che comunque riescono a garantire una complessità di $O(M \cdot \ln(N))$.

4.7 Convergenza del FastSLAM

L'algoritmo del FastSLAM è un algoritmo molto potente ed efficace, a tal punto da funzionare, e da convergere, anche con una sola particella. Il tempo richiesto per incorporare un osservazione non è affetto dalla grandezza della mappa. Nel filtro particellare standard, il *resampling* è l'unica fase dell'algoritmo che deduce il *sample set* in accordo alle osservazioni, e se esistesse una sola particella il *resampling* non può cambiare la sua traiettoria e il filtro tenderà a divergere. Visto che la nuova *proposal distribution* incorpora le osservazioni, anche con una singola particella, il FastSLAM può minimizzare l'errore della mappa proponendo una nuova posa che è in accordo alle osservazioni.

In questa sezione si dimostrerà la convergenza dell'algoritmo del FastSLAM, per semplicità, costituito da una sola particella, che implica che esso converge per M particelle. La dimostrazione ha due importanti conseguenze: la prova della convergenza di questo metodo di SLAM con complessità computazionale lineare e, oltretutto, il superamento del concetto che è necessario mantenere l'intera matrice di covarianza tra i *landmark* per attuare la convergenza. Il FastSLAM infatti non mantiene alcuna informazione sulla loro correlazione.

Il risultato di convergenza presentato qui si applica a problemi di SLAM lineare con rumore gaussiano. Lo SLAM lineare gaussiano, LG-SLAM, è costituito da un robot che opera su un piano cartesiano equipaggiato di una bussola senza rumore e di un sensore che misura le distanze delle *feature* sugli assi coordinati. In questo scenario, il mapping tra osservazioni e *landmark* è assunto conosciuto. I problemi di LG-SLAM hanno modelli di misurazione e di moto definiti come:

$$g(s_t, \theta_{n_t}) = \theta_{n_t} - s_t \tag{4.59}$$

$$h(u_t, s_{t-1}) = u_t + s_{t-1} \tag{4.60}$$

L'LG SLAM è troppo restrittivo per avere significato pratico, ma ha un significato importante nella letteratura scientifica dello SLAM. La proprietà principale di convergenza per il FastSLAM è la seguente:

Teorema 1 *Il FastSLAM Lineare gaussiano converge alla mappa reale con M=1 particelle se tutte le feature vengono osservate infinitamente, e se la locazione di una feature è conosciuta in anticipo.*

Se nessuna *feature* è conosciuta preventivamente, la mappa risulterà essere corretta in termini relativi, su un *offset* fissato che è applicato uniformemente a tutte le *feature*.

4.7.1 Prova di convergenza del FastSLAM

Prima di mostrare la prova della dimostrazione, è necessario riformulare le equazioni di aggiornamento per il problema LG-SLAM, con le apposite definizioni di g e h della (4.59) e (4.60). Si inizia con il riscrivere le equazioni per il calcolo della *proposal distribution*:

$$\begin{aligned}
\hat{s}_t &= h(s_{t-1}^{[m]}, u_t) = s_{t-1}^{[m]} + u_t && (4.61)\\
\hat{z}_t &= g(\hat{s}_t, \mu_{n_t,t-1}^{[m]}) = \mu_{n_t,t-1}^{[m]} - s_{t-1}^{[m]} - u_t && (4.62)\\
G_\theta &= \nabla_{\theta_n} g(s_t, \theta_n)|_{s_t=\hat{S}_t;\theta_n=\mu_{n_t,t-1}^{[m]}} = I && (4.63)\\
G_s &= \nabla_{s_t} g(s_t, \theta_n)|_{s_t=\hat{S}_t;\theta_n=\mu_{n_t,t-1}^{[m]}} = -I && (4.64)\\
Z_t &= R_t + G_\theta \Sigma_{n_t,t-1}^{[m]} G_\theta^T = R_t + \Sigma_{n_t,t-1}^{[m]} && (4.65)\\
\Sigma_{S_t} &= \left[G_S^T Z_t G_S + P_t^{-1}\right]^{-1} = \left[(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} P_t^{-1}\right]^{-1} && (4.66)\\
\mu_{S_t} &= \Sigma_{S_t} G_S^T Z_t^{-1}(z_t - \hat{z}_t) + \hat{s}_t && (4.67)\\
&= -\Sigma_{S_t}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1} \cdot \\
&\quad \cdot(z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + s_{t-1}^{[m]} + u_t && (4.68)\\
s_t^{[m]} &\sim \mathcal{N}(s_t;\, \mu_{S_t}, \Sigma_{S_t}) && (4.69)
\end{aligned}$$

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

$$\mu_{n_t,t} = \mu_{n_t,t-1} + \Sigma_{n_t,t-1}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(z_t - \mu_{n_t,t-1}^{[m]} + s_{n_t,t-1}^{[m]} + u_t) \quad (4.70)$$

$$\Sigma_{n_t,t} = (I - \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1})\Sigma_{n_t,t-1}^{[m]} \quad (4.71)$$

L'equazione 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 $\beta_t^{[m]}$, rispettivamente indicanti l'errore assoluto sulla posa del robot e sulla posizione del *landmark*, al tempo t.

$$\alpha_t^{[m]} = s_t^{[m]} - s_t \tag{4.72}$$

$$\beta_t^{[m]} = \mu_{n,t}^{[m]} - \theta_n \tag{4.73}$$

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 convergenza del FastSLAM è verificata tramite dei lemmi. Il primo lemma caratterizza gli effetti degli errori della mappa β con gli errori sulla posa α:

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

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

$$E[\alpha_t^{[m]}] = E[s_t^{[m]} - s_t] = E[s_t^{[m]}] - E[s_t] \tag{4.74}$$

Il primo termine può esser ricalcolato tramite la distribuzione di *sampling* data dalla (4.69), e il secondo termine è ottenuto dal modello di moto lineare della (4.60):

$$\begin{aligned} E[\alpha_t^{[m]}] &= E[-\Sigma_{S_t}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + s_{t-1}^{[m]} - u_t] \\ &\quad -E[u_t + s_{t-1}] \\ &= -\Sigma_{S_t}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(E[z_t] - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t) + \underbrace{s_{t-1}^{[m]} - s_{t-1}}_{\alpha_{t-1}^{[m]}} \end{aligned} \tag{4.75}$$

È possibile notare che nel LG-SLAM 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 (4.75) diventa:

$$\begin{aligned} E[z_t] - \mu^{[m]}_{n_t,t-1} + s^{[m]}_{t-1} + u_t &= \theta_{n_t} - u_t - s_{t-1} - \mu^{[m]}_{n_t,t-1} + s^{[m]}_{t-1} + u_t \\ &= \theta_{n_t} - s_{t-1} - \mu^{[m]}_{n_t,t-1} + s^{[m]}_{t-1} \\ &= \alpha^{[m]}_{t-1} - \beta^{[m]}_{n_t,t-1} \end{aligned} \tag{4.76}$$

Utilizzando questo risultato nella (4.75) e sostituendo $\Sigma^{[m]}_{S_t}$ dalla (4.67) risulta:

$$\begin{aligned} E[\alpha^{[m]}_t] &= \alpha^{[m]}_{t-1} + \Sigma^{[m]}_{S_t}(R_t + \Sigma^{[m]}_{n_t,t-1})^{-1}(\beta^{[m]}_{t-1} - \alpha^{[m]}_{t-1}) \\ &= \alpha^{[m]}_{t-1} + \left[(R_t + \Sigma^{[m]}_{n_t,t-1})^{-1} + P_t^{-1}\right]^{-1}(R_t + \Sigma^{[m]}_{n_t,t-1})^{-1}(\beta^{[m]}_{t-1} - \alpha^{[m]}_{t-1}) \\ &= \alpha^{[m]}_{t-1} + \left[I + (R_t + \Sigma^{[m]}_{n_t,t-1})P_t^{-1}\right]^{-1}(\beta^{[m]}_{t-1} - \alpha^{[m]}_{t-1}) \end{aligned} \tag{4.77}$$

Le matrici R_t, $\Sigma^{[m]}_{n_t,t-1}$ e P_t 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 posa attesa $\alpha^{[m]}_t$ si restringe se $\beta^{[m]}_{n_t,t-1}$ è più piccolo di $\alpha_{n_t,t-1}$. Succede il contrario, invece, se $\alpha_{n_t,t-1}$ è più piccolo di $\beta^{[m]}_{n_t,t-1}$. L'equazione (4.77) 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 $\beta^{[m]}_{n_t,t-1}$ nel suo valore atteso.

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

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 (4.77):

$$\begin{aligned} E[\alpha^{[m]}_t] &= \alpha^{[m]}_{t-1} + \left[I + (R_t + 0)P_t^{-1}\right]^{-1}(0 - \alpha^{[m]}_{t-1}) \\ &= \alpha^{[m]}_{t-1} - \left[I + R_t P_t^{-1}\right]^{-1}\alpha^{[m]}_{t-1}) \end{aligned} \tag{4.78}$$

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

Lemma 3 *Se l'errore di posa $\alpha_{t-1}^{[m]}$ è di grandezza più piccola dell'errore $\beta_{n_t,t-1}^{[m]}$ della feature osservata, l'osservazione z_t fa diminuire $\beta_{n_t,t}^{[m]}$ nel valore atteso. Se, invece, $\alpha_{t-1}^{[m]}$ è più grande dell'errore di misura $\beta_{n_t,t}^{[m]}$, quest'ultimo può aumentare, ma non superare il valore atteso di $\alpha_{t-1}^{[m]}$.*

Dimostrazione del Lemma 3: La dimostrazione di questo lemma è analoga a quella del Lemma 1. Dalla (4.70) segue che l'errore atteso della stima del *landmark*, dopo l'aggiornamento della sua posizione risulta essere:

$$\begin{aligned} E[\beta_{n_t,t}^{[m]}] &= E[\mu_{n_t,t}^{[m]} - \theta_{n_t}] = E[\mu_{n_t,t}^{[m]}] - \theta_{n_t} \\ &= E[\mu_{n_t,t}^{[m]} + \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(z_t - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t)] - \theta_{n_t} \\ &= \mu_{n_t,t}^{[m]} + \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(E[z_t] - \mu_{n_t,t-1}^{[m]} + s_{t-1}^{[m]} + u_t)] - \theta_{n_t} \end{aligned} \tag{4.79}$$

Con la (4.76) è possibile scrivere la precedente come:

$$\begin{aligned} E[\beta_{n_t,t}^{[m]}] &= \mu_{n_t,t}^{[m]} + \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(\alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]})] - \theta_{n_t} \\ &= \beta_{n_t,t-1}^{[m]} + \Sigma_{n_t,t-1}^{[m]}(R_t + \Sigma_{n_t,t-1}^{[m]})^{-1}(\alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]})] \\ &= \beta_{n_t,t-1}^{[m]} + (I + R_t\Sigma_{n_t,t-1}^{[m]-1})^{-1}(\alpha_{t-1}^{[m]} - \beta_{n_t,t-1}^{[m]})] \end{aligned} \tag{4.80}$$

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

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

$$\hat{\beta}_t^{[m]} = \arg\max_{\beta_{n_t}^{[m]}} |\beta_{n_t}^{[m]}| \tag{4.81}$$

Il Lemma 3 dichiara che il valore atteso di questo errore può aumentare, ma soltanto se l'errore assoluto della posa del robot $\alpha_{t-1}^{[i]}$ supera in grandezza questo errore. Tuttavia, questo sarà valido 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 (4.77)). Per incrementare $\alpha_{t-1}^{[m]}$ al suo precedente valore atteso, l'errore complessivo atteso sulle *feature* diminuisce secondo la (4.80). 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 posa atteso (Lemma 1), si ottiene la convergenza del valore atteso dell'errore sulla posa del robot. Il teorema di convergenza implica inoltre, un corollario riportato qui a seguito, che caratterizza la convergenza del FastSLAM lineare gaussiano a più di una particella:

Corollario 1 *Il FastSLAM converge nei valori attesi all'LG-SLAM se tutte le feature sono osservate infinitamente e la posizione di una feature è conosciuta in anticipo.*

4.8 FastSLAM in ambienti dinamici

Sì è illustrato in queste pagine il funzionamento del FastSLAM in ambienti statici, in ambienti cioè dove i *landmark* sono fissi. L'incertezza delle posizioni dei *landmark* può soltanto decrescere, diventando, eventualmente, una mappa completamente correlata. Questa assunzione di un mondo statico è più che accettabile per determinati tipi di SLAM, per esempio quando viene applicato per esplorazioni sotterranee, oppure interplanetarie, ma in un ambiente popolato da esseri umani, come per esempio strade o uffici, ciò è poco ragionevole.

La soluzione di aggiungere un modello di moto ai *landmark* non è una modalità risolutiva convincente, poiché porta a un mal condizionamento dell'intero problema. La posa del robot nello SLAM può solo essere determinata rispetto alla mappa. Se, effettivamente, gli elementi stessi della mappa si muovono nel

tempo, si può dedurre molto poco dalle informazioni ricavate dall'ambiente sulla posa del robot. In questa situazione, ne l'errore della mappa tantomeno quello della posa del robot può essere limitato nel tempo.

Un approccio risolutivo comune a questo problema è classificare il mondo circostante al robot in due grandi classi: oggetti statici ed oggetti dinamici. Grazie a questa distinzione è possibile trattare tali oggetti in modo diverso durante il processo di stima. La classe di oggetti statici, viene utilizzata per limitare l'errore sulla mappa e della posa del robot, mentre gli oggetti dinamici sono inseguiti separatamente. Questo approccio è quello comunemente utilizzato con tecniche di *scan-matching*. Montemerlo [13], invece, ha sviluppato, ed applicato con buoni risultati un'altra metodica. Assumendo di conoscere in anticipo la mappa degli oggetti statici del mondo (magari dopo aver effettuato uno SLAM completo dell'area), si determina la posa del robot e le posizioni degli oggetti dinamici relativamente alla mappa. L'incertezza della posa del robot correla le posizioni degli oggetti dinamici osservati, similmente alla tecnica di FastSLAM classico.

Capitolo 5

L'implementazione del FastSLAM in Matlab

Dopo aver dettagliatamente descritto la teoria connessa al FastSLAM, in questo capitolo viene descritto il primo passo progettuale del lavoro svolto, l'implementazione simulativa della tecnica del FastSLAM in Matlab[1]. Questo è stato un passo necessario per la verifica, la comprensione e lo sviluppo della teoria, e il fondamento per l'applicazione pratica sul robot mobile. È stato scelto questo ambiente matematico come piattaforma simulativa per la sua estrema potenza e flessibilità. Per i nostri scopi è necessaria una buona potenza di calcolo e sopratutto l'utilizzo di strumenti matematici complessi e di matematica algebrico matriciale, tutto questo rende Matlab il candidato migliore rispetto ai suoi concorrenti. Un altro motivo è stato dettato dal fatto che Matlab è uno standard *de facto* per la comunità scientifica internazionale.

Non esistono, al momento, applicazioni Matlab di FastSLAM con associazioni di dati sconosciuti, per questo è stato necessario scriverlo completamente, basandosi principalmente sulla teoria e su alcuni esempi rilasciati dalla comunità scientifica.

Al momento della scrittura di questa tesi il FastSLAM rappresenta lo stato

[1]MathWorks Matlab è un software matematico/scientifico.

dell'arte dello SLAM. Avere un'implementazione funzionante su Matlab, e poi sul robot, rappresenta un passo fondamentale dello sviluppo del FastSLAM per estendere lo studio al funzionamento multirobot, o in ambienti dinamici.

L'algoritmo del FastSLAM è molto complesso, in questo capitolo verranno spiegati i passaggi fondamentali del lavoro svolto per l'implementazione simulativa.

5.1 Simulare un robot

Per simulare il FastSLAM si è dovuto simulare un modello semplificato di robot. Tale modello è stato progettato intorno al robot reale, per rendere la simulazione consistente con la futura implementazione pratica. Per il tipo di equipaggiamento delle apparecchiature di misura del robot è concepibile modellizzare esso come un ente geometrico piano in un mondo bidimensionale.

Lo scopo dell'implementazione del FastSLAM in Matlab è quello della verifica dello SLAM e non quello di svolgere metodi di esplorazione o metodi di *obstacle avoidance*. Per questo motivo si è supposto il robot come un puro ente geometrico, un punto, senza massa, con incertezza gaussiana sulla sua posa. Il sistema sensoriale di osservazione scelto del robot reale è rappresentato da un apparecchiatura di misurazione laser. Esso effettua delle osservazioni discretizzate sull'orizzonte visibile, cioè restituisce delle misure di distanza campionate del mondo circostante, coprendo un angolo piano. Similmente è stato costruito il modello di osservazione in Matlab, per essere consistente anch'esso con l'apparecchiatura laser presente nel robot.

5.1.1 Modello di moto e controllo

Il robot simulato si muove su una mappa bidimensionale. Si muove in moto uniformemente accelerato/decelerato quando procede traslando e di una combinazione di moto rotazionale e traslazionale uniformemente accelerato/decelerato

quando ruota. Questo tipo di rappresentazione costituisce una buona modellizzazione del robot reale, adatto a variazioni di velocità date dal processo di controllo. Il controllo u_t del robot è funzione della velocità di traslazione, accelerazione di traslazione, velocità angolare e accelerazione angolare e degli ingressi dell'algoritmo che decide la traiettoria: $u_t = f(v_t^r, a_t^r, \omega_t^r, \dot{\omega}_t^r, wp)$. Queste variabili vengono opportunamente disturbate da rumore gaussiano a media nulla sagomando la matrice di covarianza Q (diagonale) del rumore di moto.

5.1.2 Modello di misura

Il robot è equipaggiato con un apparecchiatura di rilevamento laser che restituisce per ogni osservazione effettuata una misura di distanza e l'angolo per cui è stata effettuata. Non è difficile progettare un modello virtuale di questo sensore, che, analogamente alla realtà, restituisce le coppie misura/angolo disturbate da rumore gaussiano a media nulla, tramite la matrice di covarianza diagonale R.

