

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

1

ESTENSIONE E VALUTAZIONE
DI UN LOCALIZZATORE BAYESIANO
PER ROBOT MOBILI

Relatore:

Chiar.mo Prof. STEFANO CASELLI

Correlatori:

Ing. DARIO LODI RIZZINI

Ing. FRANCESCO MONICA

Tesi di laurea di:

BRUNO FERRARINI

ANNO ACCADEMICO 2005-2006

Alla mia famiglia

Vorrei ringraziare il Prof. Stefano Caselli per essere stato un prezioso punto di riferimento durante tutto il periodo di questa mia seconda esperienza come studente e la cui competenza scientifica è stata un indispensabile sostegno durante il periodo della tesi.

Fondamentale è stata la guida di Dario nella disciplina della localizzazione. Gli sono estremamente grato per i suoi consigli, per l'infinita pazienza e per tutto il tempo che mi ha dedicato dal primo all'ultimo giorno dell'attività di tesi.

Il Nomad e io dobbiamo ringraziare Francesco. Il nostro robot è ormai vecchio e stanco, e soffre di parecchi acciacchi che il dottor Monica ha sempre egregiamente curato.

Naturalmente un grazie anche ai ragazzi che hanno condiviso con me quello che considero la parte migliore del periodo degli studi.

“Non ho paura dai computer, ma della loro eventuale mancanza.”

Isaac Asimov

Indice

| | | |
|----------|--|-----------|
| 1 | Introduzione | 1 |
| 2 | Cenni sui localizzatori bayesiani | 4 |
| 2.1 | Il problema della localizzazione | 4 |
| 2.1.1 | Tipologia e classificazione | 5 |
| 2.2 | Filtri Bayesiani e localizzazione | 8 |
| 2.3 | Filtri Particellari | 12 |
| 2.3.1 | Applicazione dei filtri particellari alla localizzazione | 13 |
| 2.4 | Real Time Particle Filter | 14 |
| 2.4.1 | Ricerca del partizionamento ottimo | 17 |
| 2.4.2 | Problema del bias e motivazioni del clustering | 20 |
| 2.4.3 | Algoritmo del RTPF | 22 |
| 3 | Clustering | 24 |
| 3.1 | Motivazione del clustering nei filtri particellari. | 24 |
| 3.2 | Introduzione al clustering | 25 |
| 3.3 | Principali algoritmi di clustering | 27 |
| 3.3.1 | Algoritmi di Partizionamento | 29 |
| 3.3.2 | Algoritmi Gerarchici | 32 |
| 3.3.3 | Algoritmi basati su funzioni di densità | 36 |
| 3.3.4 | Algoritmi basati su Griglia | 39 |
| 3.3.5 | Algoritmi basati su Modelli | 40 |
| 3.4 | L'algoritmo realizzato | 42 |
| 3.4.1 | Algoritmo di clustering con l'uso di una griglia a risoluzione variabile | 46 |
| 3.4.2 | Implementazione dell'algoritmo di clustering | 47 |

| | | |
|----------|---|-----------|
| 4 | Integrazione del localizzatore nel robot mobile | 50 |
| 4.1 | Componenti Hardware del sistema | 50 |
| 4.1.1 | Il robot mobile | 50 |
| 4.1.2 | Il laser scanner | 52 |
| 4.2 | Componenti Software | 56 |
| 4.2.1 | Smartsoft | 56 |
| 4.2.2 | Il Name Service | 60 |
| 4.3 | Installazione del SICK 200 sul Nomad | 61 |
| 4.4 | La piattaforma software | 62 |
| 4.4.1 | Il localizzatore | 62 |
| 4.4.2 | Il Nomad Server | 64 |
| 4.4.3 | Position Client | 65 |
| 4.5 | Soluzione alternativa per la Piattaforma Software | 66 |
| 5 | Il collaudo del sistema di localizzazione | 69 |
| 5.1 | Prove di simulazione | 70 |
| 5.1.1 | Formazioni di artefatti nella distribuzione di campioni | 72 |
| 5.2 | Sperimentazione sul Robot Mobile | 73 |
| 5.2.1 | Rilevamento della posizione reale del robot | 73 |
| 5.2.2 | Il sistema di tracking realizzato | 76 |
| 6 | Conclusioni | 86 |
| | Appendici | 87 |
| | Bibliografia | 87 |

Capitolo 1

Introduzione

La robotica mobile è la disciplina che studia l'interazione con l'ambiente dei robot, dispositivi gestiti in modo automatico mediante sistemi informatici. Le applicazioni in cui la robotica è stata impiegata con successo sono molteplici e spaziano in numerosi settori. Si considerino, ad esempio, i robot impiegati in missioni esplorative su altri pianeti, i manipolatori meccanici utilizzati in ambito industriale, veicoli capaci di circolare autonomamente e, ancora, gli apparati robotici impiegati nelle sale operatorie. L'obiettivo più ambizioso di questa disciplina è quello di realizzare sistemi capaci di svolgere compiti sempre più complessi in piena autonomia interagendo con ambienti non strutturati e quindi anche di muoversi pianificando la traiettoria da seguire. La capacità di localizzarsi, cioè la stima delle coordinate rispetto ad un riferimento è, quindi, un requisito fondamentale per un robot mobile.

Un robot deve spesso misurarsi con ambienti dalla dinamica imprevedibile la cui rappresentazione è affetta da incertezza. Essa è dovuta delle inevitabili approssimazioni applicate al modello utilizzato, alla quali si aggiunge anche quella derivante dalla rumorosità delle misure sensoriali e dalle semplificazioni introdotte dagli algoritmi utilizzati dal sistema di elaborazione. Per questa ragione la strategia vincente vede l'utilizzo di metodi probabilistici per rappresentare le variabili in gioco. Quindi la rappresentazioni delle grandezze di interesse è affidata a variabili aleatorie e le relazioni causa effetto da opportune densità di probabilità.

In letteratura, uno degli approcci che si sono rivelati più efficaci è quello che fa uso dei filtri Bayesiani: essi permettono di stimare ricorsivamente la distribuzione di

probabilità dello stato del robot rappresentato dalla sua posizione ed orientamento (belief) mediante l'impiego di modelli che rappresentano la cinematica del robot e l'apparato sensoriale. La caratteristica distintiva dei filtri bayesiani è la modalità con la quale il belief viene rappresentato. Di particolare rilievo sono i filtri particellari che sfruttano un'approssimazione discreta del belief costituita da un insieme di campioni della distribuzione a cui è associato un *Importance weight*. L'algoritmo alla base del loro funzionamento prevede che ad ogni iterazione venga generato un insieme di campioni che rappresenta la previsione dello stato del sistema per poi effettuare un intervento di correzione sfruttando le informazioni reperite dai sensori. Nel contesto dei filtri particellari è particolarmente importante riuscire ad estrarre dall'insieme di campioni un'informazione sintetica che rappresenti lo stato del robot e cioè la posizione e l'orientamento. Particolarmente efficace in tal senso, è l'applicazione di algoritmi di clustering capaci di riconoscere la formazione di agglomerati di campioni nello spazio, in quanto le particelle tendono a raggrupparsi intorno agli stati con maggiore verosimiglianza con lo stato reale del sistema. La progettazione di un algoritmo di clustering richiede la conoscenza del contesto, in cui va applicato in quanto non esiste un unico approccio alla ricerca di cluster e pertanto è necessario scegliere quello corretto. Gli algoritmi di clustering possono differire gli uni dagli altri anche per il tipo dei requisiti richiesti per la loro applicazione. L'utilizzo con i filtri particellari, in particolare, richiede che la ricerca possa essere praticata senza conoscere a priori la quantità di cluster in quanto, il loro numero è uno dei motivi per il quale viene applicato il clustering.

L'applicazione degli algoritmi di clustering si dimostra utile anche in contesti particolari come quello del *Real Time Particle Filter* (RTPF) che è una particolare evoluzione del filtro particellari orientata verso applicazioni in tempo reale. Nel RTPF è particolarmente sentito il problema del *bias* che viene attenuato grazie al riconoscimento di cluster di particelle da far evolvere in modo indipendente gli uni dagli altri.

L'obiettivo di questo lavoro di tesi è la realizzazione di un algoritmo di clustering con cui integrare un localizzatore basato su filtri particellari e la sua valutazione in ambito simulativo e in un contesto reale. Per raggiungere tale scopo è stata realizzata una struttura di controllo per una piattaforma robotica nella quale è stato integrato il localizzatore realizzato.

La tesi è organizzata in modo seguente. Nel capitolo 2 viene discusso il problema della localizzazione con e viene illustrato in che modo l'utilizzo del clustering apporta alla localizzazione mediante filtri particellari. Il capitolo 3 è dedicato all'introduzione dei più diffusi algoritmi per la ricerca di cluster e alla descrizione di quello implementato in questo lavoro di tesi. Nel capitolo 4 viene descritta la piattaforma software realizzata per l'utilizzo del localizzatore su un robot mobile reale con alcuni cenni sul funzionamento del laser scanner impiegato come sensore. Il capitolo finale, il 5, raccoglie i risultati ottenuti in simulazione e descrive lo strumento software che sfruttando la visione artificiale, viene utilizzato per rilevare la reale posizione del robot al fine di confrontarla con quella indicata dal localizzatore nel corso delle prove sperimentali. Infine, questo elaborato si chiude con le conclusioni fatte sul lavoro svolto e alcune indicazioni per gli sviluppi futuri.

Capitolo 2

Cenni sui localizzatori bayesiani

2.1 Il problema della localizzazione

La robotica mobile ha tra i suoi principali obiettivi la realizzazione di apparati mobili in grado di navigare in modo autonomo nell'ambiente in cui si trovano ad operare. Le applicazioni sono varie e molteplici: dall'esplorazione di pianeti lontani alla pulizia di una stanza. La navigazione in un ambiente presuppone almeno la conoscenza della posizione occupata istante per istante. Un robot mobile ha bisogno di conoscere la propria posizione per capire come muoversi. La localizzazione di un robot è il problema della stima delle coordinate della sua posizione, espresse rispetto ad un sistema di riferimento esterno e solidale con l'ambiente in cui si localizza. I dati impiegati per le operazioni di localizzazione provengono da sensori che forniscono una lettura dello stato dell'ambiente (*sorgenti eterocettive*) e da altri (*sorgenti propriocettive*) che danno indicazioni sullo stato interno del robot, come la velocità dei motori, l'angolo di sterzata delle ruote o la distanza percorsa. Tutti i dati, sia propriocettivi che eterocettivi, sono affetti da rumore e incertezza e questo è ciò che rende la localizzazione un problema da trattare con strumenti statistici, che siano in grado di tenere conto di tali effetti aleatori. Risultano invece difficilmente applicabili i metodi deterministici.

In letteratura è universalmente accettata l'ipotesi di Markov (o di stato completo). Tale ipotesi ha due chiavi di lettura: la prima è quella di considerare un ambiente statico o lentamente varibile, mentre la seconda è l'ipotesi che l'ambiente sia

chiuso e rappresentabile nella sua interezza dall'elaboratore di bordo del robot. L'ipotesi di stato completo può apparire restrittiva, ma nella maggior parte dei casi pratici si rivela piuttosto ragionevole. Infatti il mondo, anche se non è statico, varia generalmente molto più lentamente della posizione del robot; inoltre è spesso possibile ricavare all'interno di un ambiente aperto un'area circoscritta da punti di riferimento. Naturalmente esistono casi in cui non è possibile applicare l'ipotesi di stato completo: si pensi ad un robot che deve navigare in un ambiente in cui possono presentarsi ostacoli improvvisi come ad esempio esseri umani in movimento. Di certo non è possibile assumere che tale ambiente sia statico. In alternativa lo si potrebbe continuare a considerare tale a patto di adattare opportunamente il modello del sistema in modo che tenga conto di eventi improvvisi.

La figura 2.1 rappresenta lo schema generale adottato dai più comuni algoritmi di localizzazione. Si può notare come la stima dello stato (posizione e orientazione) del sistema venga ottenuta integrando le due sorgenti sensoriali citate in precedenza: il risultato dell'odometria e le osservazioni sensoriali. Il processo si articola, quindi, in due fasi: la predizione e la correzione mediante le percezioni sensoriali, sfruttando la mappa come piano di confronto. Si badi che lo stato del sistema è un dato potenzialmente variabile e pertanto, lo schema si presenta come ricorsivo.