5.2 I *landmark*

Il primo passo per una applicazione di SLAM è la scelta della definizione di un *landmark*; la scelta, cioè, di cosa il robot deve riconoscere nell'ambiente come oggetto di riferimento per la correzione della sua posa, e fondamento della mappa. Due sono stati gli approcci inizialmente scelti. Il primo consisteva nell'estrazione di segmenti dalle osservazioni effettuate dal laser. L'altro nella selezione di punti particolari della mappa, quali angoli o spigoli.

Lo sviluppo del primo metodo è suggerito dal tipo ambiente in cui il robot sarebbe stato utilizzato in fase sperimentale, un ambiente *indoor* strutturato, quindi con presenza di pareti, angoli, porte; il *landmark* costituito da un segmento può ben modellizzare ognuna di queste strutture. Un altro vantaggio risiede nell'inferiore occupazione di memoria, in quanto, modellizzando i *landmark* come segmenti è facile intuire che durante un'esplorazione in un tale ambiente i

landmark totali saranno in un numero certamente basso. Il metodo di estrazione di segmenti ha però una alta complessità computazionale. Questo nelle prove sperimenali avrebbe rappresentato un problema, poiché la complessità di questa procedura si sarebbe riflettuta direttamente nel tempo di esecuzione di tutto il software, non ottenendo più esecuzione *real-time* del FastSLAM.

Il secondo metodo funziona estraendo da una catena di punti la posizione degli angoli. L'algoritmo di *corner detection* è una diretta implementazione dell'algoritmo IPAN99 di D. Chetverikov e Z. Szabò [4]. Questo è un algoritmo davvero veloce, circa cento volte tanto rispetto all'estrazione dei segmenti, e molto preciso. Con questo tipo di algoritmo i *landmark* sono rappresentati dagli angoli.

La *routine* selezionata per l'uso effettivo del FastSLAM è stato il procedimento di riconoscimento di angoli IPAN99. Per completezza, queste due procedure vengono spiegate nel seguito.

Per provare il corretto funzionamento del FastSLAM in diversi ambienti, si è programmato in Matlab uno strumento per la costruzione dell'ambiente virtuale da esplorare. Tale utilità permette di creare ambienti complessi strutturati dove provare il robot virtuale (si veda la figura 5.1). È possibile utilizzare linee, punti e cerchi per costruire facilmente con il *mouse* la mappa e salvare il risultato in un formato utilizzabile per la simulazione.

5.2.1 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, ambedue con la stessa complessità computazionale. In Matlab si è implementato un metodo nuovo basandosi sullo studio del metodo di Radon. Questo metodo consiste nel far passare per tutti i punti osservati delle rette, che variano il loro orientamento[2] $\alpha \in [-pi, pi]$.

[2]Si consideri che l'orientamento $0rad$ punta esattamente davanti al robot.

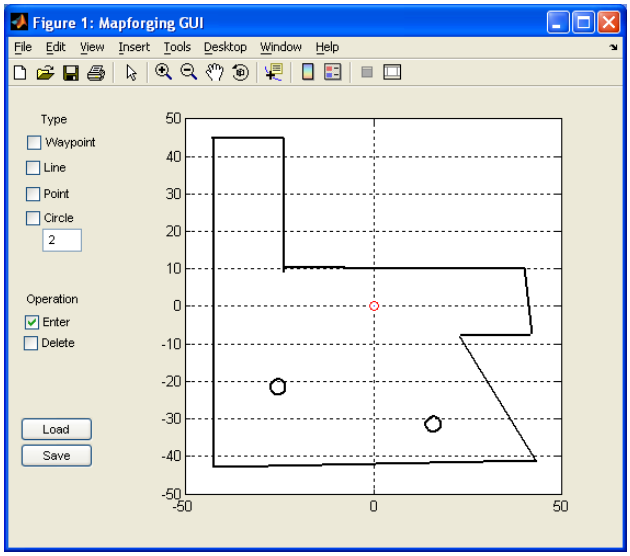


Figura 5.1: *Utilità per la creazione di ambienti virtuali.*

Vengono quindi contati il numero di punti che appartengono a tali rette. Da queste informazioni si costruisce un istogramma, mostrato in figura 5.2, 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 caratteristico rumore gaussiano 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. Particolare attenzione è stata concessa alle rette che hanno punti in comune, come gli angoli, per fare in modo che il metodo anche in tali situazioni funzionasse a dovere.

Utilizzare una soglia per selezionare le rette candidate è una procedura particolarmente delicata. Accade spesso che sopra questa soglia ci siano molte rette candidate di pendenza molto simile per il passaggio di punti molto vicini. È evidente che una situazione del genere indica in realtà, che è solo una retta quella che passa per tutti quei punti. È opportuna, quindi, una procedura di media che

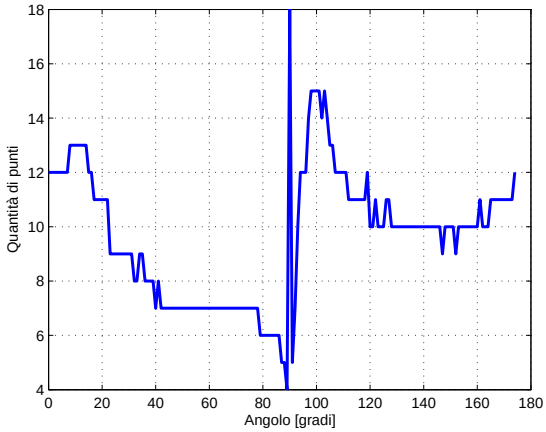


Figura 5.2: *Istogramma utilizzato per il riconoscimento di segmenti.*

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 per esempio se 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 che se si è in condizioni in cui sono stati 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 lavorazione quasi omogeneo per tutte le condizioni simulative nei vari ambienti.

5.2.2 IPAN99 Corner detection

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. Proprio sulle catene di punti lavora bene l'algoritmo IPAN99 sviluppato da Dmitry Chetverikov e Zsolt Szabó [4], il principale vantaggio rispetto agli altri algoritmi di *corner detection*, escludendo la maggiore velocità, è che è in grado di fun-

zionare su catene di punti non equispaziate, a differenza di altri algoritmi come quelli proposti da A. Rosenfeld e E. Johnston [21], A. Rosenfeld e J.S. Weszka [22], H. Freeman e L.S. Davis [6] e, H.L. Beus e S.S.H. Tiu [3] 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:

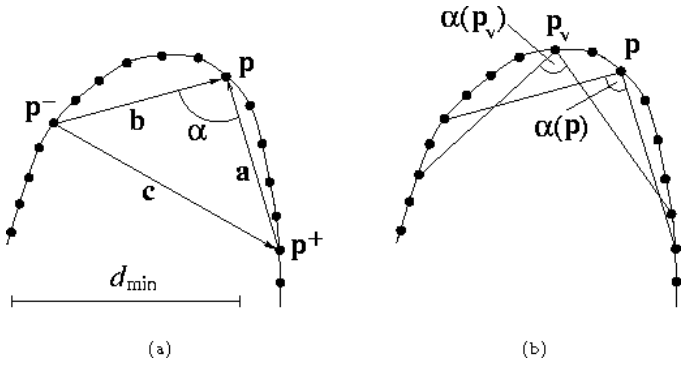
$$d_{min} \leqslant |p_i - p_i^+| \leqslant d_{max} \tag{5.1}$$

$$d_{min} \leqslant |p_i - p_i^-| \leqslant d_{max} \tag{5.2}$$

$$\alpha \leqslant \alpha_m ax \tag{5.3}$$

Dove d_{min}, d_{max} e $\alpha_m ax$ sono costanti fissate in anticipo 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, ad esempio 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 (5.2) definisce la distanza a tra p e p^+, il modulo $|p_i - p_i^-|$ della (5.3) definisce la

distanza b tra p e p^-, l'angolo α è proprio l'angolo di apertura del triangolo, si veda per chiarezza la figura 5.2.2. Il valore di α proviene direttamente



dal teorema di Carnot:

$$\alpha \overset{Carnot}{=} arc\cos(\frac{a^2+b^2-c^2}{2ab}) \tag{5.4}$$

Tutte quelle variazioni del triangolo che soddisfano le tre condizioni (5.2) sono chiamati 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 $c = (c_x, c_y)$ é facile distinguere se l'angolo è convesso o e concavo. Basta verificare se $b_x c_y - b_y c_x$ se $\geqslant 0$ 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 perché 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 $p_v : \alpha(p) \geq \alpha(p_v)$.

5.3 Il FastSLAM in Matlab

Verrà qui descritta l'implementazione del FastSLAM in Matlab e l'algoritmo nel dettaglio.

Lo *script* in Matlab è costituito da queste principali funzioni:

`init_particles`: inizializza le particelle del filtro.

`predict`: funzione addetta alla predizione.

`get_observation`: restituisce possibili osservazioni dei *landmark* dall'analisi delle misure.

`compute_jacobians`: effettua dalla posa di una particella del robot e dalla posa di un *landmark* selezionato, il calcolo dell'osservazione predetta, dello jacobiano degli stati del robot, dello jacobiano degli stati della *feature*, e della covarianza dell'osservazione della *feature*, condizionato alla posa del robot.

`data_association_unknown`: ha il compito di fare le corrette associazioni di dati tra *landmark* e osservazioni.

`compute_weight`: calcola i pesi delle particelle.

`resample_particles`: effettua il *resampling* delle particelle.

`sample_proposal`: crea la *proposal distribution* per le particelle.

`feature_update`: aggiorna la stima dei *landmark* per le particelle.

`add_feature`: aggiunge un *landmark* a una determinata particella.

5.3.1 L'inizializzazione del filtro

L'inizializzazione del filtro particellare è compiuta dalla funzione `init_particles`, che inzializza la posa iniziale di tutte le particelle al valore nullo, e i pesi al valore omogeneo $w = 1/M$, ove M è il numero di particelle presenti nel filtro.

5.3.2 La predizione

La predizione è un passaggio fondamentale per l'intero funzionamento del FastSLAM. Dato un intervallo di tempo dt, per ogni posa x_v delle particelle è calcolata la predizione al tempo t, conoscendo la legge di moto e la posa al tempo $t-1$. Gli ingressi di questa funzione sono gli ingressi di controllo: $v, \omega, a, \dot{\omega}$. Nella funzione `predict` viene calcolato il jacobiano del controllo J_u e della posizione J_v per poter calcolare la predizione della matrice di covarianza Pv:

$$J_v = \begin{bmatrix} 1 & 0 & -(v\,dt + \frac{1}{2}\,a\,dt^2)\,\sin(\delta_G + x_{v,\alpha,t-1}) \\ 0 & 1 & (v\,dt + \frac{1}{2}\,a\,dt^2)\,\cos(\delta_G + x_{v,\alpha,t-1}) \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

$$J_u = \begin{bmatrix} dt\,\cos(\delta_G + x_{v,3,t-1}) & -(v\,dt + \frac{1}{2}\,a\,dt^2)\,dt\,\sin(\delta_G + x_{v,\alpha,t-1}) \\ dt\,\sin(\delta_G + x_{v,3,t-1}) & (v\,dt + \frac{1}{2}\,a\,dt^2)\,dt\,\cos(\delta_G + x_{v,\alpha,t-1}) \\ 0 & dt \end{bmatrix} \tag{5.6}$$

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

Ricordando che la matrice Q sagoma l'errore sul modello di moto, la matrice di covarianza Pv, che misura l'incertezza della posa viene aggiornata in tale modo:

$$P_{v,t} = J_v\,P_{v,t-1}\,J_v^T + J_u\,Q\,J_u^T \tag{5.7}$$

La posa della singola particella x_v al tempo t, che si muove di moto uniformemente accelerato, viene così aggiornata:

$$x_{v,x,t} = x_{v,x,t-1} + (v\,dt + \frac{1}{2}\,a\,dt^2)\,\cos(\omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 + x_{v,\alpha,t-1}) \tag{5.8}$$

$$x_{v,y,t} = x_{v,x,t-1} + (v\,dt + \frac{1}{2}\,a\,dt^2)\,\sin(\omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 + x_{v,\alpha,t-1}) \tag{5.9}$$

$$x_{v,\alpha,t} = x_{v,\alpha,t-1} + \omega\,dt + \frac{1}{2}\,\dot{\omega}\,dt^2 \tag{5.10}$$

La matrice di covarianza P_v deve essere azzerata dopo ogni osservazione, in quanto accumula l'incertezza della posa del robot tra una misura e l'altra.

Questa funzione interviene sempre durante tutto il procedimento di FastSLAM. Nell'intervallo di tempo che trascorre tra una lettura dei sensori laser e l'altra, essa è l'unica funzione che ha il compito di aggiornare la posa delle particelle.

5.3.3 Le osservazioni

La funzione `get_observation` analizza le misure generate dal laser virtuale e restituisce le possibili osservazioni dei *landmark* come una coppie distanza-orientamento. Il laser virtuale è costituito da una *routine* che discretizza linee e cerchi generate dalla utilità di generazione di mappe, precedentemente descritta nella sezione 5.2. Questi punti vengono, quindi, disturbati con un rumore gaussiano a media nulla utilizzando la matrice di covarianza R di misura; vengono poi scartati i punti che sono più lontani della massima distanza individuabile ed infine, filtrati tramite un algoritmo di *z-buffering*[3]. Questo risultato risulta verosimile rispetto al processo di misurazione di un laser reale. L'insieme di punti così ottenuto viene passato al processamento della funzione di riconoscimento di angoli IPAN99 (si veda 5.2.2), prescelta per le sue qualità lì descritte. Dopo di ciò, i dati restituiti sono pronti per il delicato processamento della associazione di dati.

5.3.4 `Compute jacobians`

Questa funzione effettua, conoscendo la posa di una particella del robot e la posa 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 posa del robot. Si ricorda che la posa del robot è definita da $x_v = \langle x_{v,x}, x_{v,y}, x_{v,\alpha} \rangle$ e la posa del *landmark* selezionato da $\theta_n = \langle \theta_{n,x}, \theta_{n,y} \rangle$. Il vettore z_p rappresenta la distanza euclidea tra la posa del robot x_v e la posa del *landmark* θ_n selezionato.

[3]Lo *z-buffering* è una tecnica utilizzata per nascondere le linee nascoste dal campo visivo.

È possibile definire:

$$z = h(x_v, \theta_n) = \begin{bmatrix} \sqrt{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2} \\ atan(x_{v,y} - \theta_{n,y}, x_{v,x} - \theta_{n,x}) \end{bmatrix} \tag{5.11}$$

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

$$H_v = \begin{bmatrix} \frac{\partial h_1}{\partial x_v} \\ \frac{\partial h_2}{\partial x_v} \end{bmatrix} \tag{5.12}$$

$$= \begin{bmatrix} \frac{\partial h_1}{\partial x_{v,x}} & \frac{\partial h_1}{\partial x_{v,y}} & \frac{\partial h_1}{\partial x_{v,\alpha}} \\ \frac{\partial h_2}{\partial x_{v,x}} & \frac{\partial h_2}{\partial x_{v,y}} & \frac{\partial h_2}{\partial x_{v,\alpha}} \end{bmatrix} \tag{5.13}$$

$$= \begin{bmatrix} \frac{x_{v,x} - \theta_{n,x}}{\sqrt{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2}} & \frac{x_{v,y} - \theta_{n,y}}{\sqrt{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2}} & 0 \\ \frac{-(x_{v,y} - \theta_{n,y})}{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2} & \frac{x_{v,x} - \theta_{n,x}}{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2} s & -1 \end{bmatrix} \tag{5.14}$$

Analogamente si calcola il jacobiano H_f relativo alla distanza della *feature* selezionata rispetto agli stati del *landmark*:

$$H_f = \begin{bmatrix} \frac{\partial h_1}{\partial \theta_n} \\ \frac{\partial h_2}{\partial \theta_n} \end{bmatrix} \tag{5.15}$$

$$= \begin{bmatrix} \frac{\partial h_1}{\partial \theta_{n,x}} & \frac{\partial h_1}{\partial \theta_{n,y}} \\ \frac{\partial h_2}{\partial \theta_{n,x}} & \frac{\partial h_2}{\partial \theta_{n,y}} \end{bmatrix} \tag{5.16}$$

$$= \begin{bmatrix} \frac{-(x_{v,x} - \theta_{n,x})}{\sqrt{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2}} & \frac{-(x_{v,y} - \theta_{n,y})}{\sqrt{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2}} & 0 \\ \frac{x_{v,y} - \theta_{n,y}}{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2} & \frac{-(x_{v,x} - \theta_{n,x})}{(x_{v,x} - \theta_{n,x})^2 + (x_{v,y} - \theta_{n,y})^2} s & -1 \end{bmatrix} \tag{5.17}$$

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

$$S_f = H_f \cdot P_f \cdot H_f^T + R \tag{5.18}$$

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

5.3.5 L'associazione di dati sconosciuta

L'associazione di dati qui implementata è di tipo *simple gated nearest-neighbour particle-based* di complessità computazionale $O(N)$ per particella, dove N rappresenta il numero di *feature* presenti nella particella. Questo metodo si basa sul calcolo, per ogni osservazione, di due tipi di distanza tra la posizione dell'osservazione e la posizione di ogni *landmark* presente nella particella. Questi valori di distanza vengono sottoposti a una soglia da cui si decide se associarla a un *landmark*, ignorare l'osservazione o incorporare questa osservazione come un nuovo *landmark*. Nel dettaglio, viene calcolata la distanza normalizzata e la distanza di Mahalanobis, tra osservazione z e *feature n*:

$$v = z - zp \tag{5.19}$$

$$d_{Mahalanobis} = v^T \cdot S_f^{-1} \cdot v \tag{5.20}$$

$$d_{normalized} = d_{Mahalanobis} + \log(\det(S_f)) \tag{5.21}$$

La distanza di Mahalanobis misura l'innovazione normalizzata al quadrato. Il vettore v rappresenta il vettore di innovazione. S_f e z_p sono frutto del calcolo di `calculate_jacobians` per il *landmark* corrente e per la posa della particella corrente.