2.1.1 Tipologia e classificazione

In precedenza si è accennato come il problema della localizzazione possa presentarsi in maniera variegata dipendentemente dal contesto e dei dati a disposizione. La tabella 2.1 mostra in sintesi quattro varietà del problema.

| Tipo | Dati a disposizione | conoscenze errate |
|-------------------------|---------------------------|--------------------|
| Position Tracking | Mappa, Posizione iniziale | |
| Lost Robot Problem | Mappa | |
| Kidnapped Robot Problem | Mappa | Posizione iniziale |
| SLAM | | |

Tabella 2.1: Versioni del problema di localizzazione.

Il **Position Tracking** è il problema più semplice fra i quattro elencati in quanto sia la mappa dell'ambiente che la posizione di partenza del robot sono noti. Se

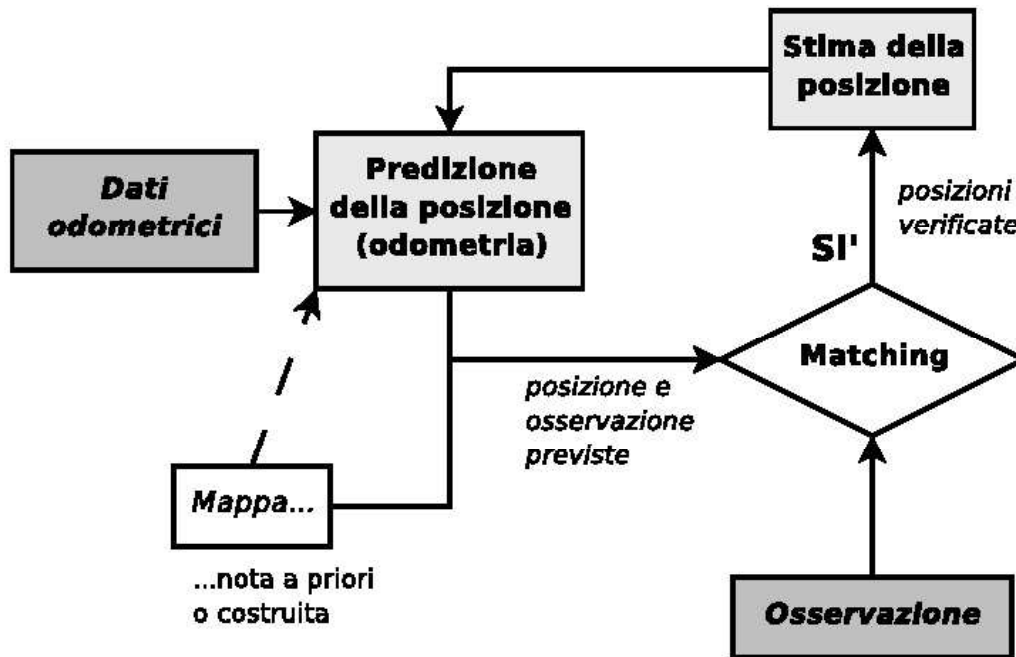


Figura 2.1: Schema di principio della localizzazione

non ci fosse incertezza nei dati dei sensori, sarebbe sufficiente la sola odometria per mantenere aggiornata nel tempo l'informazione di localizzazione. La posizione deve invece essere aggiornata con i dati odometrici e poi validata con i dati sensoriali. La figura 2.2 mostra la traiettoria di un robot e le ipotesi di posizione fatte sulla base dei soli dati odometrici, senza cioè validazione. Si può notare come le particelle siano sempre più sparse mano a mano che il robot avanza. Tanto più è sparsa è la loro distribuzione tanto maggiore è l'incertezza sulla posizione. L'applicazione della correzione con i dati sensoriali produce nuvole di particelle più concentrate e pertanto una migliore stima della posizione del robot.

Nel **Lost Robot Problem** (noto anche come *Global Localization Task*) il robot non ha informazioni sulla propria posizione iniziale per cui le maggiori difficoltà nella localizzazione si presentano nelle prime fasi. Per questo tipo di problemi risultano determinanti sia la capacità di formulare ipotesi capaci di coprire l'intero spazio degli stati, sia la presenza di caratteristiche (*features*) facilmente riconoscibili da parte dei sensori con i quali il robot è equipaggiato. Ciò richiede, per esempio, che la distribuzione iniziale delle variabili aleatorie che rappresentano le coordina-

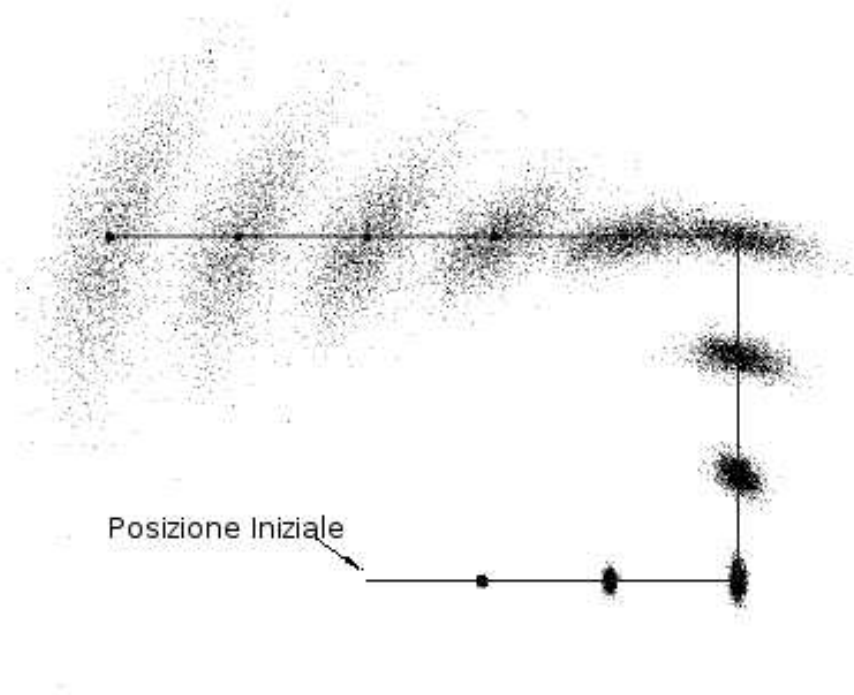


Figura 2.2: Determinazione della posizione mediante la sola odometria: effetto del rumore in un problema di Position Tracking. Immagine tratta da [1]

te del robot, sia uniformemente distribuita sullo spazio degli stati; la distribuzione normale sarebbe inadeguata per garantire una copertura completa ed uniforme. Va inoltre osservato che in ambienti simmetrici la convergenza degli algoritmi è più difficile.

Per un problema di localizzazione globale la distribuzione delle particelle potrebbe seguire il corso mostrato in figura 2.3.

Il **Kidnapped Robot Problem** (o *Relocalization Problem*) si incontra quando il robot è convinto di una posizione iniziale che non è quella corretta. La denominazione del problema è dovuta all'ipotetico scenario nel quale il robot che sta effettuando con successo il tracking, viene trasportato ("rapito") in un'altra localizzazione. La difficoltà è dovuta alla necessità di individuare l'errore per poi recuperare l'ipotesi corretta. La risoluzione del *Kidnapped Robot Problem* è sicuramente un indicatore della robustezza dell'algoritmo di localizzazione e della capacità di recupero da un fallimento.



Figura 2.3: Immagine tratta da [1] che mostra la distribuzione di particelle in tre fasi della localizzazione.

Lo **SLAM**, *Simultaneous Localization And Mapping*, è il problema di maggiore complessità in quanto comprende anche l'attività di costruzione della mappa dell'ambiente. Non è, infatti, possibile compiere una localizzazione senza informazioni sulla struttura dell'ambiente. Le tecniche di mappatura esulano dagli obiettivi di questo lavoro di tesi per cui si rimanda alla letteratura [2] per un approfondimento del problema.

2.2 Filtri Bayesiani e localizzazione

Nel precedente paragrafo è stato accennato al problema del rumore nelle misure sensoriali e di come ciò abbia favorito il diffondersi di metodologie probabilistiche per trattare il problema della localizzazione. Il punto di partenza di molti metodi di localizzazione ([1], [3], [4] sono alcuni esempi) è collocabile nei filtri Bayesiani. Il filtro Bayesiano opera sotto l'ipotesi di *Markov* [5] e si basa sull'idea di stimare la densità di probabilità dello stato del sistema condizionata dalla lettura di sensori e dall'odometria. Da ora in avanti la distribuzione a posteriori dello stato x verrà

indicata col termine come *belief* che, matematicamente, si esprime come:

$$Bel(x_t) = p(x_t | z_t, u_{t-1}, z_{t-1}, u_{t-2}, \dots, z_0, m) \quad (2.1)$$

$Bel(x_t)$ è una distribuzione di probabilità sullo spazio degli stati del sistema, x_t rappresenta lo stato al tempo t , z_t l'osservazione dell'ambiente (le distanze misurate con sonar e laser sono un esempio), u_t è l'odometria fra t e $t - 1$ che rappresenta la grandezza di controllo e m è la mappa.

I filtri Bayesiani effettuano una stima del *belief* ricorsivamente. Si è soliti indicare con il termine *a priori* la distribuzione di cui si dispone all'inizio di una iterazione, prima di condizionarla all'ultimo rispetto all'ultimo dato iniziale ottenuto. La distribuzione iniziale viene scelta uniforme in quanto non si dispone di alcun dato sulla posizione iniziale del robot e nessuna lettura sensoriale. Ad esempio, nella *Global Localization* $Bel(x_0)$ è distribuito uniformemente su tutta l'area della mappa che rappresenta l'unico elemento di condizionamento.

L'equazione 2.1 può essere trasformata applicando alcuni teoremi. Per prima cosa viene applicata la regola di Bayes:

$$Bel(x_t) = \frac{p(z_t | x_t, u_{t-1}, \dots, z_0, m)p(x_t | u_{t-1}, \dots, z_0, m)}{p(z_t | u_{t-1}, \dots, z_0, m)} \quad (2.2)$$

Si può notare che il denominatore è costante rispetto a x_t quindi è possibile alleggerire la notazione di 2.2 riscrivendola come:

$$Bel(x_t) = \eta p(z_t | x_t, u_{t-1}, \dots, z_0, m)p(x_t | u_{t-1}, \dots, z_0, m) \quad (2.3)$$

Dove, naturalmente, si è posto $\eta = p(z_t | u_{t-1}, \dots, z_0, m)^{-1}$

L'equazione 2.3 può essere semplificata ulteriormente ricordando che i filtri di Bayes operano sotto l'ipotesi di Markov, il che equivale a svincolare la densità di probabilità di z_t da tutto ciò che non è lo stato corrente.

$$Bel(x_t) = \eta p(z_t | x_t, m)p(x_t | u_{t-1}, \dots, z_0, m) \quad (2.4)$$

E' possibile esplicitare il termine più a destra nella sua forma integrale e ottene-

re, quindi, l'equazione 2.5:

$$Bel(x_t) = \eta p(z_t | x_t, m) \int p(x_t | u_{t-1}, x_{t-1}, \dots, z_0, m) p(x_{t-1} | u_{t-1}, \dots, z_0, m) dx_{t-1} \quad (2.5)$$

Ancora una volta è possibile applicare l'ipotesi di sistema Markoviano e riscrivere la precedente equazione come:

$$Bel(x_t) = \eta p(z_t | x_t, m) \int p(x_t | u_{t-1}, x_{t-1}, m) p(x_{t-1} | u_{t-1}, \dots, z_0, m) dx_{t-1} \quad (2.6)$$

Definendo, poi, $z^{-1} := \{z_0, \dots, z_{t-1}\}$ e $u^{t-1} := \{u_0, \dots, u_{t-1}\}$ si giunge a:

$$Bel(x_t) = \eta p(z_t | x_t, m) \int p(x_t | u_{t-1}, x_{t-1}, m) p(x_{t-1} | z^{t-1}, u^{t-1}, m) dx_{t-1} \quad (2.7)$$

Osservando, infine, che il termine più a destra è $Bel(x_{t-1})$ si ottiene:

$$Bel(x_t) = \eta p(z_t | x_t, m) \int p(x_t | u_{t-1}, x_{t-1}, m) Bel(x_{t-1}) dx_{t-1} \quad (2.8)$$

Di iterazione in iterazione la 2.8 consente di calcolare $Bel(x_t)$ conoscendo le probabilità $p(z_t | x_t, m)$ e $p(x_t | u_{t-1}, x_{t-1}, m)$ che sono anche definite *Sensor Model* e *Motion Model* rispettivamente. Il *Motion Model* è l'espressione della dinamica del controllo, mentre il *Sensor Model* è l'espressione per le misure effettuate con i sensori. Tali modelli danno una rappresentazione probabilistica della dinamica e delle letture sensoriali in quanto entrambe sono affette da errori ed incertezze.