Ogni *landmark n*-esimo viene sottoposto a tale procedimento per ogni osservazione. Al *landmark n*-esimo che che ha ricevuto una distanza di Mahalanobis minore di tutti gli altri, e se tale valore è minore di una soglia di minimo sufficiente, `GATE_REJECT`, viene associata questa osservazione. Se invece questo valore di distanza è maggiore sia della soglia minima `GATE_REJECT`, sia di una soglia più grande, `GATE_AUGMENT`, allora l'osservazione z rappresenta un nuovo *landmark*, osservato per la prima volta.

Mentre il robot avanza nell'ambiente il filtro particellare si arricchisce di nuovi *landmark* e viene nel contempo effettuato il processo di associazione di dati su essi. Per accelerare il processo di associazione di dati si è implementato un miglioramento di questa procedura utilizzando una associazione di dati locale. In questo modo l'associazione di dati viene effettuata su *landmark*, incorporati

nella particella, la cui posizione risiede nel campo visivo del laser; non verrà quindi valutata l'associazione di dati tra l'osservazione e un *landmark* che è troppo lontano dal robot o è fuori dall'angolo piano che viene spazzato dal sensore di osservazione. Un avvertibile miglioramento prestazionale è stato rilevato sopratutto in casi in cui la funzione `get_observation` restituisce numerose osservazioni.

5.3.6 Calcolo dei pesi

Il calcolo dei pesi per ogni particella viene effettuato nella funzione `compute_weight`. Vengono calcolati 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*:

$$S = H_v \, P_v \, H_v^T + S_f \tag{5.22}$$

infine il peso di ogni particella viene calcolato come:

$$w = w_{old} \cdot p(z_t | s_{t-1}) \tag{5.23}$$

dove:

$$p(z_t | s_{t-1}) = \frac{e^{-\frac{1}{2} v^T S^{-1} v}}{2 \pi \sqrt{\det(S)}} \tag{5.24}$$

5.3.7 Il resampling

Il processo di *resampling* è effettuato dalla funzione `resample_particle`. Questa si occupa di effettuare il *resampling* se si è raggiunta una sufficiente diversità particellare. La decisione viene effettuata tramite una operazione di sogliatura. L'assegnazione dell'adeguato valore di soglia è una operazione delicata: se tale soglia la si imposta ad un valore troppo alto si ottiene una bassa diversità particellare a causa della frequente operazione di eliminazione di particelle con basso valore di peso. Se, invece, la si imposta ad un valore troppo basso la diversità particellare nel filtro sarà alta, ma a causa di una poco frequente procedura di

resampling le particelle con basso peso avranno una vita maggiore, provocando errori nella costruzione della mappa.

In questa funzione Matlab il processo di *resampling* viene effettuato tramite l'uso di una metodo di *random resampling*, implementato nella `stratified_resample`, che utilizza un campionamento sulla base dei pesi delle particelle. Il numero di particelle totali del filtro rimane lo stesso, ma accade che le particelle con basso peso vengono eliminate per far posto a duplicati di particelle di peso maggiore. Dopo aver effettuato questo processo a tutte le particelle del filtro così rinnovato, viene assegnato un peso omogeneo, pari a $w = 1/M$, con M il numero totale di particelle. Il *resampling* delle particelle viene completato dalla funzione *resample_obs* che associa le corrette osservazioni al nuovo *set* di particelle, dopo il loro processo di *resampling*.

5.3.8 Sample proposal

La funzione `sample_proposal` computa la *proposal distribution* mostrata nella sezione 4.3.1. Viene qui calcolata una nuova posizione x_v e una aggiornata P_v per ogni particella a cui una o più *feature* associate ad essa sono state riosservate.

Per ogni particella di queste si utilizza la `compute_jacobians` per calcolare le matrici $z_{p,n}$,$H_{v,n}$,$H_{f,n}$ e $Q_{f,n}$ per ogni *feature n*-esima riosservata in esame. Quindi si effettua il calcolo dell'innovazione $v_n = z - z_{p,n}$, ove z indica la misura dell'osservazione. È possibile calcolare la *proposal* delle pose x_{vp} e delle matrici di covarianza P_{vp} dalla *proposal distribution* di queste particelle:

$$P_{vp} = (H_{v,n}^T \, Q_{f,n}^{-1} \, H_{v,n} + P_{vp})^{-1} \tag{5.25}$$

$$x_{vp} = x_{vp} + P_{vp} \, H_{v,n}^T \, Q_{f,n}^{-1} \, v_n \tag{5.26}$$

Infine, si campiona dalle *proposal* x_{vp} e P_{vp} utilizzando una multivariata gaussiana (implementata nella funzione `multivariate_gauss`), ottenendo una nuova posa per le particelle in esame.

5.3.9 Aggiornamento della posa dei landmark

Le nuove pose particellari estratte dalla *proposal distribution* per le particelle del filtro che hanno riosservato un *landmark*, si assumono essere esatte. Ogni aggiornamento della posizione del *landmark* può essere calcolato indipendentemente, senza considerare errore sulla posa del robot. L'aggiornamento della posizione del *landmark* è effettuato tramite un filtro di Kalman.

Per ogni *landmark* riosservato per la particella *n*-esima, viene calcolato tramite la funzione `compute_jacobians` l'innovazione v_n e il jacobiano $H_{f,n}$. Per esigenze di stabilità numerica è stato implementato il filtro di Kalman Esteso discreto con fattorizzazione di Cholesky [16], più stabile rispetto agli altri comuni metodi di implementazione. Esso effettua il calcolo dell'aggiornamento della posa del *landmark* tramite lo stato precedente del *landmark* θ_n, la matrice di covarianza della *feature*, l'innovazione v_n, la matrice R di covarianza del modello di misura e il modello di osservazione linearizzato, costituito proprio dallo jacobiano $H_{\theta,n}$. Il risultato restituito da questa funzione è la posa e la covarianza aggiornata della *feature* in esame. Il filtro di Kalman che utilizza la fattorizzazione di Cholesky è descritto dalle:

$$P_{H_{\theta,n}} = P\,H_{\theta,n}^T \tag{5.27}$$

$$S = H_{\theta,n}\,P_{H_{\theta,n}} + R \tag{5.28}$$

$$S = \frac{1}{2}(S + S^T) \tag{5.29}$$

$$\tag{5.30}$$

In tale modo S risulta essere una matrice simmetrica:

$$S_{Cholesky} = Cholesky(S) \tag{5.31}$$

La fattorizzazione di Cholesky fattorizza la matrice S di dimensione $K \times K$, definita positiva, nel prodotto di una matrice triangolare inferiore e della sua trasposta $S = LL^T$. Questo tipo di fattorizzazione è utilizzata per prendere vantaggio della sparsità della matrice S. Vengono calcolati gli aggiornamenti di

$x_{f,n}$ e $P_{f,n}$ eseguendo:

$$\theta_n = \theta_n + P_{H_{\theta,n}} S_{Cholesky} S^{-1}_{Cholesky} v_n \quad (5.32)$$

$$P_{\theta,n} = P_{\theta,n} - P_{H_{\theta,n}} S_{Cholesky} S^{T}_{Cholesky} P_{H_{\theta,n}} \quad (5.33)$$

5.3.10 L'aggiunta dei landmark

La funzione `add_feature` aggiunge nuovi *landmark* alle particelle dalle osservazioni effettuate se la funzione `data_association_unknown`, addetta all'associazione di dati, lo ha ritenuto necessario. La posa del nuovo *landmark* viene stabilita soltanto in base all'osservazione, senza incertezza di posa del robot, proprio perché è un procedimento effettuato su base particellare, e proprio ogni particella rappresenta una possibile posa del robot. Per cui la posa del nuovo *landmark*, espressa in coordinate cartesiane assolute (si ricorda che il vettore delle misure di osservazione $z = \langle z_d, z_\theta \rangle$ è espresso in coordinate polari relative alla posa del robot), viene calcolata con le:

$$\theta_{n+1} = \begin{bmatrix} x_{v,x} + z_d\, cos(x_{v,\alpha} + z_\theta) \\ x_{v,y} + z_d\, sin(x_{v,\alpha} + z_\theta) \end{bmatrix} \quad (5.34)$$

(5.35)

La covarianza $P_{\theta,n+1}$ non viene inizializzata con metodi di stima, bensì è preferibile assegnarla con un metodo algebrico, in modo da far modificare il suo valore dalle osservazioni successive, basandosi sul principio che non si hanno forti certezze sulla posizione dalla prima osservazione di un *landmark*; osservazioni successive restringeranno questa l'incertezza e quindi andranno ad agire su $P_{\theta,n+1}$, con precisione. La matrice di covarianza associata alla nuova *feature* $P_{\theta,n+1}$ viene calcolata tramite le:

$$G_z = \begin{bmatrix} \cos(x_{v,\alpha} + z_\theta) & -z_d\, \sin(x_{v,\alpha} + z_\theta) \\ \sin(x_{v,\alpha} + z_\theta) & z_d\, \cos(x_{v,\alpha} + z_\theta) \end{bmatrix} \quad (5.36)$$

$$P_{\theta,n+1} = G_z\, R\, G_z^T \quad (5.37)$$

5.3.11 L'algoritmo

L'architettura modulare sviluppata nell'implementazione del FastSLAM in Matlab semplifica il processo di scrittura del sorgente per il robot. L'algoritmo del FastSLAM è descritto nel diagramma di flusso in figura 5.3. È importante notare che il FastSLAM propriamente detto, è eseguito soltanto quando avviene una lettura del sensore laser virtuale, per il tempo rimanente le particelle vengono aggiornate tramite la funzione `predict`. L'attesa per la lettura del sensore virtuale viene implementata tramite un contatore che è incrementato per ogni ciclo di programma, quando esso raggiunge un valore fissato `DT_OBSERVE` inizia il processamento proprio del FastSLAM. A fine algoritmo viene creata la mappa dalle complete letture del laser basandosi sulla posa della particella con peso maggiore; questo metodo inizialmente sviluppato in MatLab viene raffinato nella implementazione su robot e verrà descritto nella sezione 7.4.

5.3.12 La piattaforma di sviluppo Matlab-C

Per completezza è importante menzionare il metodo in cui il sorgente Matlab è stato tradotto in linguaggio C, applicabile sull'architettura del robot. Le funzioni Matlab trasferiscono e processano tra loro grandi moli di dati, per una verifica palpabile di una implementazione in linguaggio C riguardante la precisione e la correttezza del sorgente scritto, è stato sviluppata una piattaforma di comunicazione su rete TCP-IP per trasferire dati dal sorgente Matlab verso l'eseguibile posto sul robot. In sostanza si è sfruttata la modularità con cui si è scritto il programma in Matlab per sviluppare in linguaggio C: si è riscritta ogni funzione Matlab in linguaggio C, quindi la si è verificata passando i dati via rete TCP-IP dalla simulazione Matlab all'eseguibile compilato su un altra macchina, i dati vengono quindi processati da questa funzione e ritrasferiti con la stesa tecnica via rete verso la simulazione Matlab. È stato così possibile verificare la correttezza di ogni singola funzione sostituendo progressivamente tutto il sorgente Matlab in C, verificando con facilità la correttezza dei dati, fino a che la simulazione Matlab era costituita soltanto dal modello dei sensori, il modello di moto

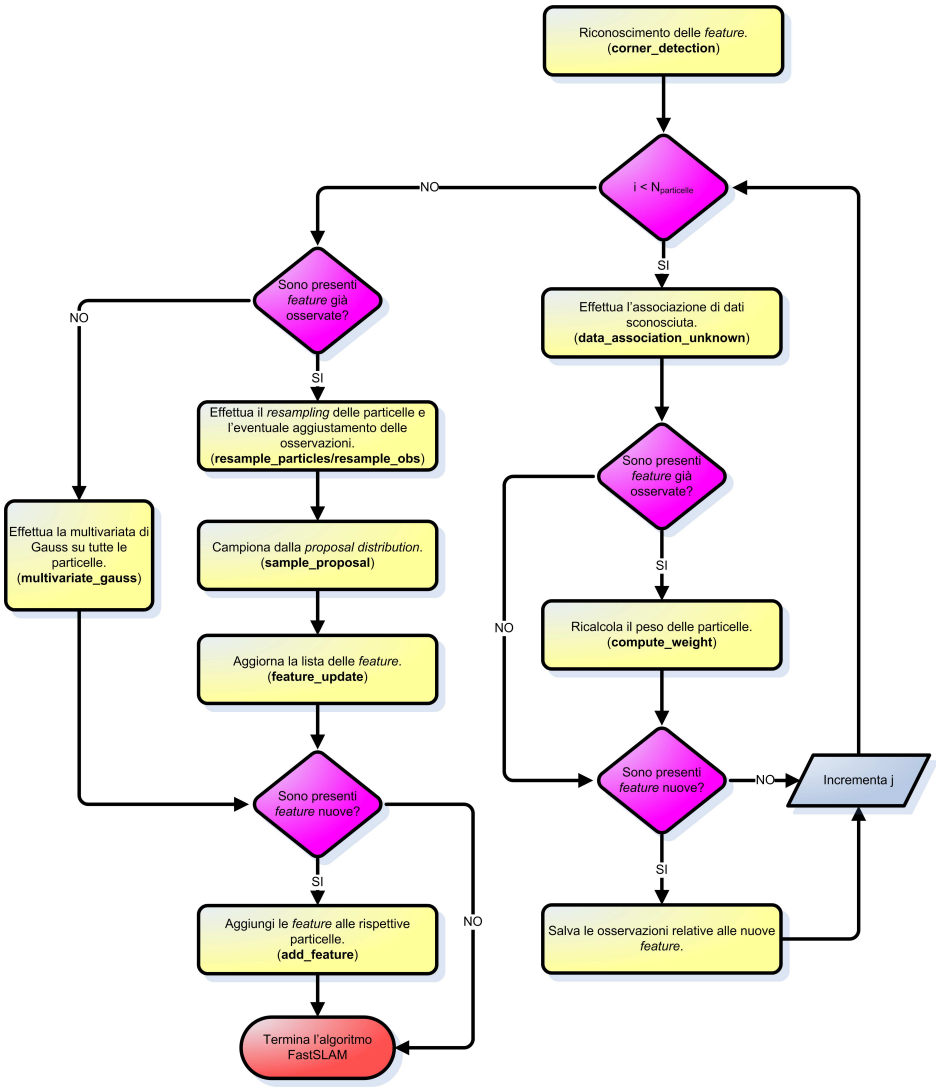


Figura 5.3: *Diagramma di flusso dell'algoritmo del FastSLAM.*

e la visualizzazione dell'andamento dello SLAM, cioè tutto il funzionamento del FastSLAM era totalmente stato trasferito in linguaggio C, si veda la figura 5.4. In Matlab lo strato di comunicazione su rete TCP-IP è implementato tramite le funzioni `send_to_linux`, che si occupa di trasferire dati verso l'eseguibile remoto, e `receive_from_linux`, che riceve i dati processati e li restituisce alla simulazione.

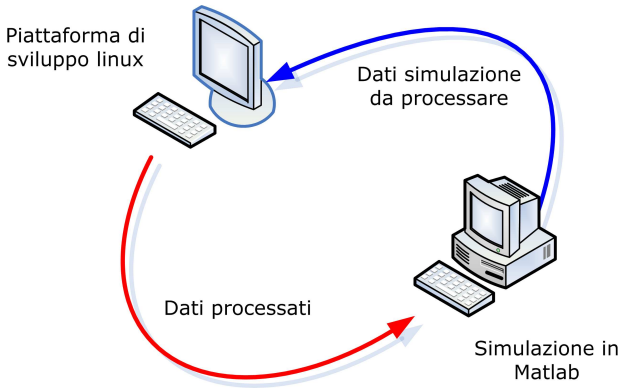


Figura 5.4: *Schema indicativo del procedimento di porting.*

5.3.13 Risultati simulativi

Vengono qui esposti i risultati simulativi delle simulazioni effettuate in Matlab del FastSLAM. L'ambiente virtuale di simulazione viene automaticamente e progressivamente quantizzato per essere utilizzato come input di dati per il modello di laser virtuale, si veda la figura 5.5. Diverse prove sono state effettuate per verificare i procedimenti teorici e per riuscire ad ottenere il risultato simulativamente migliore.

Per ogni prova è stato cambiato un solo parametro per studiarne gli effetti sulla simulazione. Sono stati assegnati dei valori verosimili, rispetto al robot, ai vari parametri che si utilizzano per la simulazione cioè: σ_V e σ_ω (rispettivamente le varianze della velocità traslazionale e rotazionale, presenti sulla diagonale della matrice diagonale Q), σ_R e σ_B (varianze di distanza e orientamento del sensore

laser, presenti sulla diagonale della matrice diagonale R), numero di particelle presenti nel filtro, livello di resampling, `GATE_REJECT` e `GATE_AUGMENT` relativi all'asssociazione di dati.