I filtri Bayesiani possono avere diverse implementazioni che differiscono per il modo in cui viene rappresentato $Bel(x_t)$. La figura 2.4 mostra in sintesi le principali rappresentazioni e le loro proprietà. Una descrizione più dettagliata ed esaustiva si può trovare in [5].

Nella categoria dei filtri bayesiani ricadono numerosi osservatori di stato. La figura 2.4 presenta una tassonomia abbastanza completa delle metodologie bayesiane

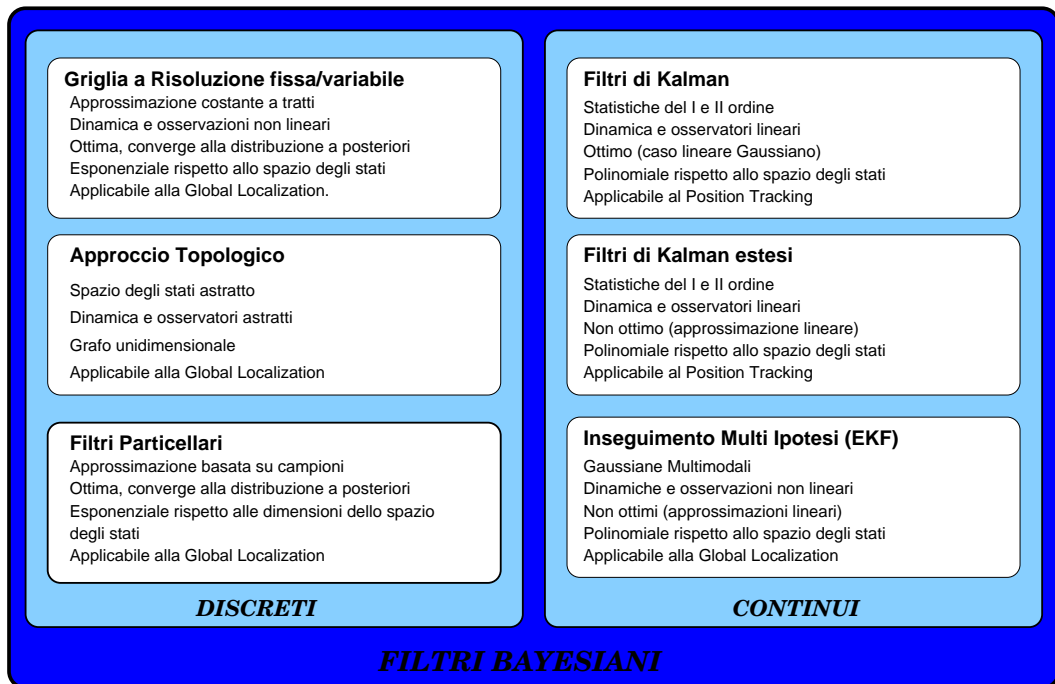


Figura 2.4: Proprietà delle più comuni implementazioni dei filtri bayesiani basate sull'assunzione dell'ipotesi di Markov

ne. Senza entrare nei dettagli si osserva che i filtri sono classificati sulla base delle modalità con cui la densità di probabilità dello stato è rappresentata. I filtri continui forniscono una rappresentazione parametrica della PDF e derivano dal filtro di Kalman. Gli altri approcci, invece, adottano una rappresentazione discreta che approssima quella reale dello stato.

Di particolare interesse per questo lavoro di tesi sono i *filtri particellari*. Essi rientrano nella categoria degli algoritmi discreti. Il loro principale vantaggio risiede nella possibilità di rappresentare qualunque distribuzione con un numero variabile di campioni scelto in relazione alle prestazioni desiderate. Il principale svantaggio è quello che il numero di ipotesi da valutare è crescente con dimensione dello spazio degli stati. Nel caso del localizzatore di questa tesi la dimensione dello spazio è tre: posizione in una mappa 2d e orientamento del robot. Nel paragrafo successivo verranno descritti con maggiore dettaglio i filtri particellari e, inoltre, verrà presentato l'algoritmo con il quale vengono realizzati.

2.3 Filtri Particellari

I *Filtri Particellari* sono una variante discreta dei filtri Bayesiani nei quali $Bel(x_t)$ è rappresentato come un set S_t di N_p campioni pesati w_i :

$$S_t = \{ \langle x_t^{(i)}, w_t^{(i)} \rangle \mid i = 1, \dots, N_p \}$$

dove i pesi sono tali che:

$$w_t^{(i)} \in R^+ \setminus \{0\} \text{ } \text{e} \sum_i w_i = 1$$

Ogni $x_t^{(i)}$ è un campione dello stato e $w_t^{(i)}$ è il suo *importance weight*. Il filtro particellare semplice realizza un filtraggio bayesiano inserendo nell'algoritmo anche una procedura di discretizzazione che serve per generare il sopracitato set di campioni S_t . Una delle tecniche di campionamento utilizzate più di frequente è *Sequential Importance Sampling with Resampling* (SISR) che prevede una prima fase di campionamento, alla quale ne segue una di ricampionamento ([6], [7], [8]).

Algoritmo 1 Filtro particellare

Require: $S_{t-1} = \{ \langle x_{t-1}^{(i)}, w_{t-1}^{(i)} \rangle \mid i = 1, \dots, n \}$ che rappresenta $Bel(x_{t-1})$, misura del controllo u_{t-1} , osservazione z_t

- 1: $S_t := \emptyset, \alpha := 0$
 - 2: **for** $i := 0, \dots, n$ **do**
 - 3: Campiona un indice j dalla distribuzione discreta data dai pesi di S_{t-1}
 - 4: Campiona $x_t^{(i)}$ da $p(x_t \mid x_{t-1}, u_{t-1})$ dove il condizionamento è dato dal campione $x_{t-1}^{(j)}$ e da u_{t-1}
 - 5: $w_t^{(i)} := p(z_t \mid x_t^{(i)})$
 - 6: $\alpha := \alpha + w_t^{(i)}$
 - 7: $S_t := S_t \cup \{ \langle x_t^{(i)}, w_t^{(i)} \rangle \}$
 - 8: **end for**
 - 9: **for** $i := 1, \dots, n$ **do**
 - 10: $w_t^{(i)} := \frac{w_t^{(i)}}{\alpha}$
 - 11: **end for**
 - 12: **return** S_t
-

Una iterazione del filtro particellare elementare, è descritta dall'algoritmo 1.

Il punto di partenza di ogni iterazione sono il set di campioni S_{t-1} che rappresenta $Bel(x_{t-1})$, la misura di controllo u_{t-1} e l'osservazione sensoriale z_t . All'inizializzazione (1), segue la generazione di N_p campioni (2-8) che rappresentano il *belief* a posteriori. In particolare alla linea 3 viene determinato quali campioni devono essere estratti da S_{t-1} . La probabilità di estrazione cresce con il peso della particella. Il passo successivo è quello di utilizzare gli N_p campioni e u_t per predire il prossimo stato (4): $x_t^{(i)}$. La predizione avviene mediante il campionamento della dinamica di sistema, rappresentata dalla funzione densità $p(x_t | x_{t-1}, u_{t-1})$. L'operazione successiva (5) assegna i pesi agli N_p campioni e utilizza, come già detto in precedenza, l'*importance sampling*. La $p(z_t | x_t^{(i)})$ rappresenta l'*Importance Weight* dell'*i-esimo* campione ed è calcolata come rapporto fra le così dette *target* e *proposal distribution*, il cui ruolo è rivestito da $Bel(x_t)$ e dalla predizione dello stato. Il passo 6 calcola il fattore di normalizzazione del peso del campione che poi, viene inserito in S_t . Infine (linea 10) vengono normalizzati i pesi dei campioni e restituisce il set completo. È, inoltre, importante sottolineare che S_t approssima lo stato e che tale approssimazione è tanto migliore quanto maggiore è il numero dei campioni; in particolare si ha convergenza alla distribuzione reale per $N_p \rightarrow \infty$.

2.3.1 Applicazione dei filtri particellari alla localizzazione

In una tipica applicazione di localizzazione, lo stato del robot è rappresentato da un vettore che contiene le due coordinate cartesiane che definiscono la posizione del robot nel piano e un angolo che ne dà l'orientamento. Sebbene esistano applicazioni in cui lo spazio in cui il robot si deve localizzare è tridimensionale, nel seguito di questa tesi si farà riferimento solo a casi bidimensionali.

Le misure z_t derivano dall'osservazione dell'ambiente tramite sensori quali sonar, laser, bussole e bumper. A seconda dei, o dei, sensori utilizzati la natura dei dati può variare: i laser e i sonar restituiscono delle distanze, le bussole un orientamento e i bumper rilevano contatti con ostacoli.

Le informazioni di controllo u_t forniscono informazioni sullo stato interno del robot: spostamenti e velocità per esempio, ma anche orientamento e angolo di torretta se questa è presente.

Il modello cinematico $p(x_t | x_{t-1}, u_{t-1})$ rappresenta la probabilità che lo stato di x_t

venga raggiunto condizionata dallo stato precedente x_{t-1} e dalle rilevazioni odometriche u_{t-1} . Il modello cinematico tiene conto dei fattori di rumore (generalmente gaussiano-bianco). In altre parole, nota la posizione corrente, la velocità e la direzione di marcia è possibile eseguire una stima del successivo stato.

Il modello sensoriale $p(z_t | x_t)$ descrive la probabilità di ottenere la lettura z_t data la posizione x_t e viene estrapolato dalla mappa, tenendo conto del modello del rumore dei rilevamenti sensoriali.

Per concludere questa breve introduzione ai filtri particellari si osservi nuovamente la figura 2.3. Essa mostra la distribuzione dei campioni su una mappa in tre istanti diversi: nel primo nota una copertura pressoché uniforme della mappa corrispondente alla fase iniziale; nel successivo la formazione di ipotesi di localizzazione; nell'ultimo una distribuzione ormai unimodale che definisce l'ipotesi finale. Si mette in evidenza che in un normale filtro particellare il numero dei campioni è costante ma una volta risolta la posizione il problema è passato dalla *Global Localization* (fig. 2.3,a) al più semplice *Position Tracking* (fig. 2.3,b). L'adattamento delle risorse utilizzate, ossia del numero di campioni, apporta benefici al costo computazionale complessivo dell'algoritmo. In [3] è mostrata una tecnica basata su *KDL-Sampling* per stimare il numero di particelle necessario ad ogni iterazione del filtro particellare.

2.4 Real Time Particle Filter

Il filtro particellare semplice lavora con uno schema rigido che prevede una fase di acquisizione delle misure sensoriali ed odometriche, seguita dalla generazione di $Bel(x_t)$ ottenuta a seguito di una predizione (campionamento della nuova distribuzione modificata dalla dinamica) e di una correzione (calcolo degli importance weight). Tali operazioni sono compiute su tutti i campioni indipendentemente dal sopraggiungere di nuovi dati. In un contesto *Real Time* tale procedimento può richiedere troppo tempo rispetto all'intervallo di interarrivo dei nuovi dati sensoriali. La soluzione di riferimento per il localizzatore sviluppato in questa tesi è quella discussa in [4] che verrà identificata col nome di *Real Time Particle Filter* o, più brevemente, *RTPF*.

Il RTPF non possiede le tipiche caratteristiche che dovrebbe avere un'applicazione *Real Time* in quanto non vi è alcuna verifica formale dei vincoli temporali e nessuna analisi di fattibilità. Il RTPF opera solo tenendo conto degli istanti di rilascio dei rilevamenti sensoriali e assumendo come *soft-deadline* il loro periodo. Prima di proseguire con l'analisi del RTPF è bene introdurre la notazione che verrà adottata da qui in avanti.

- T_c è il periodo nel quale avviene la ricezione di un controllo e un flusso di dati sensoriali (sorgente costituita da uno o più sensori).
- N rappresenta il numero dei campioni ed è costante.
- Con T viene indicato il tempo necessario al completamento di una iterazione. A causa delle limitate risorse computazionali può accadere che $T > T_c$. T verrà indicato anche come *estimation window*.
- $k := \lceil \frac{T}{T_c} \rceil$ è il numero di iterazioni nel tempo T che dovrebbero essere svolte per gestire tutti i flussi di dati dei rilevamenti sensoriali. Il significato di questo valore sarà chiarito successivamente.
- Se $T > T_c$, nell'arco di una finestra avverranno più osservazioni sensoriali e la notazione che verrà adottata sarà z_{t_i} per l' i -esima a lettura di dati entro la finestra T .

Definendo l'*Estimation Window* come l'intervallo di tempo richiesto dal filtro particellare per completare una iterazione su N campioni (N costante), si osservi la figura 2.5 che mostra tre dei più semplici approcci al compromesso fra prestazioni e risorse computazionali da adottare qualora $k > 1$, cioè quando nell'arco di una fase di aggiornamento dei campioni si presentano più controlli e rilevamenti sensoriali.

caso a) Questo approccio è definito *Skip Observation* e prevede di scartare le osservazioni e i controlli che arrivano durante il periodo di aggiornamento dei campioni T . Il vantaggio principale è quello della semplicità ma vengono persi dei dati che potrebbero essere particolarmente rilevanti.

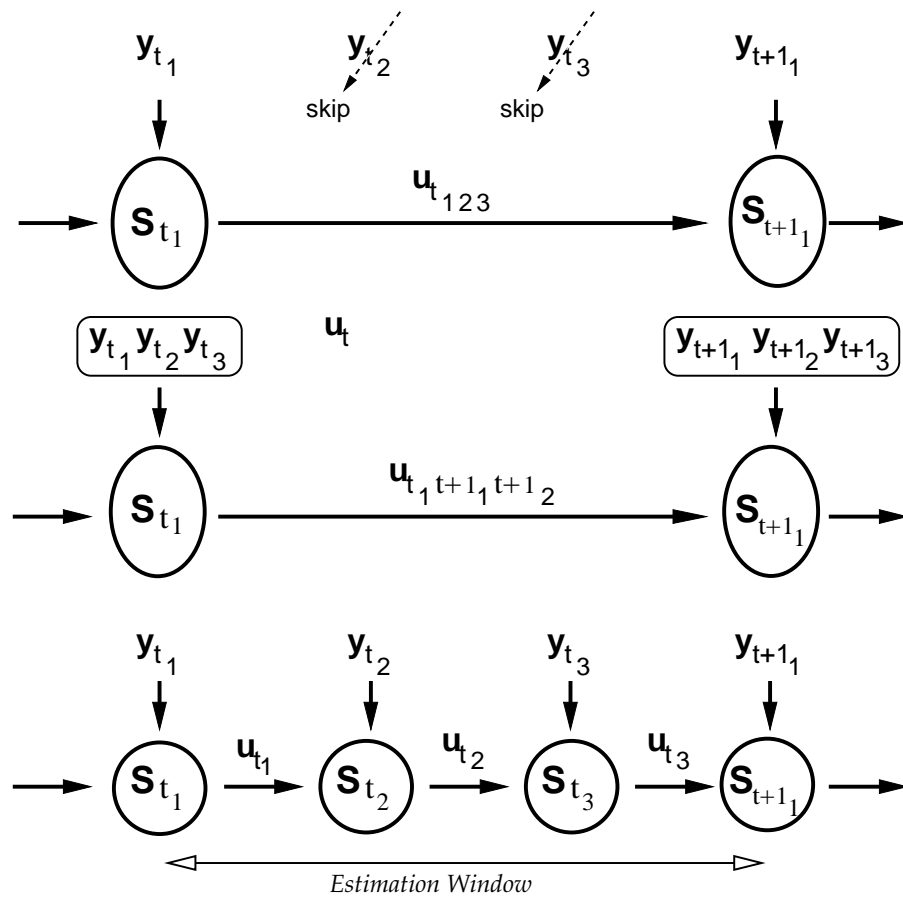


Figura 2.5: Tre differenti strategie per affrontare il problema del compromesso fra risorse computazionali e prestazioni

caso b) In questo caso i dati vengono aggregati con l'ovvio vantaggio di non scartarne alcuno. Tuttavia nella successiva fase di update la mole di dati supera quella di una singola osservazione e bisogna in qualche modo gestirla mediante un processo di sintesi. Il principale limite di questo approccio risiede nell'ipotesi che le osservazioni e i controlli possono essere aggregati in modo ottimo e questo può non essere sempre vero.

caso c) Il terzo approccio prevede che l'aggiornamento dei campioni venga interrotto all'arrivo di un nuovo rilevamento, il che equivale ad eseguire più aggiornamenti parziali (generando meno campioni) nell'arco della finestra T . Questo metodo consente di non perdere alcuna informazione ma può dar luogo alla divergenza del filtro a causa del ridotto numero di particelle utilizzato

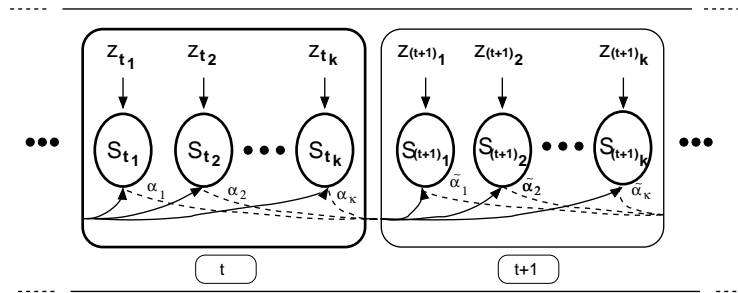


Figura 2.6: Schema di ricampionamento del RTPF. Il $Bel(x)$ risultante è un mix dei set della fi nestra temporale. I pesi α devono essere calcolati in modo opportuno per ridurre al minimo l'errore dell'approssimazione.

nelle fasi intermedie di aggiornamento dello stato.

L'approccio usato nel RTPF di [4] è diverso dai tre appena descritti. L'idea è quella di non tralasciare alcuna fonte di informazione distribuendo le k osservazioni lungo la fi nestra temporale nella quale vengono svolti anche gli aggiornamenti parziali. In ciascuna fi nestra temporale vengono eseguite k osservazioni e vengono generati altrettanti set di campioni ciascuno contenente N/k campioni. Benché somigliante con l'approccio mostrato in 2.5.c, esiste una differenza sostanziale: la distribuzione che rappresenta lo stato del sistema al tempo t_k ($Bel(x_{t_k})$) è costituita da un mix pesato di tutte le partizioni appartenenti alla fi nestra temporale precedente. Detto in altri termini, S_{t_i} viene costruita con campioni provenienti da $S_{t-1_1}, \dots, S_{t-1_k}$. Ciò che, invece, accade in 2.5.c è la creazione di k set non correlati invece che di una sintesi di più set. In figura 2.6 viene illustrata graficamente l'idea appena descritta.

2.4.1 Ricerca del partizionamento ottimo

Il RTPF è un'architettura concepita per realizzare un compromesso fra prestazioni e risorse di calcolo. Ragion per cui la ricerca della migliore combinazione va fatta in relazione alle migliori prestazioni ottenibili: quelle di un filtro particellare semplice senza limiti di risorse computazionali che opera k aggiornamenti dei campioni

nell'arco dell'*estimation window*. Il risultato del filtro particellare al tempo t_k è:

$$Bel_{opt}(x_{t_k}) \propto \int \dots \int \prod_{i=1}^k p(z_{t_i} | x_{t_i}) p(x_{t_i} | x_{t_{i-1}}, u_{t_{i-1}}) Bel(x_{t_0}) dx_{t_0} \dots dx_{t_{k-1}} \quad (2.9)$$

$Bel(x_{t_0})$ rappresenta il belief generato nel corso della precedente finestra. La 2.9 nella sua completezza, è ottenuta mediante l'integrazione di tutte le traiettorie dell'intervallo di stima il cui punto di partenza è $Bel(x_{t_0})$. Ogni traiettoria possiede una propria probabilità che è determinabile a partire dalle informazioni di controllo ($u_{t_0}, u_{t_1}, \dots, u_{t_k}$) e da quelle sensoriali ($y_{t_0}, y_{t_1}, \dots, y_{t_k}$). Il calcolo di $Bel_i(x_{t_k})$ viene eseguito limitando l'integrazione all' i -esima osservazione e l'espressione è molto simile alla 2.9: ancora una volta si tratta di un'integrazione delle possibili traiettorie da $Bel(x_{t_0})$.

$$Bel_i(x_{t_k}) \propto \int \dots \int \prod_{j=1}^k p(z_{t_j} | x_{t_j}) p(x_{t_j} | x_{t_{j-1}}, u_{t_{j-1}}) Bel(x_{t_0}) dx_{t_0} \dots dx_{t_{k-1}} \quad (2.10)$$

$Bel_i(x_{t_k})$ è una predizione con la sola correzione degli ingressi dell'istante t_i . Nell'arco di una singola finestra temporale è possibile ottenere k belief di questo tipo e combinandoli con opportuno peso si ottiene l'approssimazione per $Bel_{opt}(x_{t_k})$ ricercata:

$$Bel_{mix}(x_{t_k} | \alpha) \propto \sum_{i=1}^k \alpha_i Bel_i(x_{t_k}) \quad (2.11)$$

Trovata l'approssimazione di Bel_{opt} , rimangono da determinare i pesi α . Una possibile strada da seguire è quella di minimizzare l'errore dell'approssimazione riducendo al minimo la distanza di Kulback Leibler (KLD) fra Bel_{mix} e Bel_{opt} . Analiticamente:

$$\begin{aligned} \alpha &= \operatorname{argmin}_{\alpha \in A} KL(Bel(\bullet | \alpha) \| Bel_{opt}) = \\ &= \operatorname{argmin}_{\alpha \in A} \int Bel_{mix}(x_{t_k} | \alpha) \log \frac{Bel_{mix}(x_{t_k} | \alpha)}{Bel_{opt}(x_{t_k})} dx_{t_k} \end{aligned} \quad (2.12)$$

Dove con A rappresenta l'insieme di tutti gli α possibili:

$$A = \{\alpha \mid \sum_{i=1}^k \alpha_i = 1, \alpha_i \geq 0\}$$

I metodi che si possono utilizzare per la ricerca degli α ottimi sono svariati: EM [9] (**EM**: Estimation-Maximization) e la discesa lungo il gradiente sono due tecniche piuttosto diffuse. La seconda è quella utilizzata in [4] e consiste nell'individuare una grandezza rispetto alla quale effettuare la minimizzazione; quindi se ne calcola il gradiente rispetto al vettore degli α_i e lo si utilizza per trovare il minimo.

$$J(\alpha) = \int Bel_{mix}(x_{t_k}|\alpha) \log \frac{Bel_{mix}(x_{t_k}|\alpha)}{Bel_{opt}(x_{t_k})} dx_{t_k} \quad (2.13)$$

Il punto di partenza della discesa del gradiente viene scelto arbitrariamente e una possibilità ragionevole è $\alpha_{start} = (\frac{1}{k} \frac{1}{k} \dots \frac{1}{k})^T$. Purtroppo risulta impossibile impiegare l'espressione analitica del gradiente di 2.13 poiché essa richiederebbe il calcolo analitico del Bel_{opt} e pertanto se ne introduce una denominata *Monte Carlo Gradient*. I vari termini che contribuiscono al gradiente sono rappresentati dal rispettivo ammontare degli Importance Weight. Per la distribuzione ottima, anziché analizzare i veri Importance Weight, il peso della traiettoria è ottenuto come prodotto dei pesi calcolati sulle singole partizioni. In sintesi i parametri sono i seguenti.

$$Bel_i = \sum_{m=1}^{N_p} w_{im} \quad (2.14)$$

$$Bel_{mix} = \sum_{p=1}^k \alpha_p Bel_i \quad (2.15)$$

$$Bel_{opt} = \sum_{m=1}^{N_k} \prod_{i=1}^k w_{im} \quad (2.16)$$

Con w_{im} si è indicato il peso dell' m -esima particella dell' i -esima partizione. Con N_k si è indicato il numero di particelle per ogni partizione (N/k). È possibile notare come ogni partizione venga rappresentata dalla somma dei pesi delle particelle che la compongono e ciò determina il fatto che il gradiente privilegi le partizioni il cui peso complessivo è maggiore.