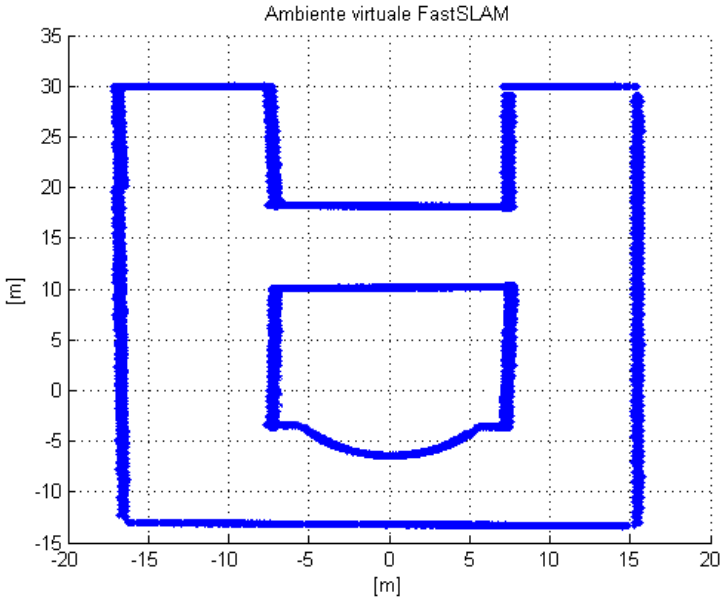


Figura 5.5: *Mappa completa dell'ambiente virtuale quantizzato utilizzato per la simulazione.*

Nei risultati simulativi qui mostrati le tracce rosse corrispondono al percorso effettuato dalle singole particelle, i segni neri rappresentano il percorso effettuato dalla particella con peso maggiore (cioè la particella che ha ottenuto maggiore fiducia nel filtro, e che quindi rappresenta la migliore stima che il FastSLAM riesce a restituire sulla posizione del robot), i segni verdi, invece, rappresentato la traiettoria reale del robot. I punti blu rappresentano la posizione stimata, per ogni particella, dei vari *landmark*. In ogni prova simulativa è descritto dal robot un giro completo dell'ambiente virtuale con coordinate di partenza $\langle 0, -10 \rangle$.

I parametri di base, diversamente variati durante ogni simulazione, sono:

- $\sigma_V = 0.1$

- $\sigma_\omega = 3.0$
- $\sigma_R = 0.1$
- $\sigma_B = 1.0$
- Numero di particelle: 100
- Livello di resampling: 90%
- `GATE_REJECT`$= 9$
- `GATE_AUGMENT`$= 5000$

Come prima prova simulativa si è aumentata l'incertezza sulla velocità angolare σ_ω, quindi l'incertezza della posa, in modo che $\sigma_\omega = 6$, il risultato della prova è mostrato in figura 5.6. La prova evidenzia come l'aumento dell'incertezza sulla velocità angolare corrisponde a un aumento del raggio del filtro, le particelle sono cioè distribuite in un area maggiore durante il percorso. Si nota anche che le posizioni stimate dei vari *landmark* soffrono dell'incertezza della posa: le varie stime, in blu, dei *landmark* sono infatti disposte in un area arcuata, con conseguenti problemi sull'associazione di dati. Nonostante la traiettoria effettutata durante il percoso sia lontana da quella reale, il robot riesce comunque a chiudere il giro, e a risistemare conseguentemente la propria stima di posizione. Appena richiude il giro, il *loop*, elimina, attraverso il *resampling*, le particelle, e quindi anche i *landmark* ad esse associate, che non sono concordi con la posa corretta.

Si è quindi studiato l'effetto sul risultato dell'ambiguità della associazione di dati, modificando i due parametri che agiscono sulla funzione di associazione di dati sconosciuta: `GATE_REJECT` e `GATE_AUGMENT`. I due risultati sono rispettivamente esposti in figura 5.7 e 5.8.

Si ricorda che `GATE_REJECT` rappresenta la massima distanza di Mahalanobis entro cui un'osservazione viene associata a un *landmark* presente nel filtro;

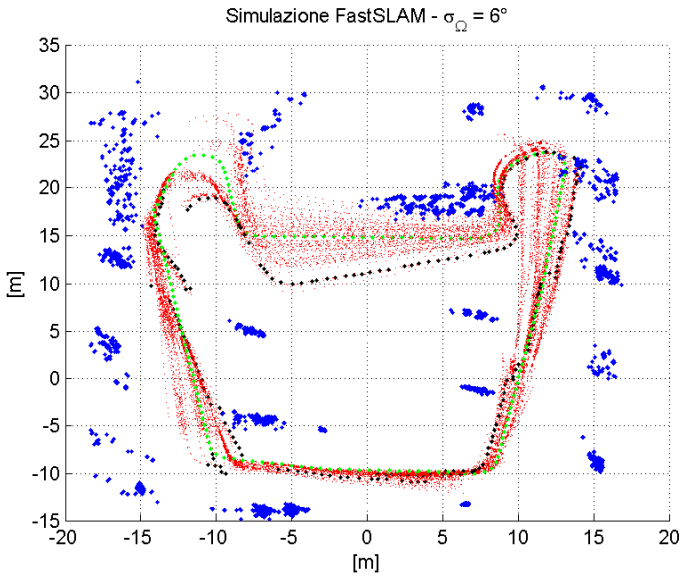


Figura 5.6: *Risultato della simulazione con filtro particellare afflitto da forte incertezza sulla velocità angolare,* $\sigma_\omega = 6^o$.

`GATE_AUGMENT` rappresenta il minimo valore di distanza di Mahalanobis per cui deve essere incorporato nel filtro una osservazione, creando un nuovo *landmark*.

Avendo aumentato il valore di `GATE_REJECT`$= 100$ si è aumentata l'ambiguità sull'associazione di dati: in questo modo è possibile erroneamente confondere osservazioni di *landmark* vicini che generano simili distanze di Mahalanobis; oppure può accadere di incorporare un solo nuovo *landmark* nel filtro particellare da osservazioni di *landmark* mai osservati tra loro contigui. Nella mappa è visibile una diminuzione del numero di *landmark*, ma la distribuzione delle particelle del filtro rimane compatta e sufficientemente precisa durante tutto il percorso. L'ambiguità sull'associazione di dati è rivelata dai "salti" della traccia nera della particella migliore rispetto alla traiettoria reale, che causa repentine correzioni sulla stima della posa in base alle osservazioni.

Diminuire il parametro `GATE_AUGMENT`$= 2000$ provoca la non chiusura del *loop*, questo infatti causa troppe ambiguità nell'associazione di dati; in tale modo vengono creati nuovi *landmark* da misure spurie. Particelle con poco peso

possono in tale modo creare nuovi falsi *landmark* e successivamente confermare questi falsi *landmark* con osservazioni spurie successive, aumentando di peso e facendo progressivamente divergere il filtro a causa di *resampling* mal condizionati. Il *loop* non viene chiuso poichè riosservando i primi *landmark* essi vengono considerati nuovi e non riassociati a quelli all'inizio incorporati del filtro, per cui la stima non viene corretta e il *loop* non si chiude.

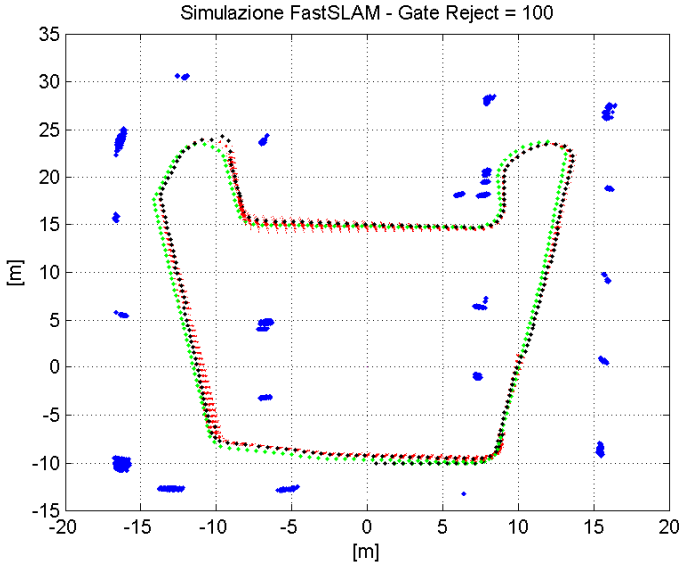


Figura 5.7: *Risultato della simulazione con il parametro* `GATE_REJECT` $= 100$.

Un'altra prova effettutata è stata quella di diminuire il livello di *resampling*, ponendolo al 10%. Tenere un livello basso di *resampling* significa dare poca fiducia al filtro e aumentare il tempo tra due *resampling* successivi, cioè aumentare il tempo di vita delle particelle con peso basso. In figura 5.9 è esposto il risultato della simulazione. È evidente come la distribuzione particellare risulti aumentare di grandezza durante percorso, ma quando si completa il giro , quando cioè si riosservano i primi *landmark*, la distribuzione collassa e raggiunge la posizione corretta, effettuando un *resampling* accurato. Tenere in vita per lungo tempo le particelle può dimostrarsi una strategia corretta in ambienti molto rumorosi, effettuando *resampling* solo quando si è assolutamente certi di farlo.

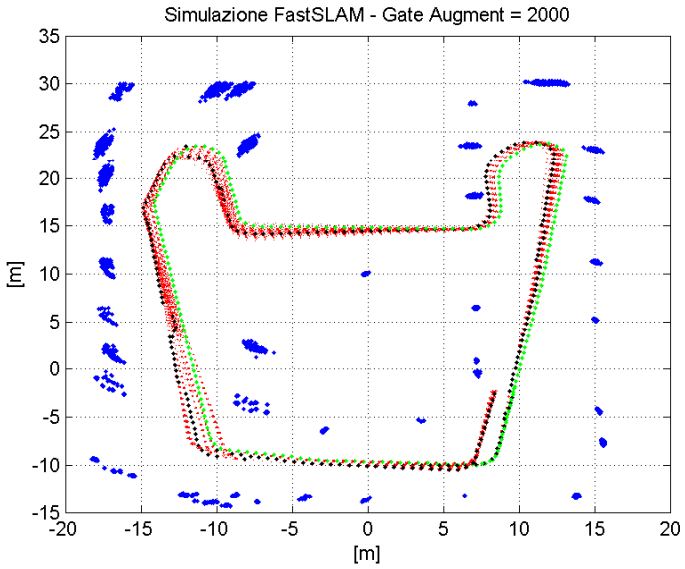


Figura 5.8: *Risultato della simulazione con il parametro* `GATE_AUGMENT` $= 2000$.

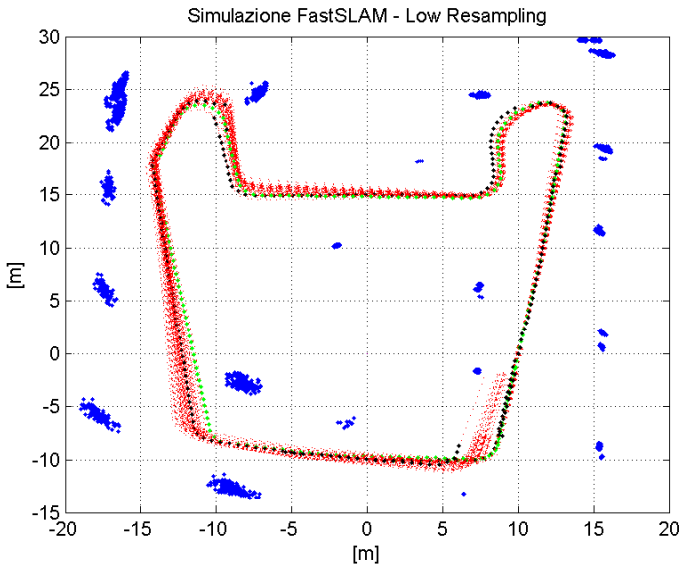


Figura 5.9: *Risultato della simulazione utilizzando un livello di resampling* 10%.

È esposta in figura 5.10 la prova di FastSLAM, effettuata con i valori di base, che dimostra la potenza di questo algoritmo, ottenendo durante tutto il percorso uno scarto di posizione molto basso, e di conseguenza un basso errore sulla stima dei *landmark*. Notevole è notare la compattezza della distribuzione del filtro particellare, e come la stima migliore, cioè la posizione della particella con maggior peso, sia spesso collimata o vicina alla traiettoria reale.

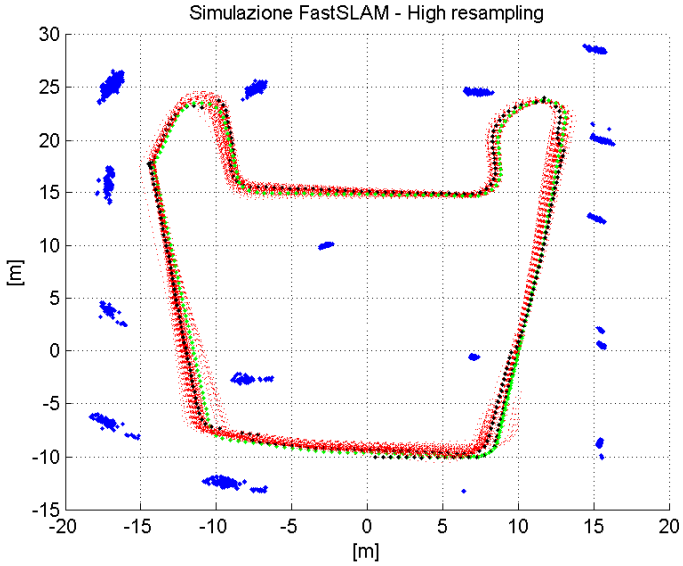


Figura 5.10: *Risultato della simulazione ottenuto con i valori di base.*

Capitolo 6

Implementazione del FastSLAM su XR4000

L'effettivo passo per la realizzazione pratica del FastSLAM è la sua implementazione su robot mobile. Il robot mobile su cui è stato sviluppato e implementato è il Nomad XR4000 della Nomadic Technologies. La struttura tecnica di questo robot verrà introdotta in questo capitolo.

Per lo sviluppo di questa metodica di SLAM sono state create apposite strutture di dati, adatte a un modello dinamico come quello del filtro particellare, e una organizzazione modulare del codice sorgente in modo da garantire una velocità computazionale adatta ad un sistema reale. Oltretutto per raggiungere lo scopo di rendere il robot autonomo è stato implementato anche l'algoritmo di controllo, l'*obstacle avoidance*, e l'algoritmo di esplorazione.

6.1 Il robot 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. Il Nomad XR4000 è costituito da un computer basato su processore Intel

Figura 6.1: *Vista frontale Nomadic Nomad XR4000.*

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 robot è un prisma a 24 facce di diametro $62cm$ e altezza $85cm$, sollevato da terra da quattro ruote di

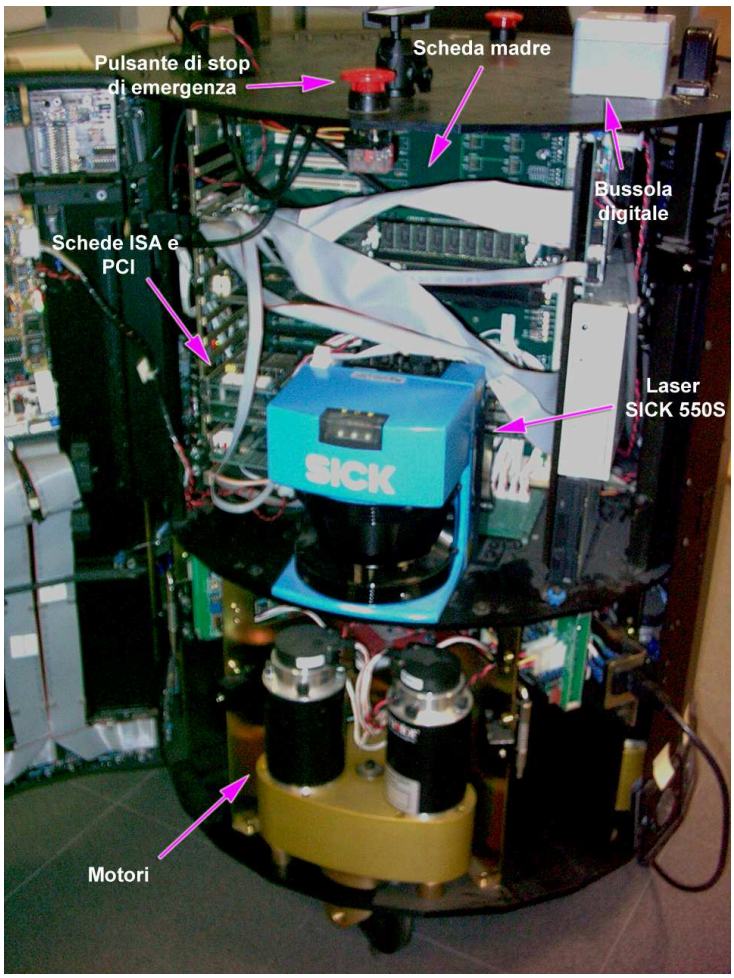


Figura 6.2: *L'interno del Nomad XR4000.*

metallo stampato. Pesa circa $115Kg$ al netto delle sue quattro batterie[1] per un peso complessivo di $160Kg$.

6.1.1 Sistema alto livello Intel Pentium III

All'interno del Nomad XR4000, è montata in verticale, saldamente ancorata a dei sostegni metallici, una particolare scheda stampata dotata di 4 slot ISA e

[1]Ogni batteria pesa 12kg.,

14 slot PCI. La scheda madre Portwell Robo-638Z è una scheda madre Pentium II/III utilizzata in processi industriali; poco più lunga di una scheda di espansione, essa occupa un particolare alloggiamento della scheda verticale. Questa scheda madre è molto compatta e ha un ottimo livello di integrazione, infatti su un così piccolo spazio è presente il doppio controller IDE UDMA/33, quattro slot per la RAM DIMM PC100, una scheda grafica AGP integrata, un controller ethernet, nonché porte seriali e parallele.

Sulla scheda madre è presente un banco RAM da 128 MB. Il processore è una CPU Intel Pentium III a 500 Mhz[2]. Le unità di archiviazione di massa sono costituite da un hard disk da 20GB e un CD-ROM 24x. Questa è una configurazione che permette prestazioni computazionali assolutamente idonei ad applicazioni di robotica mobile e di fusione sensoriale.

Sulla scheda madre è integrata una scheda Ethernet Intel EtherExpress PRO 10/100 (per il collegamento alla rete tramite un comune cavo CAT-5 con connettore RJ-45) e una scheda video AGP ATI Mach64 8MB.