Combinando le 2.14, 2.15 e 2.16 con 2.13 si ottiene l'approssimazione cercata:

$$\frac{\partial J}{\partial \alpha_i} \simeq 1 + Bel_i \log \frac{\sum_{i=1}^k \alpha_i Bel_i}{Bel_{opt}} \quad (2.17)$$

2.4.2 Problema del bias e motivazioni del clustering

La ricerca dell' α ottimo col metodo del gradiente descritto in precedenza, equivale a muoversi nello spazio degli stati lungo il gradiente con passi di lunghezza costante e con un numero di iterazioni fissato (spesso empiricamente). Tale procedimento porta allo scontro con un problema denominato *bias*. Esso è la ragione per la quale la sola ricerca degli α ottimi non produce un aumento delle prestazioni particolarmente elevato nelle applicazioni con robot mobili. La causa del bias va ricercata nella tendenza delle traiettorie a divergere nell'arco dell'*estimation window* causata dall'applicazione di controlli con incertezza crescente nel tempo che incide soprattutto sui campioni degli ultimi set della finestra. Ciò che si riscontra, in definitiva, è la presenza di agglomerati di particelle relativamente più concentrati in S_{t_1} e più sparsi in S_{t_k} . Empiricamente in [3] si è notato che i proprio i campioni in S_{t_k} , hanno in genere peso maggiore e, dunque, a seguito della scesa lungo il gradiente, l'ultima partizione tende ad essere privilegiata in fase di ricampionamento. Per questa ragione la distribuzione dei campioni di S_{t_k} a propagarsi anche nelle estimation window successive introducendo un *bias*.

Per comprendere meglio gli effetti del *bias*, si osservi la figura 2.7. È facile notare che, come preannunciato, col procedere delle iterazioni lo spargimento delle particelle è via via maggiore. In [4] l'idea proposta per ridurre gli effetti del *bias*

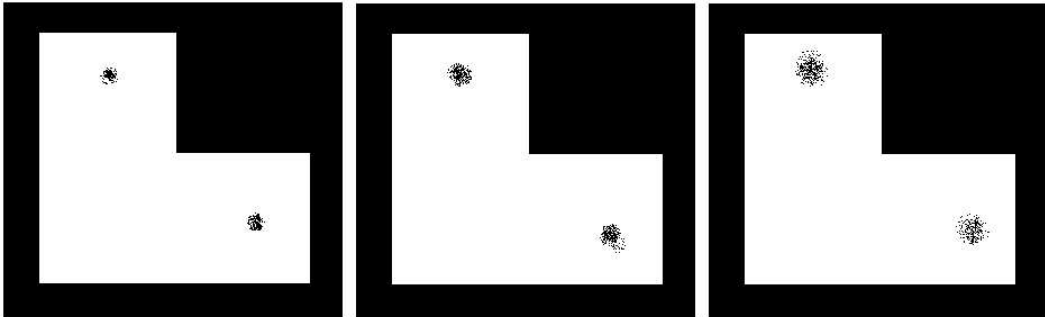


Figura 2.7: Effetti del bias nell'arco di un'*estimation window* con tre step

consiste nel separare le particelle in *cluster* e di valutare separatamente i loro *belief* calcolando per ogni raggruppamento le grandezze 2.14 e 2.16. La soluzione del *clustering* porta benefici in quanto contrasta la tendenza del metodo del gradiente a convergere con eccessiva rapidità ad un minimo locale. Tendenza questa, che è responsabile dell'eccessivo peso dato a S_{t_k} anche quando gli α_i dovrebbero essere più omogenei, come nel caso di ambienti simmetrici come quello di figura 2.7. Infatti gli ambienti simmetrici danno luogo ad osservazioni molto simili nell'arco della finestra di valutazione e pertanto i pesi delle corrispondenti partizioni dovrebbero essere altrettanto simili senza che α_k tenda a dominare. I cluster che vengono individuati all'interno delle partizioni S_i rappresentano un addensamento di particelle intorno ad una ipotesi di localizzazione. Le particelle appartenenti al cluster di maggior peso saranno quelle maggiormente importanti nel metodo del gradiente.

Quello del *bias* non è il solo aspetto a trarre giovamento dall'applicazione del *clustering* all'insieme di particelle. Infatti, l'effetto globale dell'utilizzo dei cluster può essere descritto come il mantenimento più prolungato di più ipotesi di quanto non sarebbe possibile altrimenti [10] in quanto, come è già accennato, i campioni generati dalla distribuzione a posteriori tendono a convergere rapidamente all'ipotesi più probabile. Tale comportamento è particolarmente deleterio in ambienti che presentano forti simmetrie perchè potrebbe provocare l'estinzione della nuvola di particelle corrispondente all'ipotesi corretta prima di ottenere i dati per prendere la decisione esatta. Individuare i cluster e farli evolvere separatamente utilizzando individualmente le 2.14, 2.15 e 2.16 consente al localizzatore di non convergere troppo frettolosamente verso una posizione. Infatti per un filtro particellare è impossibile recuperare a partire da un'ipotesi errata se non c'è almeno una particella nei pressi della posizione corretta [1].

L'applicazione del *clustering* al set di campioni conduce il localizzatore a lavorare su due livelli: quello della totalità di particelle e quello dei set definiti dai cluster. Il singolo cluster viene a rappresentare il *belief* della posizione del robot nella zona che circoscrive [10]. Il peso da considerare è quello complessivo del cluster che è calcolato a partire da quello delle particelle che contiene. Sarà il cluster di maggior peso ad avere la maggiore probabilità di corrispondere alla posizione reale del robot. Ad ogni iterazione del filtro, con l'aggiornamento del set di campioni, la morfologia dei cluster può cambiare e quello tipicamente dovrebbe accadere è che alcuni di essi

si estinguono in favore di quello che corrisponde alla posizione corretta.

Come nota finale a questo paragrafo va detto che svolgere un clustering di svariate migliaia o centinaia di particelle può avere un costo computazionale piuttosto rilevante: se da un lato la localizzazione può essere avvantaggiata dal clustering, dall'altro è da valutare la possibilità di utilizzo in relazione alle risorse computazionali disponibili.

2.4.3 Algoritmo del RTPF

Quanto descritto finora ha introdotto solo in termini generali il funzionamento del RTPF. In questo paragrafo viene descritto l'algoritmo vero e proprio con una sommaria descrizione di riepilogo.

L'algoritmo richiede alcune precondizioni fra cui il numero di set per ogni finestra k , il numero di campioni generati dall'osservazione N_p e l'osservazione corrente z_t . Inoltre, trattandosi di un algoritmo iterativo, sono necessari alcuni dati della precedente finestra: i set di campioni, i pesi delle partizioni e i controlli. Dopo la fase di inizializzazione si incontrano due cicli annidati: il principale opera sulle partizioni della finestra e quello annidato sul numero di campioni da generare per ogni singola partizione. Il numero di campioni contenuti nella partizione è dipendente dal peso di quella con lo stesso i della precedente finestra (5).

All'interno di questo secondo ciclo sono collocate le fasi di ricampionamento (6), di predizione (7) correzione (8)

Segue la fase di normalizzazione dei pesi delle particelle della corrente partizione proprio come avviene nel filtro particellare ma sulla totalità dei campioni. Le linee fra 15 e 19 descrivono le operazioni di passaggio fra una finestra temporale alla successiva. La funzione MIX_OPT serve per la ricerca del mix ottimo e potrebbe essere l'algoritmo del gradiente descritto nel paragrafo 2.4.2 oppure uno alternativo, come quello proposto nell'appendice A di [11].

Algoritmo 2 Algoritmo RTPF

Require: $k, l, N_p, S_{(t-1)_1}, \dots, S_{(t-1)_k}, \alpha^{t-1}, u_{(t-1)_1}, \dots, u_{(t-1)_k}, u_{t_1}, \dots, u_{t_{j-1}}, z_{t_i}$

- 1: $t := t_1$
- 2: $S_{t_i} = \emptyset, \mu = 0$
- 3: **for** $i = 1, \dots, k$ **do**
- 4: $n_i = \alpha_{(t-1)_i} N_p$
- 5: **for** $m = 1, \dots, n_i$ **do**
- 6: */* Ricampionamento: */*
 campiona l'indice j dalla distribuzione di $S_{(t-1)_i}$
- 7: */* Predizione: */*
 campiona $x_{t_i}^{(m)}$ da $p(x_{t_i} | x_{(t-1)_i}, u_{(t-1)_i}, \dots, u_{t_{l-1}})$ usando $x_{(t-1)_i}^{(j)}$ ed i controlli compresi fra gli istanti $(t-1)_i$ e t_{l-1}
- 8: $w_{t_i}^{(m)} = p(z_{t_i} | x_{t_i}^{(m)})$
- 9: $\mu = \mu + w_{t_i}^{(m)}$
- 10: $S_{t_i} = S_{t_i} \cup \{(x_{t_i}^{(m)}, w_{t_i}^{(m)})\}$
- 11: **end for**
- 12: **end for**
- 13: normalizza i pesi dell'attuale partizione, $w_{t_i}^{(m)} = w_{t_i}^{(m)} / \mu$ con $m = 1, \dots, N_p$
- 14: $l = (l + 1) \bmod k$
- 15: **if** $l == 0$ **then**
- 16: calcola i nuovi coefficienti di mix: $\alpha_{t_1}, \dots, \alpha_{t_k} = OPT_MIX(T, k)$
- 17: calcola il numero di pesi richiesti n_χ
- 18: $k = \left\lceil \frac{n_\chi}{N_p} \right\rceil$
- 19: **end if**
- 20: **return** S_{t_i}

Capitolo 3

Clustering

3.1 Motivazione del clustering nei filtri particellari

I localizzatori basati su filtri particellari consentono di descrivere la distribuzione dello stato del sistema sotto forma di insieme di campioni. Nelle applicazioni usuali è necessario estrarre un dato sintetico da questa distribuzione. Nel precedente capitolo è stato descritto come i campioni tendano ad assensarsi formando gruppi distinti. L'impiego di algoritmo di clustering consente di individuarli in modo da poterle trattare separatamente.

Il mantenimento di più ipotesi per un tempo opportuno gioca un ruolo fondamentale nelle prestazioni di un localizzatore particellare in quanto, se non esiste almeno un particella abbastanza vicina alla posizione corretta, non è possibile che l'algoritmo vi converga. Riguardo questa caratteristica dei filtri particellari, particolarmente significativo è l'esempio presentato nel paragrafo 3 di [1] dove viene mostrato che l'utilizzo di sensori troppo precisi conduce a distribuzioni a bassa varianza che possono portare al piazzamento di particelle troppo lontane dal target e quindi alla mancata convergenza all'ipotesi corretta.

Si consideri, ora, l'esempio di figura 3.2 che illustra una situazione di ambiguità. Il robot percorre la traiettoria che lo porta in fondo ad uno dei corridoio che, agli occhi dei suoi sensori, appaiono molto simili. L'unica eccezione è rappresentata da un restringimento che viene notato solo al passaggio del robot nella sua prossimità. L'esempio è banale, ma illustra con chiarezza una tipica situazione in cui una con-

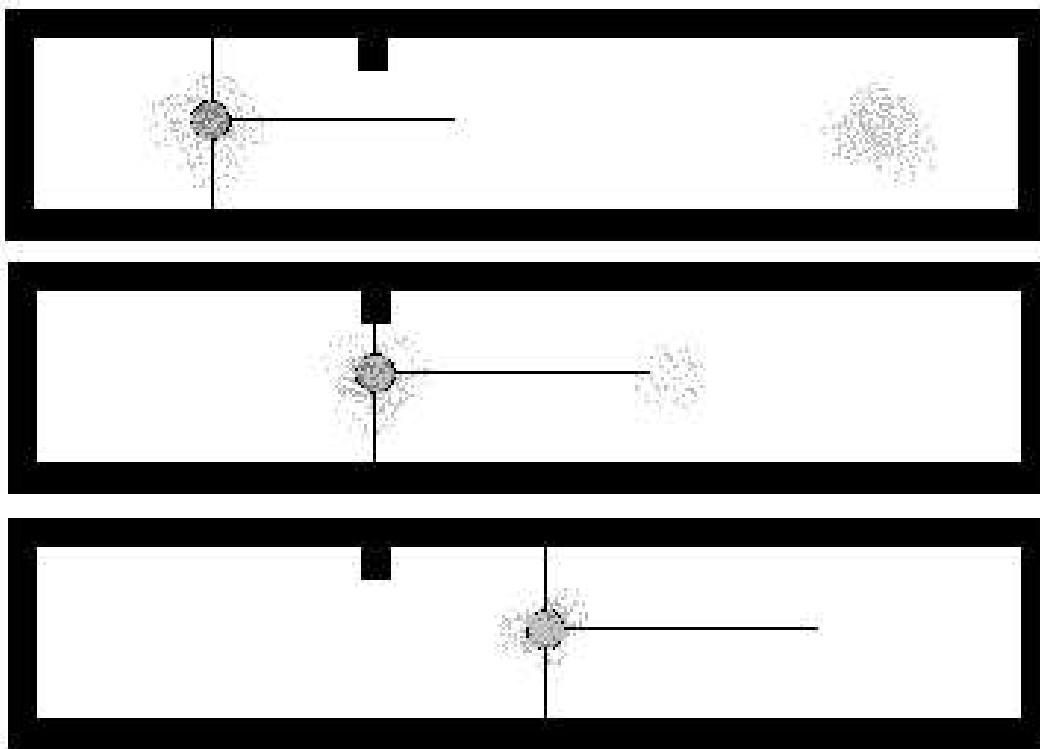


Figura 3.1: Il robot percepisce l'ambiente solo attraverso tre fasci laser. Con gli elementi percepiti si formano due ipotesi prevalenti. La situazione di ambiguità permane fin no al rilevamento di una strozzatura nel corridoio.

vergenza prematura rispetto all'acquisizione di nuovi dati sensoriali, avrebbe potuto portare ad una ipotesi errata e potenzialmente irrecuperabile.

3.2 Introduzione al clustering

L'analisi dei dati alla ricerca di cluster è lo studio formale di algoritmi e metodi per il raggruppamento o classificazioni di oggetti secondo un certo insieme di criteri. Questa disciplina affonda le sue radici nelle scienze statistiche ma è stata evoluta anche da altri settori in quanto si è dimostrata utile in un elevato numero di applicazioni disparate: ricerca di pattern in spazi multidimensionali, catalogazione di oggetti, apprendimento automatico, *data mining*, analisi finanziaria e, anche, l'elaborazione di immagini per la visione artificiale.

Occorre precisare che la ricerca di cluster è ben diversa da una normale cataloga-

zione in quanto non si conosce a priori il numero di classi in cui i dati verranno divisi e nemmeno l'esatta tipologia a cui tali classi finiranno con l'appartenere. In una semplice catalogazione, invece, si conosce a priori l'oggetto della ricerca che quindi risulta più semplice da svolgere e consente un approccio univoco senza imporre la necessità di fissare ipotesi che necessitano di una verifica a posteriori. In molti algoritmi di clustering, ad esempio, è necessario scegliere a priori in quanti cluster sarà diviso il set di dati e in che punti saranno centrati salvo, poi, spostarli nel corso di un processo iterativo.

Solitamente la ricerca di cluster viene inclusa nell'insieme delle classificazioni *Unsupervised* e cioè nella categoria dei processi di classificazione che non si appoggiano su alcuna informazione riguardante la natura dei dati e il risultato atteso. In contrapposizione all'*unsupervised* vi è la classificazione *supervised* che, invece, dispone a priori delle informazioni necessarie ad una suddivisione univoca del set dei dati e per una valutazione del risultato finale di tale operazione.

Prima di proseguire è bene indicare che il termine inglese *clustering* verrà spesso utilizzato per indicare il processo di ricerca dei cluster.

In [12] sono riportate alcune definizioni usate per i cluster:

- Un cluster è un insieme di entità affini fra di loro e dissimili da quelle che del cluster non fanno parte.
- Un cluster è un'aggregazione di punti nello spazio tali che la distanza fra due qualunque di loro sia inferiore alla distanza che separa un qualunque punto del cluster da un qualunque punto che non vi appartiene.
- Un cluster può essere descritto come una regione dello spazio n -dimensionale caratterizzata da una densità di particelle relativamente elevata e separata dagli altri cluster da zone di spazio contenenti una densità di punti relativamente ridotta.

Le ultime due definizioni assumono che gli oggetti siano rappresentabili come punti in uno spazio in cui sia definibile una distanza.

Il riconoscimento di un cluster è influenzato dal contesto e quindi non è univoco e rende difficile definire un processo ottimo con cui operare. Perfino il riconoscimento operato da un umano può essere affetto da ambiguità: in alcuni casi possono giocare

un ruolo fondamentale i fattori di forma, la definizione di distanza o la risoluzione dello spazio se questo è discreto. Molto dipende anche da ciò che si vuole o ci si aspetta di ottenere. In altre parole il risultato del clustering può, in alcuni casi, prestarsi a differenti interpretazioni.

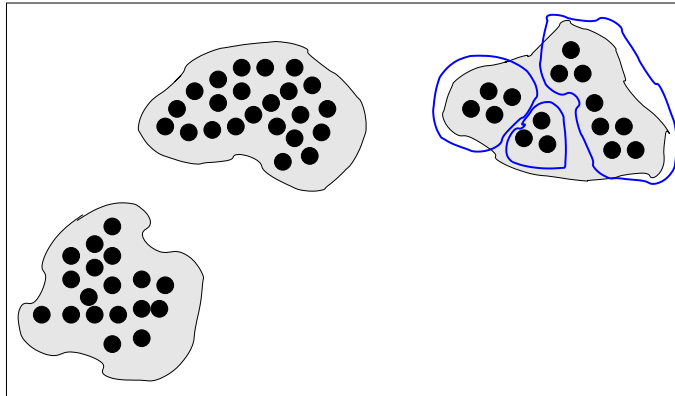


Figura 3.2: Esempio di clustering nello spazio 2D. In blu è rappresentata un clustering alternativo per uno dei gruppi di particelle che potrebbe essere ottenuto con una scelta diversa sulla distanza minima secondo la seconda definizione di cluster introdotta precedentemente.

3.3 Principali algoritmi di clustering

Come detto in precedenza non esiste un algoritmo che vada bene in ogni applicazione tuttavia esistono delle caratteristiche molto generali che ognuno dovrebbe possedere e che possono aiutare a svolgere una valutazione dei metodi. Ogni algoritmo di clustering dovrebbe essere tale che:

- la similarità delle entità intra-cluster sia molto elevata e inequivocabile;
- la similarità inter-classe deve essere molto bassa, cioè i cluster devono essere sempre perfettamente distinguibili gli uni dagli altri;
- la capacità di distinguere pattern nascosti nell'insieme di dati.
- il criterio utilizzato per la discriminazione è bene che sia piuttosto semplice e tale da evitare il più possibile le incertezze;

- il costo computazionale deve essere il minore possibile;
- sia scalabile e cioè che le dimensioni del set di dati da elaborare non influisca sul successo della ricerca;
- infine, la rappresentazione dell'output deve essere adeguata al contesto in cui opera l'algoritmo.

I vari approcci con i quali sono stati realizzati gli algoritmi più diffusi sono solitamente catalogati in cinque classi:

1. **Algoritmi di partizionamento.** Si parte con il costruire alcune partizioni da valutare in un secondo momento. Solitamente hanno una struttura iterativa che raffina la soluzione per passi successivi.
2. **Algoritmi Gerarchici.** Costruiscono una gerarchia fra set di dati. A loro volta si dividono in *agglomerativi* e *divisivi*.
3. **Algoritmi basati su densità.** Sfruttano funzioni di densità e connettività.
4. **Algoritmi a griglia.** Organizzano i dati in strutture a più livelli di diversa granularità sulle quali operano la clusterizzazione.
5. **Algoritmi basati su modelli.** Utilizzano la formulazione di ipotesi sulla clusterizzazione. In particolare viene predetta la struttura dei singoli cluster secondo modelli probabilistici.

Dato che la maggior parte degli algoritmi che verranno presentati sfruttano delle *distanze* è bene definire che la metrica alla quale si farà riferimento a meno che non venga specificato diversamente, sarà quella *euclidea*, caso particolare di *Minkowski* (3.1) con $p = 2$.

$$d_p(x_i, x_j) = \left(\sum_{k=1}^d |x_{i_k} - x_{j_k}|^p \right)^{\frac{1}{p}} \quad (3.1)$$

3.3.1 Algoritmi di Partizionamento

Gli algoritmi di partizionamento si basano sulla costruzione di partizioni in un database D di n oggetti in un set di k cluster. Il numero di cluster finale k è fissato a priori. I più noti algoritmi di questa classe sono *k-means* e *k-medoids*.

k-means

k-means è uno dei più semplici e utilizzati algoritmi *unsupervised* e produce cluster che si adattano particolarmente bene alla seconda definizione data in 3.2, quindi un presupposto per la sua applicazione è quello di avere elementi in uno spazio nel quale sia possibile definire una distanza. Come tutti gli algoritmi di partizionamento richiede che il numero di cluster in cui l'insieme di dati D verrà separato, sia fissato a priori. Sia k tale numero.

Il primo passo è quello di scegliere i k baricentri (o medie) dei cluster. Non esiste un metodo univoco e di solito, dal momento che non si ha alcuna informazione che dia qualche anticipazione del risultato finale, la scelta iniziale viene svolta in modo aleatorio cercando di spargere i punti il più possibile. Il distanziamento dei baricentri in fase di piazzamento iniziale non è necessario alla convergenza (che è sempre garantita), ma la pratica insegna che può velocizzarla. Dopo il piazzamento dei baricentri si formano i cluster secondo un criterio di distanza: l'elemento j -esimo verrà aggregato al cluster al cui baricentro è più vicino. Al termine di questa operazione, viene calcolato il nuovo baricentro dei vari cluster e viene ripetuta l'operazione di assegnazione. Il ciclo si ripete fino a quando l'aggiornamento dei baricentri non produce cambiamenti significativi rispetto all'iterazione precedente. Infine viene svolta una valutazione della clusterizzazione ottenuta che risulta tanto migliore quanto minore è il valore della *funzione obiettivo*. Normalmente la funzione obiettivo che si utilizza è la *SSE (Sum of the Squared Error)*:

$$J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - m_j\|^2 \quad (3.2)$$

L'uso della funzione obiettivo non è il solo modo per valutare la buona riuscita del clustering ma è uno dei più utilizzati.

In verità, benché la convergenza di k-means sia garantita, non è garantito che la clusterizzazione sia la migliore in assoluto, cioè che la funzione obiettivo abbia raggiunto un minimo globale per D e k . Dal momento che il risultato finale dipende dalla disposizione iniziale dei k baricentri, per ottimizzare il risultato si potrebbe ripetere l'intero algoritmo fino al raggiungimento di un risultato soddisfacente.

Il costo computazionale di k-means è dell'ordine di $O(NkI)$, dove N è il numero di elementi in D , k è, come sappiamo, il numero di cluster e I è il numero massimo di ripetizioni dell'algoritmo stabilito per la ricerca del minimo globale di J . L'algoritmo, ottimizzazione inclusa, è riassunto in "algoritmo 3".

Algoritmo 3 Algoritmo K-means

Require: D, N, k, I

```
1:  $J_{mix} = \infty; C_{set_{opt}}$ 
2: while #iter < I or il partizionamento non è ottimo do
3:   scegliere  $m_1, m_2, \dots, m_k$  baricentri (o medie)
4:   repeat
5:     for  $i = 0$  to  $N$  do
6:       cerca l' $m_j$  più vicino a  $x_i$  e assegna l'elemento al cluster  $C_j$ .
7:     end for
8:     for  $j = 0 \dots k$  do
9:       ricalcola  $m_j$ 
10:      aggiungi  $C_j$  a  $C_{set}$ 
11:    end for
12:    until i baricentri ricalcolati sono diversi dai precedenti
13:    calcola la funzione obiettivo  $J$ 
14:    if ( $J < J_{min}$ ) then
15:       $J_{min} = J$ 
16:       $C_{set_{opt}} = C_{set}$ 
17:    end if
18:  end while
```

k-medoids

La peculiarità di questo algoritmo è l'idea che i cluster possano essere rappresentati da un oggetto particolarmente caratteristico fra quelli che contengono: il *medoid*. Il fatto di non utilizzare un approccio geometrico come *k-means*, consente l'utilizzo di *k-medoids* anche in casi in cui non è definibile una media; è infatti,

necessaria solo la distanza. L'algoritmo, noto anche come PAM (*Partition Around Medoids*), parte con l'assegnazione arbitraria del ruolo di *medoid* a k elementi dell'insieme di dati D . k è uno degli ingressi dell'algoritmo e deve essere fissato a priori. La dinamica dello svolgimento è piuttosto simile a quella vista per *k-means*: ancora una volta un algoritmo iterativo nel quale ad ogni passo si cerca un rimpiazzo per i *medoids* esistenti che sia in grado di migliorare la qualità del clustering (figura 3.3). Anche per *k-medoids* è necessario ripetere il clustering più volte perchè la bontà della soluzione trovata è influenzata dalla scelta iniziale dei *medoids*. Il criterio di valutazione può essere di nuovo fatto sulla base della funzione obiettivo 3.2 definita in precedenza, con la differenza che la distanza viene calcolata utilizzando il medoid invece del baricentro (media).

Il principale punto debole di *k-medoids* è che funziona bene solo su insiemi piuttosto ristretti di dati in quanto è poco scalabile.

L'algoritmo 4 mostra il procedimento.