Un alloggiamento PCI è occupato dalla scheda di acquisizione, cuore del sistema di visione, basata sul chip Brooktree 878 che permette acquisizioni real-time fino a 30fps, con una massima risoluzione di 640x480. Le entrate disponibili sono RC e S-Video. Sull'entrata RC è montata una telecamera a colori CCD Hitachi KP-D50, dotata di ottica Pentax regolabile manualmente (fuoco / otturatore) con un'ottima resa visiva.

Uno slot ISA è occupato dalla scheda di sintesi vocale DoubleTalk (*text-to-speech*), dotata dell'apposito regolatore di volume e collegata a un piccolo altoparlante.

Su un altro slot ISA è installata la scheda *radio-lan* Proxim Rangelan2 7100, collegata all'apposita antenna (posizionata all'esterno del robot) che permette il collegamento *wireless* verso altre postazioni compatibili. Sul lato di accesso della

[2]Originariamente la CPU Pentium II a 450Mhz, è stata sostituita per esigenze computazionali.

scheda sono presenti dei *dip-switch* per la configurazione dell'indirizzi di *I/O* e delle spie di indicazione di attività.

Un altro slot ISA è occupato da una scheda Nomadic Technologies *Intellisys 200 '12 axis servo controller'* che si occupa del controllo del motore e del processamento dei dati degli *encoder.*

L'ultima, ma non meno importante, scheda ISA presente è la scheda di interfacciamento con la rete *XR SynapseNet distributed network*, essa funge da ponte di comunicazione tra il sistema sensoriale di base e la scheda madre, in questo modo è possibile comunicare e ricevere dati l'uno dall'altra.

6.1.2 XR C8 Holonomic Drive System

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.

Nella *tabella 6.1* è possibile valutare alcune caratteristiche tecniche del motore.

Max accel. traslazionale	$5m/s^2$
Max accel. rotazionale	$2rad/s^2$
Risoluzione encoder traslazione	$1120percm$
Risoluzione encoder rotazione	$422pergrado$

Tabella 6.1: *Caratteristiche tecniche del motore.*

6.1.3 Sistema sensoriale XR4000

Il *sistema sensoriale di base* del Nomad XR4000 consta di tre tipi di sensori:

- Sensori tattili a due livelli Sensus 150^{TM}
- Sensori di prossimità ad ultrasuoni Sensus 250^{TM}
- Sensori di prossimità ad infrarosso Sensus 350^{TM}

Questi sensori fanno parte e costituiscono la rete *SynapseNet distributed network* interna al robot.

Il *sistema sensoriale avanzato* è costituito da:

- Sistema Sensus 600 di orientamento
- Sistema laser di rilevamento degli ostacoli S550
- Sistema di visione

Il sistema sensoriale avanzato è collegato direttamente al pc di bordo, tramite collegamenti su porta seriale e non fa parte della rete *SynapseNet distributed network*.

Verranno qui spiegati solo i sensori utilizzati nel robot mobile per il progetto di tesi. Per una dettagliata spiegazione del restante sistema hardware, e alle diverse configurazioni possibili, si rimanda al manuale [20].

Sensori di prossimità ad infrarosso Sensus 350^{TM}

Il sistema Sensus 350^{TM} 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 ($30 - 50cm$). 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 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[3]. 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 alla 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

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

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 6.3 mostra i livelli di energia misurati da $0cm$ a $90cm$, per angoli da 0^0 a 75^0, le misure sono state effettuate al buio su un foglio di comune carta per fotocopiatrice.

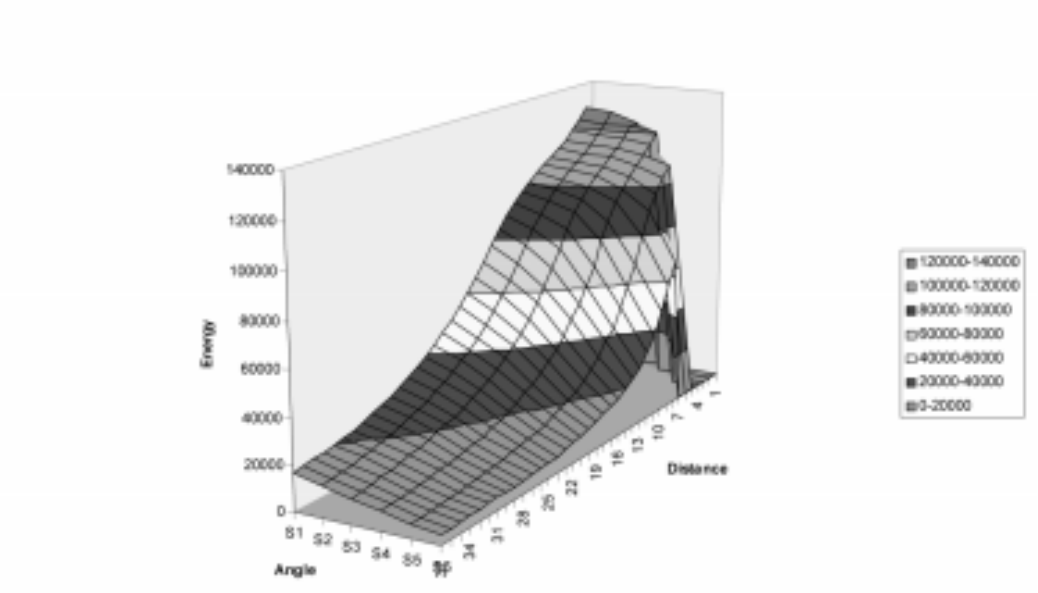


Figura 6.3: *Grafico energia / angolazione / distanza su bersaglio di superficie opaca al buio.*

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 6.4 sono stati rilevati in un ambiente illuminato con luce fluorescente, i dati di figura 6.5 invece con luce incandescente; comparando questa con la figura 6.3 è visibile quanto i valori energetici siano poco influenzati dalla luce fluorescente, poichè contiene una piccola porzione di spettro infrarosso.

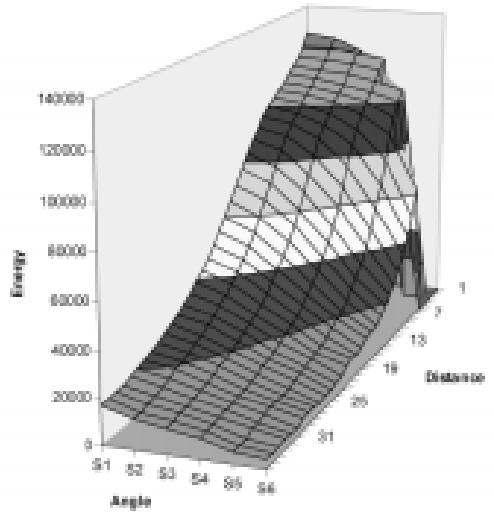


Figura 6.4: *Grafico energia / angolazione / distanza su bersaglio di superficie opaca in luce fluorescente.*

colore e caratteristiche superficiali : il colore dell'oggetto, o più precisamente il coefficiente di riflessione nello spettro infrarosso, influenza la precisione della misurazione, così come le caratteristiche superficiali dell'oggetto: superfici lucide evidenziano un picco energetico per angoli vicini alla normale dell'asse del ricevitore[4]. Materiali come cemento o tessuto assorbono molta più energia radiante. Il grafico di figura 6.6 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[5] non necessariamente assorbono luce infrarossa e viceversa.

[4] Cioè hanno un comportamento speculare.
[5] Chiamati anche *black objects*.

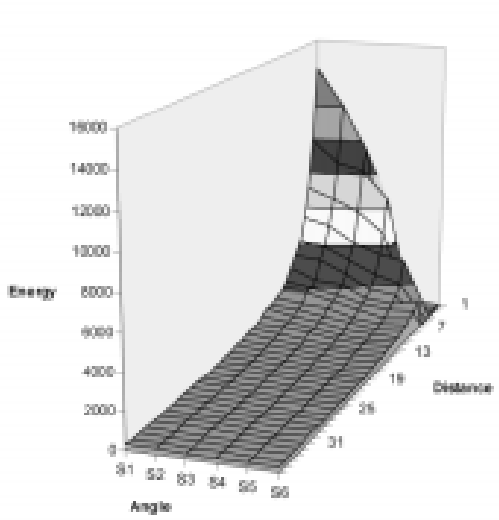


Figura 6.5: *Grafico energia / angolazione / distanza su bersaglio di superficie opaca in luce incandescente.*

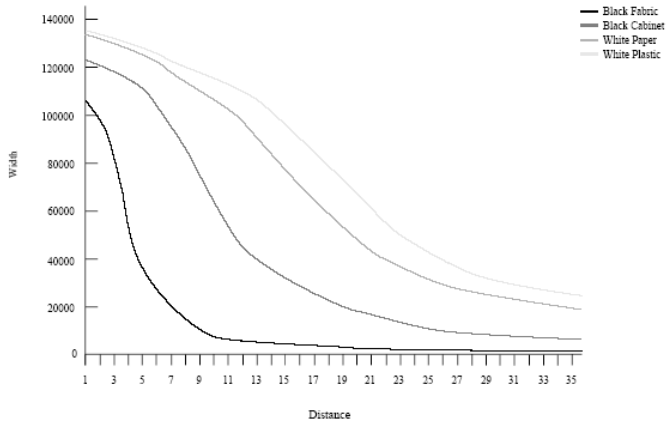


Figura 6.6: *Grafico energia / distanza ad angolazione 0: materiale bianco / nero, materiale opaco / lucido.*

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 planare ad incrementi di 0.5^0, complessivamente 360 campioni. Ogni semipiano completo di rilevazione avviene con frequenza $20Hz$. Il sistema laser Sick Sensus 550 è collegato alla scheda madre in maniera diretta tramite collegamento seriale.

Sistema Sensus 600 di orientamento

Il sistema Sensus 600 è basato su una bussola digitale KVH-C100 che è un componente industriale di grande affidabilità, compattezza e precisione. È in grado di misurare i decimi di gradi ed ha una accuratezza di 0.5^0 anche a latitudini vicine ai poli[6]; anche grazie alle sue generose temperature di esercizio, da -25^0C a 70^0C, è stato impiegato in applicazioni critiche (come robot sottomarini) e in ambito militare.

Il KVH-C100 si basa su una bussola a flussometro digitale capace di misure molto precise anche se montata non in orizzontale, o in ambienti con forti disturbi elettromagnetici. La bussola è montata esternamente in un contenitore metallico blindato e collegata alla scheda di interfacciamento *Nomadic compass interface* da cui, tramite una piattina, è connessa alla seconda porta seriale[7] della scheda madre. Il sistema è configurato per comunicare a 9600 baud, 8 bit di dati, nessun bit di parità e 1 bit di stop.

[6] Si rimanda per completezza *Magnetic Dip Angle Effects on Accuracy* del manuale *Overview of Compass technology* del KVH-C100.

[7] COM2 [dos/win] o /dev/ttyS1 [*nix].

6.2 L'implementazione del FastSLAM in linguaggio C

Il primo passo progettuale per il *porting*[8] da linguaggio Matlab a C è stato uno studio sugli adeguati strumenti esitenti per lo sviluppo di *software* matematico matriciale in ambiente Linux[9]. Diverse sono state le librerie candidate, ma ci si è indirizzati verso le più robuste, basate su solide basi di calcolo numerico. La libreria matematica prescelta è stata la *GNU Scientific Library* (GSL) [7], ampiamente documentata e largamente utilizzata in ambito scientifico, nonché *freeware* cioè liberamente utilizzabile. Questa libreria matematica comprende numerose funzioni algebrico-matriciali, gestioni di operazioni su vettori, matrici, permutazioni e generazione di numeri randomici, tutto in maniera molto semplice ed elegante. L'altra libreria utilizzata è stata la libreria di sviluppo del robot $DRobot$, con la quale è possibile l'interfacciarsi con l'hardware del robot.

6.2.1 La struttura di dati

Le strutture di dati sono state progettate con attenzione in modo da essere utilizzate con flessibilità nel corso del programma. Nel dettaglio, ogni particella è costituita da una struttura di dati:

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

    double w;                  // Particle weight
```

[8] Termine tecnico con cui si indica lo sviluppo di uno stesso programma da un linguaggio di programmazione ad un altro.

[9] Sul robot è montato RedHat Linux 7.3

```
    int Nf;                     // Number of features

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

La struttura *particle* contiene il suo vettore di posa `xv`, la sua matrice di covarianza `Pv`, il peso `w`, il numero delle *feature* presenti `Nf`. I vettori e le matrici sono definite dal tipo di dato `gsl_matrix` della libreria *GSL*. È importante notare che ogni particella contiene un puntatore alla lista delle *feature* associate ad essa. Il filtro particellare è definito come un array di strutture di dimensione fissata al numero di `NPARTICLES`:

```
struct particle Particles[NPARTICLES];.   // Particle filter
```

Si ricorda che il numero di particelle presenti nel FastSLAM è costante durante tutta l'esecuzione. Le *feature* associate ad ogni particella sono state implementate come liste di strutture. Le liste sono delle strutture di dati allocate dinamicamente e collegate l'un l'altra. Il vantaggio risiede nel poter modellare al meglio un processo di allocazione dinamica di memoria come il filtro particellare persente nel FastSLAM. Basti pensare a quando è necessario aggiungere una *feature* ad una determinata particella. In questo modo basta semplicemente aggiungere alla lista delle *feature* un nuovo elemento. Lo svantaggio rappresentato dalle liste risiede nella maggiore complessità di gestione. È necessario infatti scrivere delle funzioni addette all'aggiunta, alla cancellazione, alla ricerca, al conteggio dei moduli della lista. La lista di strutture che rappresentazione le *feature* per ogni particella è così definita:

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

};
```

Ogni struttura contiene la sua matrice di posa `xf`, la propria matrice di covarianza `Pf`, e le matrici `zp`, `Hv`, `Hf`, `Sf`, utili a calcoli in alcune funzioni dello SLAM. Sono presenti nella struttura anche i puntatori alla struttura successiva e precedente, necessari al collegamento tra i vari elementi della lista.

Con una lista di strutture è anche implementata la lista che raccoglie le osservazioni che la funzione relativa all'associazione di dati restituisce:

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

    int Index;

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

Dove `idf` contiene l'indice delle particelle delle quali *feature* associate sono state riosservate; `z` contiene le misure delle osservazioni di tali *feature*. La struttura è completata dai puntatori alla struttura precedente e successiva.

6.2.2 Il programma

Grazie alla architettura modulare sviluppata in Matlab si è portato in maniera efficiente il codice sorgente dal Matlab al linguaggio C. L'organizzazione del programma riflette, con le opportune modifiche dettate dalle diverse strutture di dati, il programma scritto in Matlab. Esso è composto dalle principali funzioni:

control() : Funzione addetta al controllo.

obstacle_avoidance() : Funzione addetta all'*obstacle avoidance*.

predict() : Funzione addetta alla predizione.

FastSlam_Engine() : Funzione addetta al FastSLAM.

6.2.3 `control()` e `obstacle_avoidance()`

Questa funzione si occupa dell'emissione dei riferimenti di controllo dei motori. Il controllo nel robot è costituito da segnali di riferimento di velocità/accelerazione traslazionale e velocità/accelerazione angolare. Questa *routine* di controllo viene eseguita dopo una aggiornamento della lettura del laser.

Nell'implementazione in un robot reale è di fondamentale importanza una procedura di *obstacle avoidance* per evitare ostacoli come muri od oggetti, oppure nel caso di ambienti dinamici, di persone che si muovono nell'ambiente.

Queste funzioni verranno descritte in dettaglio nel capitolo 7, in cui sarà spiegata la navigazione autonoma.

6.2.4 La predizione: `predict()`

Il processo di predizione viene svolto dalla funzione `predict()`. Questa ha un compito molto importante nell'intero funzionamento del FastSLAM, poiché essa si occupa dell'aggiornamento della posa di ogni particella nell'intervallo di tempo in cui non viene eseguito il FastSLAM perché si attendono nuove misure

dal sensore laser. La predizione aggiorna secondo la legge di moto descritta in sezione (5.10), la posa di ogni particella dal tempo $t-1$ al tempo t con un intervallo dt, con precisione $10^{-3}\,s$, che rappresenta il tempo intercorso tra due processamenti successivi. In ingresso alla predizione viene inserita l'accelerazione calcolata dall'algoritmo di controllo e le velocità lette direttamente dal controller del motore, per avere una quanto più possibile fedele velocità di moto; su questi dati vengono calcolate le nuove pose.

6.2.5 Il FastSLAM: `FastSlam_Engine()`

In `FastSlam_Engine()` sono implementate completamente tutte le funzioni che compongono la simulazione del FastSLAM in Matlab. L'esecuzione di questa funzione avviene soltanto quando viene effettuata una nuova misura del sensore laser SICK-S550, solo in questo caso è infatti possibile effettuare la correzione della posa del robot e l'aggiornamento della posizione dei *landmark*. In questa funzione viene eseguita l'estrazione delle *feature* dalle misure dei sensori, tramite la funzione `corner_detection()` IPAN99, descritta nella sezione 5.2.2, che restituisce la matrice z contenente tutte le *feature* trovate nell'ambiente. Quindi per base particellare si calcola l'associazione di dati attraverso `data_associate_unknown()`, e si generano due liste differenti: una per le *feature* che sono state riosservate, `old_feat()`, in cui vengono riportate quali sono state riosservate e le misure effettuate dal sensore su esse; l'altra per quelle nuove, `new_feat()`, in cui vengono segnate le misure effettuate su cui incorporare nuovi *landmark* nel filtro particellare. Se esistono osservazioni di *feature* già esistenti è necessario effettuare *resampling* tramite la funzione `resample_particles()`. Dopo aver effettuato il *resampling* la funzione `resample_obs()` associa coerentemente le corrette osservazioni al nuovo set di particelle. Quindi si effettua il *sample_proposal* e il *feature_update* dalle informazioni contenute nella lista `old_feat()`. La lista di strutture `new_feat()` viene utilizzata in `add_feature()` per aggiungere nuovi *landmark* alla particella.

Se non esistono osservazioni di nuove *feature* si effettua un operazione rando-

mica basata su una multivariata di gauss sulle pose delle particelle, per favorire la diversità del filtro particellare, nonchè si annullano le matrici `Pv`. Le liste `new_feat()` e `old_feat()` vengono disallocate al termine della funzione

6.3 Problemi tecnici e individuazione di soluzioni teoriche e pratiche

Nell'implementazione pratica si sono verificati diversi problemi legati ad alcune difficoltà incontrate durante le prove sperimentali.

Problemi numerici Un primo problema emerso è quello dovuto a problemi numerici della matrice di covarianza della posa P_v di una generica particella. La matrice P_v viene posta a zero ogni qual volta viene eseguita la funzione `multivariate_gauss()` per annullare l'incertezza di posa. Anche se questa è un operazione teoricamente corretta, si è verificato sperimentalmente che è causa di errore quando essa viene scomposta con la fattorizzazione di Cholesky, che avviene durante il processamento dell'EKF. La fattorizzazione di Cholesky è calcolabile soltanto su matrici definite positive, e quindi P_v risulta causa di questo errore. Per ovviare a questo problema si è introdotta una costante di stabilità numerica $\epsilon = 1 \cdot 10^{-7}$ che, moltiplicata per la matrice identità, costituisce l'inizializzazione di P_v, per cui:

$$P_v = \begin{bmatrix} \epsilon & 0 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon \end{bmatrix} \tag{6.1}$$

Con questo medodo di sicurezza numerica si evitano i problemi nella fattorizzazione di Cholesky. L'errore introdotto sulla matrice P_v in questo modo è comunque trascurabile, poiché l'incertezza è dell'ordine di $10^{-7}\, m$.

Errori odometrici Un altro problema è stato rivestito dall'errore effettuato nel modello di moto. L'errore nell'odometria è composizione di errore sistematico e di uno non sistematico. A.Martinelli [12] mostra come modellizzare l'errore sistematico con due parametri e con altri due quello non sistematico. L'errore non sistematico viene contemplato nelle matrici di covarianza. Per risolvere il problema dell'errore sistematico nell'odometria, costituito da slittamenti e pattinamenti, Montemerlo [12] suggerisce o di modellizzare l'errore sistematico con una dinamica da introdurre nel modello di moto oppure di utilizzare sensori adatti per misurarlo. In questo modo la posa risulta soltanto affetta da errori non sistematici, tutti contemplabili in una distribuzione normale. Il robot Nomad XR4000 è fortemente affetto da errori sistematici sull'odometria, e per correggere questo problema sono state percorse ambedue le strade suggerite.

Nel robot Nomad XR4000, la componente dell'errore sistematico risulta essere un fattore molto influente. La massa del robot è distribuita in maniera non uniforme, per cui esiste una pressione differente sui punti di contatto tra il pavimento e le due ruote più esterne. Questo provoca durante la procedura di camminamento uno sbandamento del veicolo. Il robot presenta anche un movimento sussultorio a causa di questa cattiva distribuzione di peso. Probabilmente dovuto a cause similari è un movimento di rotazione del mezzo durante un moto traslatorio: il robot percorre una traiettoria curva, a causa dello sbandamento, e nel contempo cambia lentamente orientamento tramite una lenta rotazione. Questa rotazione sembra essere provocata da più fattori, tra cui l'orientamento delle quattro ruote durante un moto. Il moto è ben illustrato in figura 6.7.

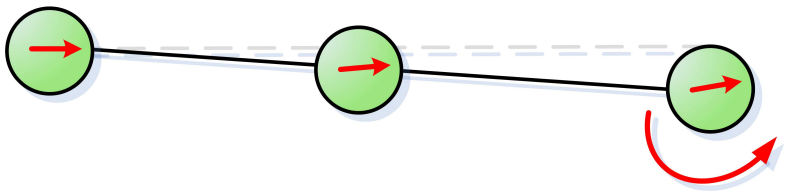


Figura 6.7: *Traiettoria ideale del robot (linea tratteggiata) e suo comportamento reale. Il robot non procede lungo la rotta designata e cambia il suo orientamento.*

Oltre a questi problemi di natura meccanica il robot non risulta rispondere in maniera corretta alle velocità assegnate al *controller* del motore, risulta infatti procedere a velocità traslazionale e rotazionale minore.

È stata tentata una modellizzazione tramite prove sperimentali dell'errore sistematico di sbandamento dovuto a tali problemi fisico-meccanici. Si è fatto percorrere al robot per diverse volte un percorso rettilineo di $5m$, e si è misurata l'effettiva distanza percorsa secondo i due assi coordinati, da qui si è estrapolata una costante correttiva di moto e la si è applicata al modello di moto. I risultati, che su un moto rettilineo erano più che soddisfacenti, in un percorso più complesso costituito da curve e rotazioni risultava ampiamente mal condizionato. I disequilibri meccanici presenti nel robot provocano effetti di moto imprevisti quando il veicolo mobile effettua curve o correzioni di rotta, per cui tale modellizzazione non è risultata efficace ed utile. Per lo spontaneo moto di cambiamento di orientamento, invece, non è stato in alcun modo possibile effettuare una stima di tale movimento in quanto questo avviene su base non regolare, e può dipendere da cause molto diverse e non osservabili.

Si è quindi proceduto con il metodo dell'osservazione tramite sensori dell'errore sistematico. Il robot Nomad XR4000 è equipaggiato con una bussola digitale[10], che avrebbe potuto risolvere almeno il problema dell'orientamento in modo effettivo. Si è scritta una funzione di osservazione dell'orientamento tramite la bussola attraverso l'uso un filtro di Kalman, implementato in `observe_heading()`. Questa soluzione non è risultata nuovamente efficace, per via dell'estrema condizione di rumorosità elettromagnetica presente nell'area[11] in cui sono stati svolti i test sperimentali. Anche dopo una accuratissima procedura di calibrazione, la bussola magnetica restituiva risultati completamente inaffidabili e quindi non utili ai fini del FastSLAM. Oltretutto, l'osservazione tramite la bussola, poteva restituire la misura corretta dell'orientamento del robot, ma non certo, per via dei problemi meccanici di movimentazione del robot spiegato in precedenza, la sua direzione di moto.

[10] Si veda la sezione 6.1.3.

[11] Primo e quarto piano dell'edificio di ingegneria dell'informazione - DISP

L'unica modalità possibile per tener conto di questi errori sistematici è stata quella di aumentare l'incertezza angolare e posizionale tramite la matrice Q del modello di moto, e l'utilizzo di un numero sufficientemente alto di particelle.

La soluzione architettata per ovviare al problema della non fedeltà tra la velocità imposta al controller del motore e la reale velocità del robot, è stata quella della ricerca di una costante di correzione. Questa è stata calcolata con accurate prove sperimentali: si è fatto percorrere al robot un moto rettilineo a velocità costante, portandolo a velocità di regime, per la fissata distanza di $5\,m$, e si è misurato, grazie ad un timer implementato nel robot con precisione $10^{-3}\,s$, il tempo trascorso. Da questi due parametri è stato banalmente trovata la velocità effettiva calcolando $v_{eff} = \frac{x_{eff}}{t_{eff}}$, da cui rapportandola con la velocità nominale assegnata al controller del motore si è trovata la costante di correzione di velocità traslazionale. Procedimenti analoghi sono stati effettuati per la ricerca della costante di correzione di velocità angolare. Queste costanti sono nell'ordine di ~ 0.4, le velocità nominali sono cioè, circa la metà delle velocità effettive, e sono un indice di quanto poco sia accurata l'architettura di controllo del robot.

6.3.1 Tuning dei parametri del FastSLAM

Il FastSLAM ha molteplici parametri da settare per avere un corretto funzionamento. Tutti i parametri fondamentali sono inclusi nel file `configure.h`. Verranno qui descritti quelli che hanno maggiore importanza. Variazioni effettuate a questi parametri agiscono in modo fondamentale nel processo di SLAM, ed è necessario prestare particolare attenzione ad ognuno di essi.

`NPARTICLES`

Questa costante definisce il numero di particelle presenti nel filtro particellare. È un parametro molto importante che influenza la velocità e la precisione dell'intero processo di SLAM. Avere un numero di particelle alto permette una migliore approssimazione sulla posa del robot, perché ne quantifica l'incertezza con un

livello alto di raffinatezza, quantizzando con fine granularità le varie possibili pose espresse dalle singole particelle e quindi di migliorare i risultati dello SLAM. Avere un numero alto di particelle, non rappresenta però sempre un vantaggio, all'aumentare del loro numero la complessità computazionale dell'intero algoritmo aumenta linearmente, nonché anche l'occupazione di memoria. Tuttavia esiste un limite teorico in cui aumentare il numero di particelle non porta ad alcun vantaggio in termini di precisione del risultato. Utilizzare, invece, un numero di particelle particolarmente basso nel filtro particellare permette un significativo vantaggio in termini di prestazioni computazionali, ma anche una significativa perdita di precisione nel processo di SLAM.

`GATE_REJECT`

Questo parametro viene utilizzato in `data_association_unknown()` e definisce la massima distanza di Mahalanobis per cui ad una *feature* viene associata una determinata osservazione. Se tale distanza risulta essere inferiore a questa soglia allora l'osservazione è associata a tale *feature*. Scegliere i parametri per l'associazione di dati è una scelta delicata. Più il valore di questa soglia viene impostata a un valore alto, più facilmente si tenderanno a confondere le osservazioni di *feature* vicine e quindi a compiere errori sulla mappa ma anche alla correzione della posa del robot effettuata dal FastSLAM. Più questa soglia ha un valore piccolo, più facilmente è possibile distinguere le osservazioni, ma, se impostata a un valore troppo basso risulterà molto difficile associare le osservazioni alle corrette *feature* a meno di avere una posa del robot molto poco affetta da errore durante tutto l'esperimento. Se ciò non è ottenibile allora il FastSLAM potrebbe divergerere poichè non viene attuata la correzione della posa, in quanto non vengono incorporate nessuna, o troppo poche, osservazioni.

`GATE_AUGMENT`