Algoritmo 4 Algoritmo K-medoids

Require: D, N, k .

```
1:  $J_{mix} = \infty$ 
2: scegliere  $o_1, o_2, \dots, o_k$  medoids
3: for  $i = 0$  to  $N$  do
4:   cerca l' $o_j$  più vicino a  $x_i$  e assegna l'elemento al cluster  $C_j$ .
5: end for
6: repeat
7:   sostituisci un medoid a caso con un non-medoid
8:   for  $i = 0$  to  $N$  do
9:     cerca l' $o_j$  più vicino a  $x_i$  e assegna l'elemento al cluster  $C_j$ .
10:  end for
11:  if c'è stato almeno un riassegnamento then
12:    calcola  $J = SSE$ 
13:    if ( $J < J_{min}$ ) then
14:       $J_{min} = J$ 
15:      mantieni la sostituzione
16:    else
17:      annulla la sostituzione
18:    end if
19:  end if
20: until da  $I$  iterazioni nessuno scambio ha migliorato SSE
```

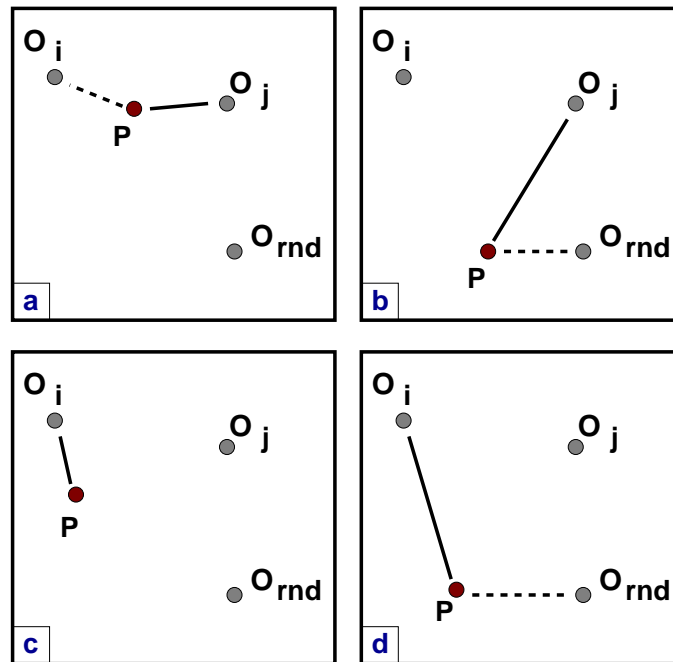


Figura 3.3: In seguito allo scambio fra di ruolo fra un medoid O_j e uno O_{rnd} : **a)** P , originariamente assegnato a O_j passa a O_i ; **b)** da O_j al nuovo medoid O_{rnd} che lo ha sostituito; **c)** non avviene alcuna riassegnazione; **d)** il punto P , assegnato ad un medoid estraneo allo scambio passa a O_{rnd} .

3.3.2 Algoritmi Gerarchici

Gli algoritmi gerarchici utilizzano una *matrice delle distanze* per definire i cluster. L'aspetto particolarmente interessante è che non richiedono di fissare il numero di cluster k a priori anche se è necessario stabilire una condizione di terminazione del ciclo di iterazioni. Esistono due tipologie di algoritmi gerarchici: quelli *agglomerativi* e quelli *divisivi*. Il loro approccio è opposto poiché il primo ha come punto di partenza l'insieme dei singoli elementi che vengono, via via, aggregati fino ad ottenere un cluster che li include tutti (*bottom-up*), mentre il secondo opera in modo contrario (*top-down*) (figura 3.4). Il principale difetto di questi algoritmi è la scarsa scalabilità dovuta al costo computazionale, che è dell'ordine di $O(n^2)$. Inoltre le decisioni relative alla divisione o aggregazione sono definitive e hanno un grosso impatto sul risultato finale. In varie implementazioni e versioni, sono stati riscontrati tutti o alcuni di questi problemi:

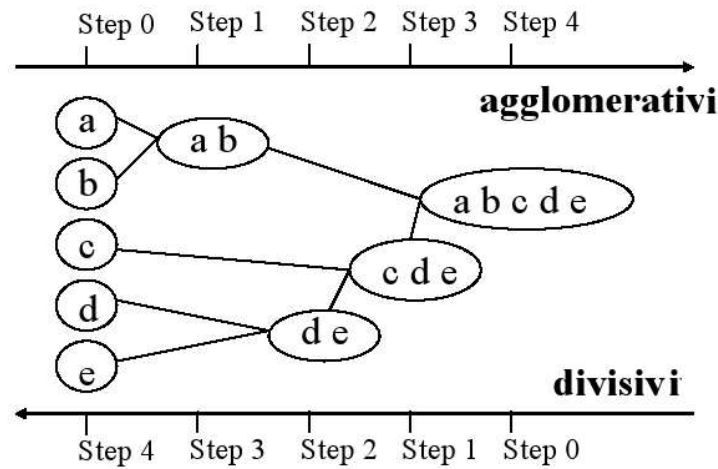


Figura 3.4: Rappresentazione grafica degli approcci *agglomerativo* e *divisivo*.

- sensibilità al rumore e agli outliers;
- difficoltà di gestire cluster di dimensioni differenti e forme non convesse;
- frammentazione dei grandi cluster.

Algoritmi Gerarchici Agglomerativi

Dato uno spazio di dati D contenente N elementi elemento sono possibili N^2 confronti, intesi come distanze fra gli elementi, che si possono organizzare in una matrice $N \times N$. Nel contesto degli algoritmi gerarchici il criterio di similarità è costituito dalla distanza: quanto è minore tanto maggiore è l'affinità. Il primo passo della procedura prevede di assegnare tutti gli elementi ad un cluster. Se le entità sono N allora N saranno i cluster, ciascuno con un solo elemento. In questa situazione la distanza fra i cluster è la stessa di quella fra gli oggetti che contengono.

A questo punto viene individuata la coppia a più elevata similarità per fonderla in uno nuovo cluster. Segue l'aggiornamento della matrice delle distanze. Il procedimento viene ripetuto fino alla formazione di un singolo cluster di cardinalità N .

Naturalmente non serve avere un singolo cluster quando ciò che si desidera è una separazione in cluster di un insieme di dati. È quindi necessario decidere una soglia di similarità superata la quale si smette di congiungere i cluster. In alternativa si può

decidere quanti cluster si desidera avere e mantenere solo i k più simili. Il calcolo della distanza fra due cluster può essere svolto con diversi approcci.

single-linkage. La distanza fra due cluster equivale al minimo delle distanze che separano i suoi elementi.

$$d(C_k, C_h) = \min_{x_i \in C_k, x_j \in C_h} d(x_i, x_j)$$

complete-linkage. La distanza fra due cluster equivale al massimo delle distanze che separano i suoi elementi.

$$d(C_k, C_h) = \max_{x_i \in C_k, x_j \in C_h} d(x_i, x_j)$$

average-linkage. La distanza fra due cluster equivale alla distanza media che separa i loro elementi.

$$d(C_k, C_h) = \text{mean}_{x_i \in C_k, x_j \in C_h} d(x_i, x_j)$$

Una variante (UCLUS) dell' *average-linkage* prevede l'uso della mediana delle distanze che rende l'algoritmo molto più resistente agli *outlier*¹. Si tenga presente che l'utilizzo di ognuno di questi approcci porta ad un differente risultato, al punto che molti autori applicano una ulteriore classificazione agli algoritmi gerarchici basata su questi metodi di misurare la distanza fra cluster.

ESEMPIO con single-linkage. Sia D un insieme di oggetti: O_1, O_2, O_3, O_4, O_5 e O_6 . Si organizzino le loro distanze in una matrice dove l'elemento a_{ij} rappresenta la distanza fra O_i e O_j .

$$\mathbf{M}_0 = \begin{pmatrix} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 \\ 0 & 63 & 80 & 25 & 43 & 100 \\ 63 & 0 & 30 & 47 & 27 & 40 \\ 80 & 30 & 0 & 75 & 55 & 10 \\ 25 & 47 & 75 & 0 & 20 & 87 \\ 43 & 27 & 55 & 20 & 0 & 67 \\ 100 & 40 & 10 & 87 & 67 & 0 \end{pmatrix}$$

¹**outlier** è un termine che deriva dalla statistica e indica osservazioni molto distanti da quelle attese dalla distribuzione di probabilità.

All'inizio i cluster sono sei e sono tutti a livello 0 ($L = 0$). Si cerca la coppia di cluster più simile e si scopre che è quella formata da C_3 e C_6 . Si eliminano le righe e le colonne corrispondenti dalla matrice e si aggiunge quella del cluster fuso $C_{3,6}$ al quale viene assegnato $L_{3,6} = dist(3, 6) = 10$.

Si calcolano le distanze nel nuovo assetto e la nuova matrice è:

$$\mathbf{M}_1 = \begin{pmatrix} C_1 & C_2 & C_{3,6} & C_4 & C_5 \\ 0 & 63 & 80 & 25 & 43 \\ 63 & 0 & 30 & 47 & 27 \\ 80 & 30 & 0 & 75 & 55 \\ 25 & 47 & 75 & 0 & 20 \\ 43 & 27 & 55 & 20 & 0 \end{pmatrix}$$

L'operazione viene ripetuta e i cluster più vicini sono C_4 e C_5 .

$L_{4,5} = 20$. Dopo il calcolo delle nuove distanze, la matrice diviene:

$$\mathbf{M}_2 = \begin{pmatrix} C_1 & C_2 & C_{3,6} & C_{4,5} \\ 0 & 63 & 80 & 25 \\ 63 & 0 & 30 & 27 \\ 80 & 30 & 0 & 55 \\ 25 & 27 & 55 & 0 \end{pmatrix}$$

Ora i più simili sono $C_{3,6}$ e C_1 , la loro distanza è 25. $L_{1,3,6} = 25$. La successiva matrice è:

$$\mathbf{M}_3 = \begin{pmatrix} C_{1,3,6} & C_2 & C_{4,5} \\ 0 & 27 & 55 \\ 27 & 0 & 30 \\ 55 & 30 & 0 \end{pmatrix}$$

Si ottiene $C_{1,2,3,6}$ con $L_{1,2,3,6} = 27$.

$$\mathbf{M}_4 = \begin{pmatrix} C_{1,2,3,6} & C_{4,5} \\ 0 & 30 \\ 30 & 0 \end{pmatrix}$$

Infine il cluster unico viene raggiunto con un livello pari a 30. In figura 3.5 una

rappresentazione schematica della gerarchia dei cluster ottenuti.

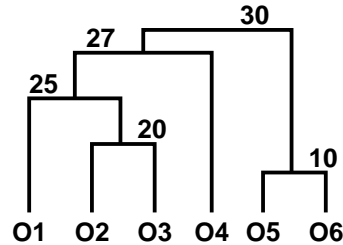


Figura 3.5: Rappresentazione grafica degli approcci *agglomerativo* e *divisivo*. Se la similarità fosse fissata a 10, i cluster sarebbero 5, se fosse fissata a 20 sarebbero 4, e così via.

Algoritmo 5 Algoritmo Gerarchico Aggregativo

Require: D, N //numero di elementi di D

- 1: Creazione di N cluster da un solo elemento, con $L(0) = 0$ e numero di sequenza $m = 0$;
 - 2: Creazione della matrice delle distanze
 - 3: **repeat**
 - 4: trovare i cluster più simili e fonderli
 - 5: $m := m + 1$
 - 6: impostare il livello del cluster appena creato al valore della distanza che separava i cluster dai quali è stato generato.
 - 7: aggiornamento della matrice delle distanze eliminando le righe e le colonne corrispondenti ai cluster fusi e aggiungendo una riga e una colonna per quello nuovo con distanze aggiornate.
 - 8: **until** \exists un solo cluster di N elementi
-

Algoritmi Gerarchici Divisivi

Gli algoritmi divisivi seguono una procedura esattamente opposta a quella degli aggregativi: partono da un unico cluster per arrivare a cluster formati da un solo elemento.

3.3.3 Algoritmi basati su funzioni di densità

Si basa su concetti di connettività fra punti secondo densità. La densità viene definita secondo due parametri [13]:

Eps: massimo raggio di vicinato da un punto $x \in D$.

MinPts: minimo numero di punti entro Eps .