Questa costante rappresenta, in `data_association_unknown()`, la minima distanza di Mahalanobis che un osservazione deve ottenere per creare una nuova

feature. Se si imposta un valore basso si rischia di confondere una osservazione di una *feature* con un nuovo *landmark*, oppure di creare troppe *feature* non necessarie, date da osservazioni spurie, con un conseguente aumento della memoria occupata. Assegnare un valore troppo alto a questo parametro invece, provoca che poche *feature* siano incorporate nel filtro, di conseguenza poche volte sarà possibile riosservare tali *feature* e quindi saranno possibili poche correzioni sulla posa e sulla mappa.

`N_MIN`

Il *resampling* è un processo critico che influisce sulla diversità, quindi sulla capacità di convergenza, dell'intero filtro particellare. Impostare questo parametro ad un valore percentuale alto significa obbligare il FastSLAM ad effettuare un *resampling* frequente, quindi a impoverire la diversità del filtro, ma anche costringere il FastSLAM a prendere decisioni sull'andamento delle pose in modo precoce. Impostare questo parametro a valori molto bassi significa aumentare la diversità ma anche allungare la vita di particelle con basso peso. Se `N_MIN` risulta essere troppo basso può portare alla divergenza del FastSLAM, perché lo SLAM verrà condizionato su pose molto lontane dalla realtà. Un lavoro sulla sensibilità di questo parametro e sulla sua importanza nel FastSLAM è presentato da S.Thrun [8].

`Z_OVERBOUND_DISTANCE`

Questo parametro individua la massima distanza delle letture effettuate del sensore laser. È stato verificato sperimentalmente che l'apparecchiatura laser SICK-S550 imposta a un valore massimo tutte le letture che eccedono il suo *range* di misura, oltretutto molte misure vicine a tale distanza sono spurie e affette da forte rumore. Con la soglia `Z_OVERBOUND_DISTANCE` si imposta il nuovo massimo *range* di misura e si ignorano tutte le misure del sensore che la eccedono, in modo da filtrare questo genere di problema. Questa soglia influenza anche l'algoritmo di controllo per scegliere la direzione di camminamento.

`SIGMA_V` e `SIGMA_OMEGA`

Le matrici Q ed R definiscono rispettivamente l'errore del modello di moto e di misura. Vengono definite come:

$$Q = \begin{bmatrix} \sigma_V^2 & 0 \\ 0 & \sigma_\Omega^2 \end{bmatrix} \tag{6.2}$$

$$R = \begin{bmatrix} \sigma_R^2 & 0 \\ 0 & \sigma_B^2 \end{bmatrix} \tag{6.3}$$

I parametri σ_V e σ_Ω sono impostati rispettivamente tramite `SIGMA_V`, espressa in m/s, e `SIGMA_OMEGA`, espressa in rad/s, e rappresentano la varianza della velocità traslazionale e della velocità angolare. Più si aumentano tali valori più l'incertezza della posa sarà alta, quindi il filtro particellare presenterà una maggiore sparsità spaziale.

I valori σ_R e σ_B sono impostati rispettivamente tramite `SIGMA_R`, espressa in m, e `SIGMA_B`, espressa in rad, e rappresentano l'errore dell'apparecchiatura di misurazione laser sulla distanza e la sua l'incertezza angolare. Questi valori sono impostati ad un valore basso poichè il laser ha una precisione molto alta e permette una affidabilità sulle misure effettuate.

6.4 Stima della posizione tramite *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 rappresentava l'effettiva velocità di moto. 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*.

6.4.1 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.

6.4.2 *ICP Scan-Matching*

L'*ICP* (acronimo per *Iterative Closest Point*) [25], è stato utilizzato per effettuare una stima delle velocità tra due istanti di tempo, utilizzando semplicemente le letture del sensore laser.

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 è quindi calcolata le velocità medie intercorse nell'intervallo di tempo dt, da inserire nella funzione di predizione, così sollevata dal compito della rilevazione 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.

6.5 Risultati sperimentali

I risultati sperimentali, applicazione di tutta la teoria e frutto di tutto lo sviluppo presentato in questo progetto di tesi, verranno qui presentati e descritti.

L'ambiente di prova è costituito dal quarto piano dell'edificio DISP[12]. Il robot appena avviato considera il proprio punto di partenza come l'origine del sistema di riferimento assoluto. Il Nomad XR4000 naviga autonomamente in questo ambiente[13] ed esegue un percorso che permette di eseguire poco più di un giro.

Per esigenze di studio si è innanzitutto costruita una mappa utilizzando il sensore laser, basandosi sulla traiettoria costituita dalla lettura dei sensori odometrici (il cosidetto *dead-reckoning*) senza utilizzare alcuna correzione e senza applicare il metodo del FastSLAM. I risultati, riportati in figura 6.8, riportano uno scarsissimo livello di precisione, e una fortissima influenza dell'errore sistematico sulla qualità dell'esperimento. La mappa non si chiude, e non ha un comportamento fedele al reale percorso che il robot ha effettuato, risulta quindi completamente inutile. È per questo evidente l'importanza del metodo del FastSLAM per ottenere un *mapping* e una stima del percorso fedele alla realtà.

In tutte le figure qui riportate le tracce rosse corrispondono al percorso effettuato dalle singole particelle, le tracce nere rappresentano il percorso effettuato dalla particella con peso maggiore (cioè la particella che ha ottenuto maggiore fiducia nel filtro, e che quindi rappresenta la migliore stima che il FastSLAM riesce a restituire sulla posizione del robot).

Il *tuning* dei parametri che agiscono direttamente sulla qualità del metodo del FastSLAM è un'operazione complessa e molto laboriosa; verrano di seguito presentati i risultati frutto di modifiche sui parametri: `SIGMA_V` e `SIGMA_OMEGA` (rispettivamente le varianze della velocità traslazionale e rotazionale, presenti sulla diagonale della matrice Q), `SIGMA_R` e `SIGMA_B` (varianze di distanza e orientamento del sensore laser, presenti sulla diagonale della matrice R), livello di

[12] Acronimo di Dipartimento Informatica Sistemi e Produzione

[13] Esplora l'ambiente secondo i metodi descritti nel capitolo 7

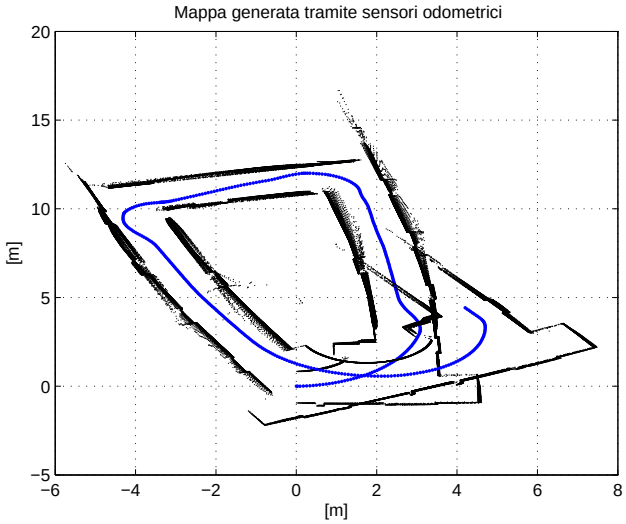


Figura 6.8: *Risultato della prova sperimentale effettuato utilizzando come stima di posizione la lettura dei sensori odometrici (in blu).*

resampling, e i parametri `GATE_REJECT` e `GATE_AUGMENT` relativi all'asssociazione di dati.

Analogamnente alla simulazione in Matlab è stato modificato un solo parametro alla volta, per studiarne l'effetto sulla prova sperimentale. Il robot è stato impostato con i seguenti parametri:

- `SIGMA_V`$= 5.0$
- `SIGMA_OMEGA`$= 5.0$
- $\sigma_R = 0.1$
- $\sigma_B = 0.5$
- Numero di particelle (`NPARTICLES`): 30
- Livello di resampling: 95%
- `GATE_REJECT`$= 100$

- GATE_AUGMENT= 800
- NIT= 20 numero di iterazioni per ogni scan matching.
- GATE1= 20 prima soglia di associazione per ogni punto della scansione dello scan matching.
- GATE2= 20 seconda soglia di associazione per ogni punto della scansione dello scan matching.

Aumentando il valore delle varianze SIGMA_OMEGA e SIGMA_V, si aumenta pure il determinante della matrice di incertezza del *motion model* Q. Ne risulta, si veda la figura 6.9, un filtro particellare spazialmente distribuito; proprio a causa della alta incertezza indicata, il filtro distribuisce le particelle in un area più ampia in modo da tenere conto dell'alto grado di errore. Nonostante questo la traiettoria della particella di maggior peso risulta ben centrata rispetto alla traiettoria reale, e il *loop* si chiude precisamente, per cui la stima di posizione del robot e le *features* convergono alle loro posizioni reali.

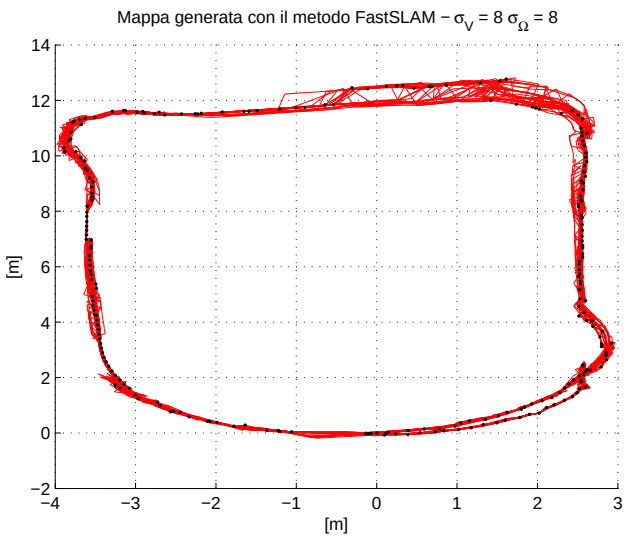


Figura 6.9: *Risultato della prova sperimentale effettutato con SIGMA_OMEGA= 8.0 e SIGMA_V= 8.0.*

Analogamente al caso in Matlab, si è studiato il comportamento della prova sperimentale al cambiamento dei due parametri relativi alla funzione di associazione di dati sconosciuta: `GATE_REJECT` e `GATE_AUGMENT`. Si ricorda che `GATE_REJECT` rappresenta la massima distanza di Mahalanobis entro cui un osservazione viene associata a un *landmark* presente nel filtro; `GATE_AUGMENT` rappresenta il minimo valore di distanza di Mahalanobis per cui deve essere incorporato nel filtro una osservazione, creando un nuovo *landmark*.

Impostando `GATE_REJECT`$= 20$, diminuendone cioè il valore, si verifica che il *loop* non viene chiuso, si veda la figura 6.10. Diminuire il `GATE_REJECT` vuol dire associare le osservazioni ai *landmark* incorporati nel filtro solo se la distanza di Mahalanobis è inferiore a tale soglia diminuita. Effettuare un operazione di decremento di tale soglia vuol dire associare le osservazioni solo se sono molto nettamente, con poca ambiguità, associabili ai *landmark* già incorporati, ma significa, di conseguenza, che si concede molta fiducia alle misure dell'ambiente. Se un ambiente è molto rumoroso, come quello della prova sperimentale, la variazione del parametro `GATE_REJECT` risulta uno svantaggio: poche osservazioni risultano essere più basse di `GATE_REJECT`, quindi pochi *landmark* vengono riosservati; questo spiega perchè la stima effettuata dal FastSLAM durante il percorso è poco affidabile. Quando il robot torna a esplorare di nuovo la parte della mappa per prima visitata, ove sono stati incorporati i primi *landmark* (poco affetti da errore[14]), la stima della posa del robot è lontana dal reale percorso per cui, a causa di questo scostamento, l'associazione di dati non riesce a associare l'osservazione ai *landmark* già incorporati.

Aumentando il valore di `GATE_AUGMENT` a 200, si mostra come il *loop* viene correttamente chiuso, ma il percorso stimato risulta però essere affetto da errore. Questo è dovuto al fatto che aumentando `GATE_AUGMENT` si definisce una soglia di distanza di Mahalanobis più alta per incorporare un nuovo *landmark*, si diminuisce

[14]La posizione dei primi *landmark* incorporati nel filtro è poco affetta da errore, poichè avendo il robot presumibilmente effettuato un percorso breve per incorporarli ad inizio prova sperimentale, il filtro particellare in quell'istante aveva poca incertezza di posizione; quindi tali *landmark* rappresentano un ottima stima della loro posizione reale.

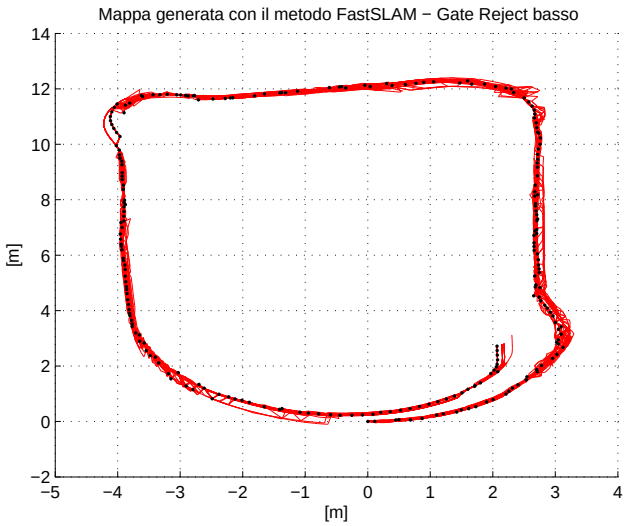


Figura 6.10: *Risultato della prova sperimentale effettutato con* `GATE_REJECT`$=20$.

quindi il numero di *landmark* presenti nel filtro particellare, decrementando la quantità si diminuisce di conseguenza la possibilità di correggere la stima della posizione. Il risultato della prova sperimentale è visibile in figura 6.11.

In figura 6.12 si è imposto il livello di *resampling* a 10%. Il FastSLAM è costretto ad effettuare il *resampling* meno spesso, effettuando l'eliminazione delle particelle con poco peso meno frequentemente. Questo causa, seppure in poca misura, un aumento dell'errore di stima durante il percorso, ma il FastSLAM riesce comunque a chiudere correttamente il ciclo. Appena il robot ha effettuato sufficienti riosservazioni delle prime *features* incorporate, la distribuzione particellare collassa verso la corretta stima, e di conseguenza anche le *features* ad esso collegate covergono verso la reale posizione.

Per completezza si è aumentato il numero di particelle presenti nel filtro particellare a 100. Il risultato della simulazione, mostrato in figura 6.13, è molto buono, il *loop* viene correttamente chiuso e la stima della posa del robot nel tempo è molto preciso. Dato il grande numero di particelle, il filtro riesce ad approssimare bene l'incertezza sulla posa proponendo molte pose che bene ap-

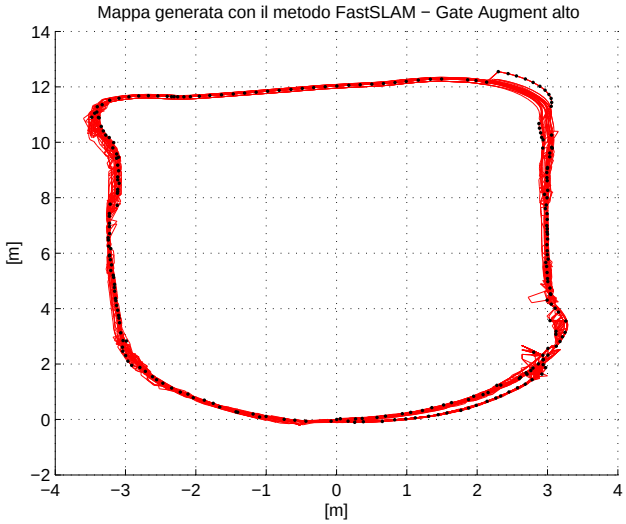


Figura 6.11: *Risultato della prova sperimentale effettutato con* `GATE_AUGMENT`$=200$.

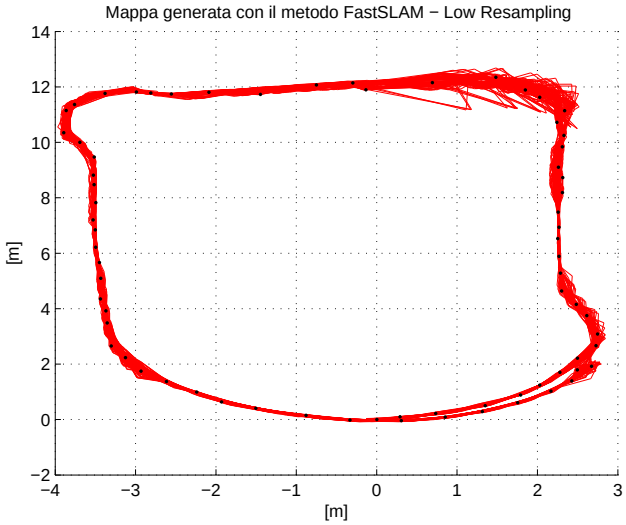


Figura 6.12: *Risultato della prova sperimentale effettutato con resample* 10%.

prossimano quella reale e conseguentemente la reale localizzazione dei *landmark*, restituendo un risultato molto preciso.

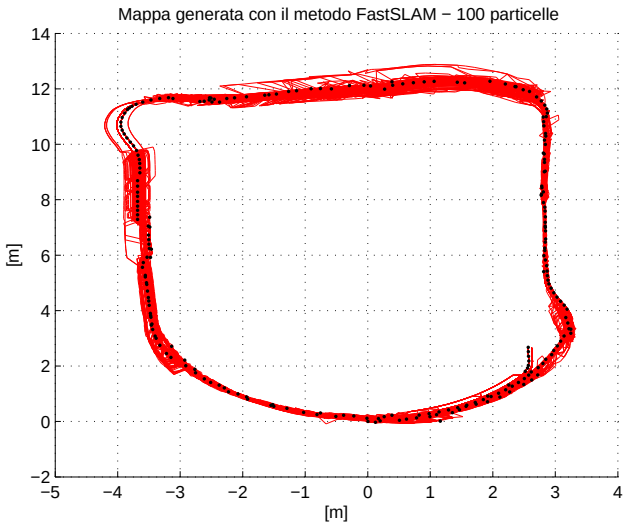


Figura 6.13: *Risultato della prova sperimentale effettutato con* 100 *particelle.*

Per diminuire i tempi di calcolo e gli oneri computazionali si è posto il numero di particelle pari a 30 e, in accordo alla teoria formulata da M. Montemerlo [13], i risultati sono stati molto simili. Il *loop* viene chiuso correttamente e il percorso stimato è molto fedele a quello reale. In figura 6.14 è mostrata anche la mappa costruita durante l'esplorazione.

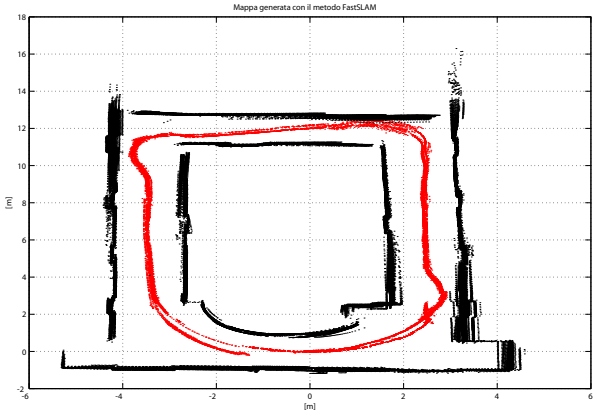


Figura 6.14: *Risultato finale della prova sperimentale.*

È importante sottolineare che, grazie all'utilizzo combinato del FastSLAM e dello Scan-Matching, il robot effettua uno SLAM di qualità con l'utilizzo del solo sensore laser, non effettuando alcuna misura odometrica.

Il robot per completare il percorso ha impiegato 7 minuti e 11 secondi, percorrendo un totale di circa $36m$. La velocità media risultante è circa $83mm/s$, tenendo in considerazione il fatto che il controllo del robot satura per sicurezza la velocità massima a $100mm/s$ e il robot procede secondo un moto uniformemente accelerato, si evince che il robot ha una sostenuta velocità di esplorazione, molto vicina alla velocità massima ottenibile.

Capitolo 7

Autonomia del robot e controllo

In questo capitolo verranno descritte le metodiche teoriche e pratiche utilizzate per rendere il robot Nomad XR4000 autonomo. Per rendere indipendente il robot è stato sviluppato un metodo di *obstacle avoidance* e una tecnica di esplorazione dell'ambiente; insieme forniscono le informazioni per effettuare il controllo del robot. Per completezza, in questo capitolo verrà trattata anche la costruzione della mappa.

7.1 *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 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].

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* implementato è il seguente: considerando le letture dei sensori 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 la velocità di riferimento da impartire ai motori, sulla base di un controllo proporzionale, in modo tale da poter evitare in maniera efficace l'ostacolo. Per la descrizione dettagliata dell'algoritmo di esplorazione si rimanda a [19].

7.2 Algoritmo di esplorazione

Dopo aver verificato efficienza della tecnica del FastSLAM si è sviluppato un metodo di navigazione in ambiente sconosciuto. Data la scarsa capacità com-

[1]Inizialmente si è pensato di utilizzare i sonar presenti, ma l'ipotesi è stato presto accantonata, 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.

putazionale del robot, occupato principalmente nell'esecuzione dell'algoritmo di FastSLAM, è stato necessario scrivere una quanto mai efficiente *routine* di esplorazione.

L'ambiente che esplora il robot, è totalmente sconosciuto; in fase di progettazione e sviluppo dell'algoritmo di esplorazione non sono stati impostati vincoli di alcun tipo, come ad esempio modelli di ambienti del sito di esplorazione. 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 $2500m^2$ in poco spazio di memoria, con una ridotta necessità computazionale.

L'algoritmo di esplorazione decide la direzione di moto per la naviazione 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_r e Ω_r da passare al controller del motore attraverso l'utilizzo di un controllo proporzionale.

Nel seguito viene descritto nel dettaglio il funzionamento e la teoria di questa tecnica.

7.2.1 Principi di funzionamento

La scelta del sensore

Per un processo di esplorazione la scelta del sensore da utilizzare rappresenta il punto di partenza. L'*hardware* sensoriale a medio-lungo raggio presente nel robot restringe le scelta al sensore sonar e al sensore laser.

Il vantaggio del sistema sonar, capace di misurare fino a una distanza massima di circa $5m$, è che i sensori sono distribuiti su due corone che circondano il robot; avere un informazione dell'ambiente a 360^o senza dubbio favorisce la

scelta della direzione di moto. Tale sistema sensoriale ha la particolarità di soffrire di frequenti misure spurie e di restituire misurazioni di un ambiente molto rumoroso, spesso deformato a causa di particolarità geometriche degli ostacoli circostanti. A meno di effettuare successive misurazioni e costruire uno stadio di filtraggio efficiente del rumore, i sensori sonar risultano svantaggiosi e molto imprecisi. D'altronde effettuare più misurazioni significherebbe diminuire la velocità di andatura del robot e costruire uno stadio di *filtering* significherebbe aumentare il carico computazionale complessivo del robot.

Per esclusione si è selezionato il sensore laser come sensore da utilizzare per la tecnica di esplorazione. Ha il vantaggio di essere molto preciso e sufficientemente veloce, ma restituisce soltanto le misure del semipiano frontale del robot. L'algoritmo di esplorazione deve tenere conto di questa limitazione per decidere la più favorevole direzione di moto per l'esplorazione.

Grid-based Map

Durante un processo di esplorazione è necessario sapere in ogni momento la propria posizione e sapere che direzione prendere nell'ambiente sconosciuto. Conoscere la posizione del veicolo è una operazione semplice: viene considerata come posizione attuale la posa della particella con maggior peso nel filtro particellare del FastSLAM. Lo spazio esplorabile, fissato all'inizio del test, viene discretizzato in macrocelle quadrate; queste celle hanno un area di dimensioni $1m^2$. Il punto di partenza del robot è il centro della mappa.

Ogni cella è definita dalla sua posizione nella mappa e dal suo stato $cell_{x,y} = \langle x_{map}, y_{map}, s \rangle$, come descritto in figura 7.1. Lo stato cella può assumere soltanto tre valori discreti:

1. **Non visitato:** La cella non è stata ancora visitata.
2. **Visitato:** La cella è stata già visitata dal robot.
3. **Candidato:** La cella è stata segnalata come una cella da visitare in futuro.

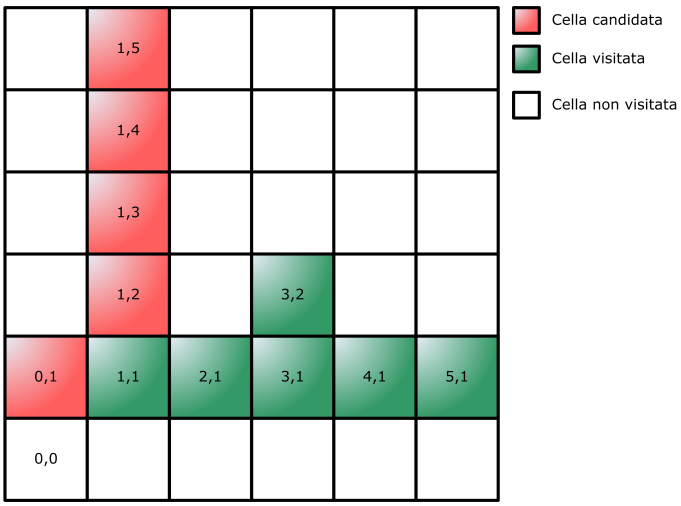


Figura 7.1: *Disposizione delle celle in una mappa grid-based.*

Lo stato di una cella viene assegnato a `Visitato` quando il robot la ha occupata, `Candidato` se essa è stata osservata nelle vicinanze del robot dal sensore. Le celle che hanno uno stato diverso da *Non visitato* vengono organizzate in grafi. Ogni cella mantiene l'informazione sulla propria cella padre, cioè sulla sua radice.

Lo svantaggio di un metodo *grid-based* di divisione dello spazio è costituito proprio dal livello di quantizzazione dello spazio e dalla dimensione massima fissata della mappa. Questi problemi sono stati ovviati scegliendo una dimensione delle celle consona alla dimensione del robot, e una dimensione della mappa molto grande del massimo ambiente esplorabile, mantenendo comunque un'occupazione di memoria accettabile.

7.2.2 Criteri di selezione della traiettoria

Il primo passo per la scelta della traiettoria è l'individuazione delle possibili direzioni di moto. Si è perciò sviluppata una funzione di analisi dei dati restituiti dal sensore laser. La funzione `heading_finder()`, individua la direzione in cui sono presenti sufficienti letture maggiori della soglia `Z_OVERBOUND_DISTANCE`[2].

[2]Si veda sezione 6.3.1

Trovare sequenze di misurazioni maggiori di questa soglia vuol dire verosimilmente trovare un possibile spazio percorribile, come un corridoio ad esempio. Gli orientamenti di tali direzioni vengono calcolati mediando gli orientamenti delle sequenze di misure consecutive, figura 7.2, e vengono calcolati rispetto al sistema di riferimento fisso. Questo semplice metodo funziona molto efficacemente e con poche risorse computazionali.

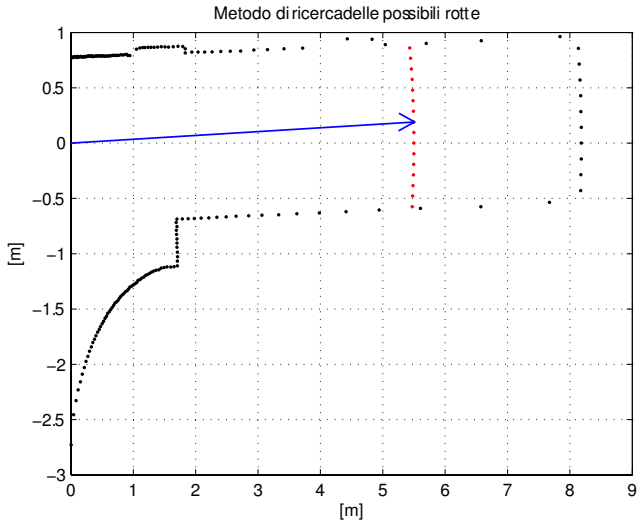


Figura 7.2: *Metodo di ricerca delle possibili rotte. In blu è visualizzata la possibile rotta, in rosso l'orizzonte limitato dalla soglia* `Z_OVERBOUND_DISTANCE`.

Dopo aver calcolato tutti i possibili orientamenti con la funzione precedente, vengono quindi processati dalla funzione `decision()` che decide quale direzione selezionare. Questa decisione viene presa basandosi su determinati criteri descritti nel dettaglio nel diagramma di flusso in figura 7.4. Il criterio di decisione di fondo è quello di cercare di procedere verso la direzione che porta il robot più lontano dalla propria posizione di partenza, in modo potenzialmente di esplorare tutto l'ambiente. Nel caso che ci si trovasse in un vicolo cieco il robot sceglierà di ruotare finché non viene trovato un nuovo possibile orientamento. Un altro caso interessante da menzionare è se il veicolo si trova in una condizione in cui tutte le direzioni puntano a celle già visitate. Quando avviene questa possibilità si

effettua una ricerca sul grafo di tutte le celle segnate come candidate; per ogni direzione si trova la cella candidata più vicina, calcolando la distanza sul grafo, alla cella in cui cade tale osservazione (figura 7.3). Quindi, tra tutte le distanze

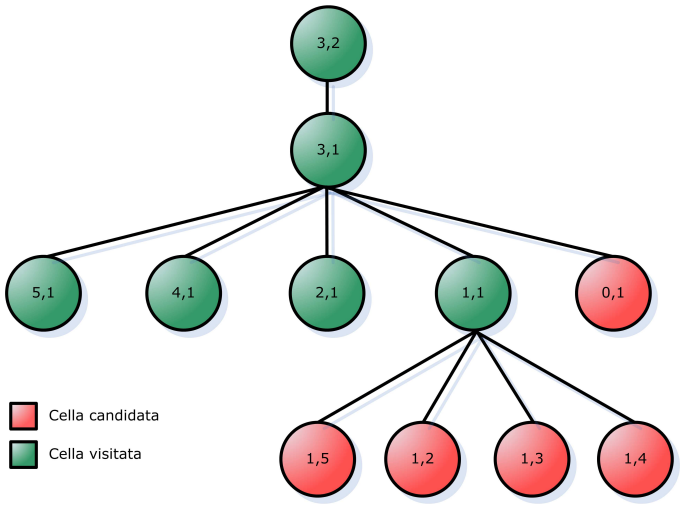


Figura 7.3: *Grafo delle celle.*

si sceglie la minore e si procede per quella direzione. Il calcolo della distanza sul grafo avviene contando il numero di passi necessari per arrivare da una cella all'altra; questo è possibile grazie all'informazione contenuta in ogni nodo del grafo del suo nodo padre.

7.2.3 Il controllo di navigazione

Per restituire i valori di riferimento al controller del motore sono stati utilizzati due controllori proporzionali, figura 7.5. Questi due controllori agiscono rispettivamente sui riferimenti V_r e su Ω_r. Per Ω_r viene calcolato l'orientamento relativo selezionato rispetto al robot, e tramite una proporzione con l'orientamento attuale viene indicato il valore di riferimento per la velocità angolare. Per V_r vale un ragionamento analogo: valutando il modulo dell'orientamento della direzione selezionata si calcola con una legge proporzionale il valore del riferimento V_r, in modo che più tale angolo è grande più è piccola la velocità traslazionale. In

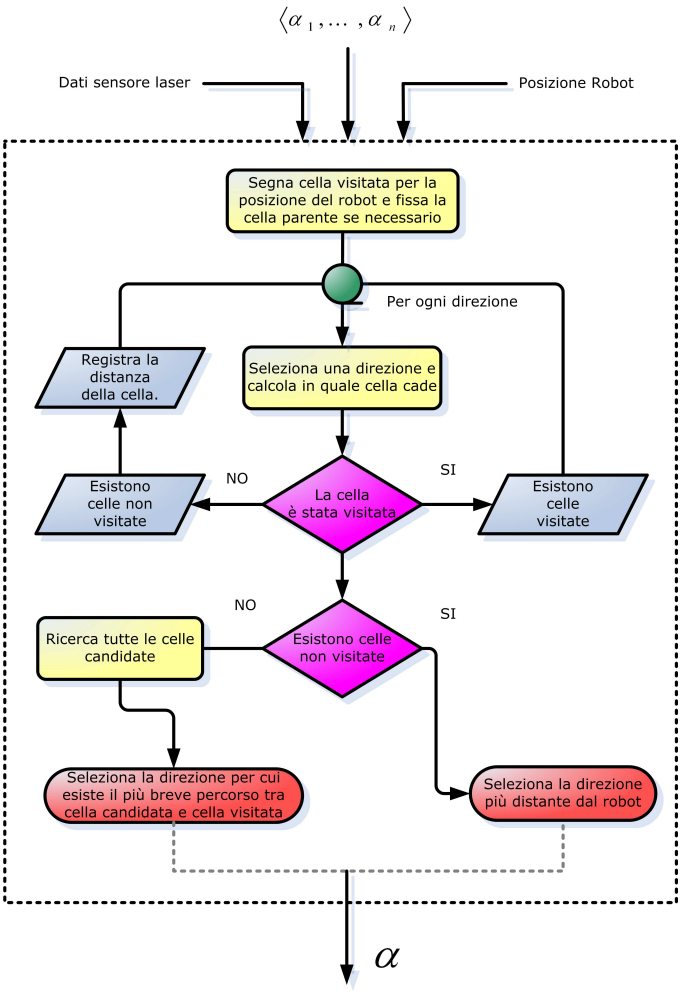


Figura 7.4: *Diagramma di flusso per la funzione di decisione della rotta.*

questo modo si aiuta il robot mobile a compiere le curve in maniera più morbida, senza avvicinarsi troppo alle pareti.

Durante la fase di uscita di un ostacolo da parte dell'*obstacle avoidance*, viene amplificato il controllo sulla velocità angolare moltiplicandola per una costante K_{obs}, in questo modo, per esempio, si riesce a recuperare velocemente il centro di un corridoio dopo una curva, evitando oscillazioni da una parete all'altra.

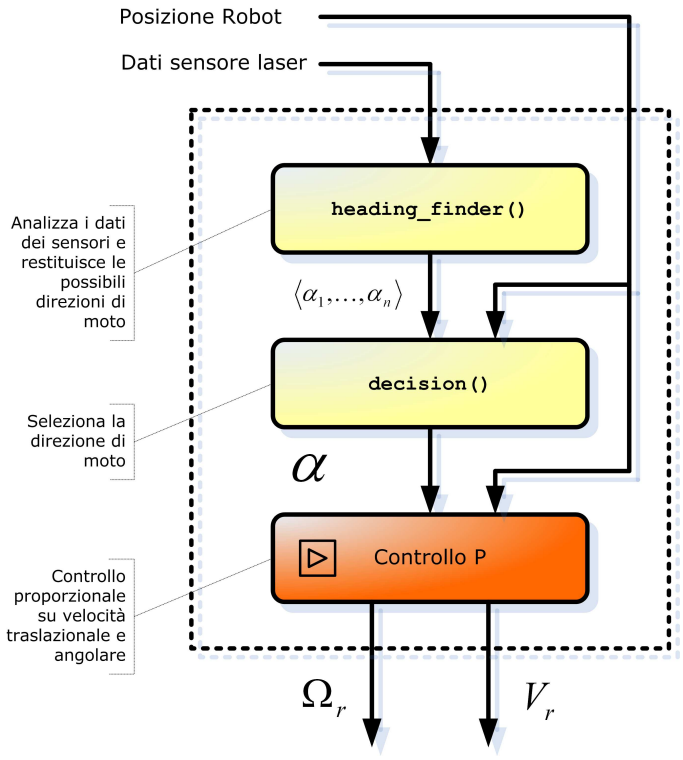


Figura 7.5: *Diagramma di flusso dell'algoritmo di controllo per l'algoritmo di esplorazione.*

7.2.4 L'implementazione dell'algoritmo

L'algoritmo di navigazione ed esplorazione è sviluppato dentro la funzione `control()` che invoca `heading_finder()`, che si occupa di trovare le possibili direzioni di

moto attraverso l'analisi dei dati del sensore laser, e `decision()`, che si occupa della gestione della mappa *grid-based* e di decidere la direzione di moto del robot. Infine vengono qui calcolati tramite propozioni i riferimenti V_r e Ω_r.

7.3 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`). Questi passano uno stadio di saturazione per evitare valori di velocità troppo alti o inferiori al minimo progettuale. Lo schema complessivo di controllo è riportato in figura 7.6.

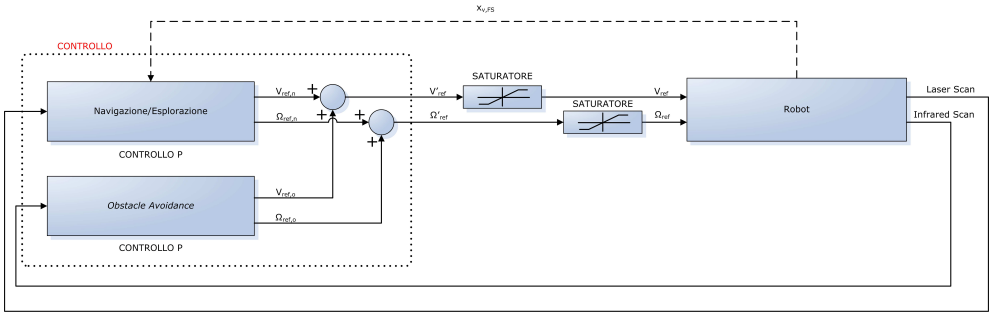


Figura 7.6: *Schema di controllo complessivo del robot.*

7.4 Costruzione della mappa

Per completare questo lavoro di tesi è stato sviluppato un metodo di costruzione della mappa in un formato facilmente comprensibile. La mappa per il FastSLAM è costituita dalla posizioni delle *feature*, rappresentate da punti in uno spazio bidimensionale. Questa mappa non rappresenta però una interpretazione intuibile dell'ambiente esplorato dal robot. È difficile comprendere la forma dell'ambiente

soltanto dalla posizione delle *feature*. Per questo motivo è stata creata la funzione `create_map()`. In questa vengono, per ogni particella, elaborate le letture del sensore in modo da trasformarle in coordinate assolute, condizionandole sulla posa della particella corrente. In aggiunta, ogni lettura laser viene legata alla posa della *feature* più vicina tramite una coppia di misure distanza-angolo. Questo è stato sviluppato per aggiustare la mappa dinamicamente al variare dell'aggiustamento delle *feature*. In questo modo l'aggiustamento della posa del robot e della locazione delle *feature* provocano un corretto aggiustamento della mappa generata. Questa correzione è sviluppata nella funzione `adjust_map()`, che recuperando i dati dei vettori distanze misure-*feature*, aggiusta la mappa tramite le nuove coordinate dei *landmark* aggiornate dal FastSLAM. A fine simulazione viene salvata la mappa della particella con peso più alto.

Capitolo 8

Conclusioni e sviluppi futuri

I risultati sperimentali del progetto confermano la qualità del metodo del FastSLAM combinato con lo *SCAN-Matching*, sia nell'implementazione in Matlab sia nella reale applicazione sperimentale sul robot mobile. Come descritto nei risultati del capitolo 6, il robot è in grado di costruire una mappa affidabile, e di ottenere una stima precisa della sua posizione.

Per riuscire ad ottenere tali risultati sono state effettuate decine di prove sperimentali complete e lunghe e laboriose verifiche. Il robot risulta compiere autonomamente un giro completo del quarto piano dell'edificio DISP di ingegneria, circa $36m$ in poco più di sette minuti, navigare autonomamente decidendo le giuste rotte ed evitare le pareti. Il robot effettua il percorso con un moto continuo, senza fermarsi.

Questi risultati, di qualità, confermano la teoria descritta in questo lavoro di tesi e rappresentano un ottimo metodo di esplorazione di un ambiente sconosciuto in maniera autonoma, effettuandolo cioè senza ausilio di un operatore. Gli obiettivi della tesi sono stati raggiunti con successo.

8.1 Sviluppi futuri

Il successivo passo di sviluppo di questo lavoro di tesi risiede nell'implementazione su robot multipli della tecnica combinata FastSLAM/*SCAN-Matching* . Implementare cioè le tecniche qui sviluppate in una squadra collaborativa di robot autonomi, dotati anche di sensori eterogenei, in grado di dividere il lavoro e di esplorare e mappare ambienti sconosciuti molto vasti. Questo obiettivo è stato preso in considerazione durante la fase progettuale del lavoro e si è sviluppato il cuore del *software* del robot, costituito dal binomio FastSLAM/*Scan-Matching*, in modo che fosse indipendente dal tipo di sensore montato sul robot mobile, rendendo cioè facilmente applicabili modifiche al modello di osservazione e quindi rendendolo riutilizzabile su un altro tipo di robot mobile. Tutti i parametri dipendenti dall'*hardware* del robot sono facilmente accessibili e adattabili ad un altro tipo di robot.

Un altro sviluppo potrebbe essere quello dell'applicazione di successive tecniche di *path planning* e/o *motion planning* alla fase di SLAM. Cercare cioè di trovare una legge di moto o una traiettoria applicando un criterio di ottimo per raggiungere un punto da un altro della mappa ormai conosciuta.

Bibliografia

[1] Ercan U. Acar, Howie Choset, Yangang Zhang, and Mark Schervish. Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods. In *The International Journal of Robotics Research*, volume 22, pages 441–466. July-August 2003.

[2] Ann Arbor. Umbmark: A method for measuring, comparing and correcting dead-reckoning errors in mobile robots. Technical report, Univ. of Michigan, December 1994.

[3] 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.

[4] 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.

[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, M. Booth, and F. Rossi. *GNU Scientific Library - Reference Manual*. Free Software Foundation, 1.6 edition, December 2004.

[8] D. Hahnel, 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.

[9] Julier, Simon, and Jeffrey Uhlman. A general method of approximating nonlinear transformations of probability distributions. Technical report, Robotics Research Group, Department of Engineering Science, University of Oxford, November 1995.

[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] P. Moutarlier and R. Chatila. Stochastic multisensory data fusion for mobile robot location and environment modeling. Technical report, 5th International Symposium on Robotics Research, Tokyo, 1989.

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

[16] J. C. Nash. The choleski decomposition. In Bristol, editor, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*, chapter 7, pages 84–93. 1990.

[17] J. Niento, J. Guivant, E. Nebot, and S. Thrun. Real time data association in fastslam. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, 2003.

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

[19] F. Romanelli. Filtri particellari per la localizzazione e la navigazione di robot mobili in ambienti non noti privi di landmark. Master's thesis, Università degli studi di Roma "Tor Vergata", 2004/2005.

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

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

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

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

[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] Z. Y. Zhang. Iterative point matching for registration of free-form curves and surfaces. In *Int. J. of Computer Vision*, pages 119–15. October.