Con $N_{Eps}(P)$ si indica l'insieme dei punti entro il raggio Eps da P :

$$N_{Eps}(P) := \{q \in D \mid dist(P, q) < Eps\}$$

La principale algoritmo basato su densità è DBSCAN (Density Based Spatial Clustering of Application with Noise). Per poter descrivere adeguatamente l'algoritmo, sono necessarie alcune definizioni:

- $P \in D$ si dice **raggiungibile direttamente secondo densità** da $q \in D$ se:
 - 1) $P \in N_{Eps}(q)$
 - 2) $q \mid |N_{Eps}(q)| \geq MinPts$. Si può anche dire che q è un *core point* per N_{Eps} . vice versa si definisce *border point*.
- $P \in D$ si dice **raggiungibile secondo densità** da $q \in D$ se \exists una catena di punti P_1, P_2, \dots, P_n , con $P_1 = P$ e $P_n = q$ tali che P_{i+1} è direttamente raggiungibile secondo densità da $P_i \forall i$
- un punto $P \in D$ si dice **densamente connesso** a $q \in D$ se \exists un punto $o \in D$ tale che sia P che q siano raggiungibili secondo densità da quest'ultimo.

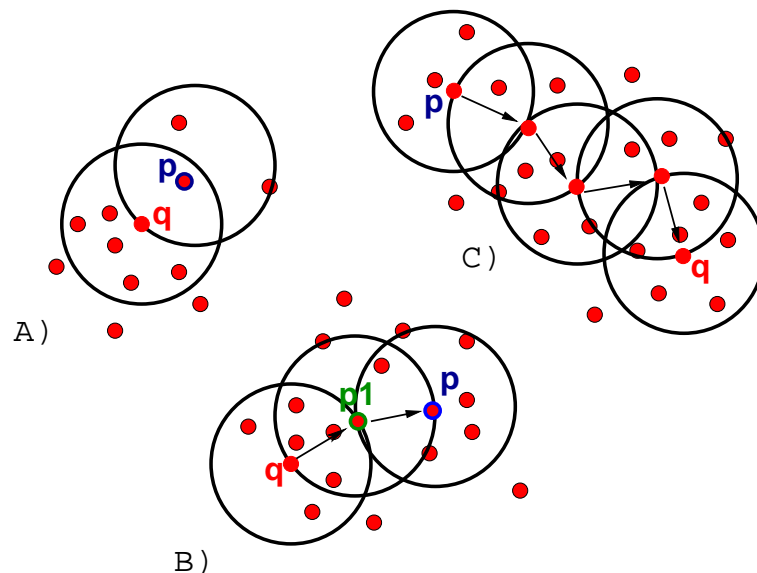


Figura 3.6: a) raggiungibilità diretta, b) raggiungibilità, c) punti densamente connessi.

DBSCAN [13] richiede come ingressi l'insieme dei dati D , Eps e $MinPts$. Un punto P arbitrario viene selezionato e vengono individuati tutti gli elementi da esso raggiungibili secondo densità. Se il numero di elementi trovati supera $MinPts$, cioè P è un *core*, allora forma un cluster. Viceversa, se è *border point* si passa direttamente al punto successivo. Il ciclo termina quando tutti i punti sono stati esaminati. Segue, poi, una fase nella quale i cluster vengono fusi con quelli con i quali hanno una intersezione. Il metodo di ricerca delle intersezioni è piuttosto semplice: si calcola il baricentro (centroid) di ogni cluster, poi si fondono tutti cluster i cui baricentri distano meno di $2Eps$. Il costo computazionale complessivo è $N \log N$, con N numero degli elementi.

Uno schema riepilogativo di questa descrizione può essere consultato in "algoritmo 6".

Algoritmo 6 DBSCAN

Require: $D, Eps, MinPts$

```
1: repeat
2:    $C_s = \emptyset$  /* cluster set */
3:   Scelta di un  $P \in D$  fra quelli non ancora esaminati
4:   marca  $P$  come "visitato"
5:    $j =$  punti raggiungibili da  $P$  entro il raggio  $Eps$ 
6:   if  $j \geq MinPts$  then
7:      $P$  e i punti da lui raggiungibili formano un nuovo cluster  $C_{new}$ 
8:     aggiunta di  $C_{new}$  al cluster set  $C_s$ 
9:   end if
10: until tutti gli elementi di  $D$  sono stati esaminati
11: for all  $C_j \in C_s$  do
12:   calcola il centroide  $c_j$  di  $C_j$ 
13: end for
14: for all  $C_i, C_k \in C_s$  con  $i \neq k$  do
15:   if  $dist(c_i, c_k) \leq 2Eps$  then
16:      $C_i = C_i \cup C_k$ 
17:     calcola il nuovo centroide di  $C_i$ 
18:   end if
19: end for
```

I punti di forza di DBSCAN sono senza dubbio la capacità di individuare cluster di forma e dimensioni arbitrarie e di richiedere una sola scansione dei dati. Inoltre

presenta la capacità di individuare elementi di rumore ad outlier fra i dati, caratteristica che si può rivelare molto utile per individuare anomalie o per eseguire filtri dei dati in applicazione come la visione artificiale (3.7).

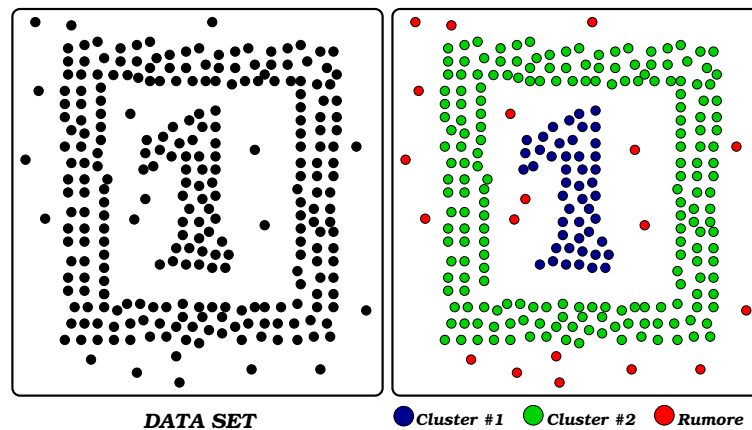


Figura 3.7: Set di dati prima e dopo DBSCAN. Si noti la presenza di punti che non appartengono a nessun cluster: il rumore

3.3.4 Algoritmi basati su Griglia

Si basano sulla suddivisione dello spazio in celle iper-rettangolari la cui dimensione viene scelta in relazione alla risoluzione desiderata. La complessità del algoritmo dipende dal numero di celle della griglia che sono occupate da almeno un elemento e non dal numero di elementi.

Dopo aver diviso lo spazio, ogni elemento viene assegnato ad una cella.

Una volta completata l'assegnazione, per ognuna delle celle viene calcolata la densità di dati. A questo punto può essere applicato un filtro a soglia che elimina le partizioni che non raggiungono un certo valore di densità (δ).

Il cluster vengono formati sulla base delle adiacenze.

Le principali implementazioni di questo algoritmo sono *STING*[14], *CLIQUE*[13] e *waveCluster*[14].

3.3.5 Algoritmi basati su Modelli

Gli algoritmi basati su modelli consistono nella ricerca del migliore adattamento (*best fit*) dei parametri di un modello che è stato scelto a priori per il cluster, sulla base dei dati a disposizione. Il modello di un cluster è rappresentato matematicamente da una funzione di distribuzione. Ad esempio, una gaussiana in un dominio continuo e una distribuzione di poisson in uno discreto, sono le scelte più frequenti da parte di chi implementa algoritmi di questo tipo. Il risultato prodotto è un *mix* di distribuzioni ciascuna delle quali individualmente rappresenta un cluster. Solitamente il termine utilizzato per indicare una distribuzione individuale è *distribuzione componente* o semplicemente *componente*.

I principali vantaggi di questa tecnica sono: la possibilità di utilizzare le numerose tecniche statistiche disponibili, la flessibilità data dalla possibilità di scegliere distribuzioni ad *hoc*, la stima della densità di particelle e la scelta di una divisione in cluster *soft*.

Mix di Guassiane

In questo algoritmo ogni cluster viene rappresentato da una distribuzione gaussiana e l'intero set di dati dopo il clustering da una distribuzione complessiva che è un mix di quelle individuali che rappresentano i cluster 3.8.

Sia $P(\omega_i)$ la probabilità che venga scelto un cluster fra tutti. Si supponga di avere stabilito la necessità di k cluster: $P(\omega_1), P(\omega_2), \dots, P(\omega_k)$.

La distribuzione di probabilità associata ad un singolo elemento di trovarsi nel cluster i - *esimo* è data da:

$$f_i(x | \mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(x - \mu)^2}{2\sigma^2} \right]$$

Conseguentemente la distribuzione composta, assume la seguente forma:

$$f(x | \mu_{1\dots k}, \sigma_{1\dots k}) = \sum_{i=1}^k P(\omega_i) f_i(x | \mu_i, \sigma_i) \quad (3.3)$$

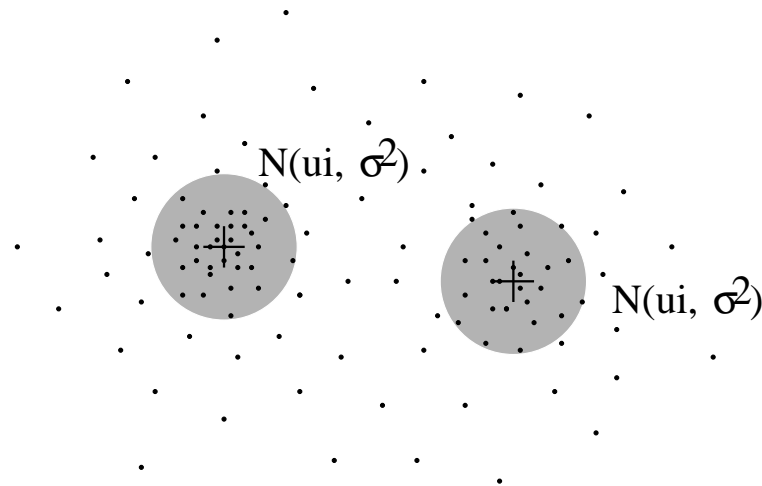


Figura 3.8: Due cluster. L'area grigia ha raggio σ .

Segue immediatamente che:

$$f(\text{data} \mid \mu_{1\dots k}, \sigma_{1\dots k}) = \prod_{i=1}^N f_i(x \mid \mu_i, \sigma_i) \quad (3.4)$$

L'obiettivo è quello di massimizzare la probabilità $P(x \mid \mu_1, \dots, \mu_k)$, cioè la probabilità composta, associata al singolo campione dati i baricentri delle k gaussiane. La distribuzione composta massimizzata tende a formare picchi in corrispondenza del centro delle componenti in quanto i dati dovrebbero essere più densi nelle zone delimitate dal cluster. Inoltre espone una copertura dei dati molto efficiente e quindi rende possibile la scoperta della maggior parte (o di tutti) i pattern.

La massimizzazione è un problema piuttosto complicato da risolvere per via analitica, quindi si ricorre a metodologie che consentono di ricavare una soluzione approssimata come, ad esempio, EM (*Estimation-Maximization*).

Per la precisione EM non identifica un metodo in particolare, ma bensì una classe di metodi, caratterizzata da una fase predittiva ed una che ricerca il massimo di una funzione obiettivo in relazione ad alcuni parametri. Quella presentata in algoritmo 7 [15] è un'implementazione di EM che può essere sfruttata per il mix di gaussiane

Algoritmo 7 EM

Require: /* assegna i centri e le probabilità iniziali */

$$\mu_1^{(0)}, \mu_2^{(0)}, \dots, \mu_k^{(0)}, p_1^{(0)}, p_2^{(0)}, \dots, p_k^{(0)}.$$

1: **repeat**

2: /* E-step */

$$\frac{P(\omega_j | x_k, \lambda_t) P(\omega_j | \lambda_t)}{P(x_k | \lambda_t)} = \frac{P(x_k | \omega_i, \mu_i^{(t)}, \sigma^2) P_i^{(t)}}{\sum_k P(x_k | \omega_j, \mu_j^{(t)}, \sigma^2) P_j^{(t)}}$$

3: /* M-step */

$$\mu_i^{(t+1)} = \frac{\sum_k P(\omega_i | x_k, \lambda_t) x_k}{\sum_k P(\omega_i | x_k, \lambda_t)}$$

$$P_i^{(t+1)} = \frac{\sum_k P(\omega_i | x_k, \lambda_t)}{N} \text{ /* N numero dei dati in D */}$$

$$\sigma_i^{(t+1)} = \frac{\sum_{j=1}^N P_{j,i} (x_j - \mu_i^{(t+1)})^2}{\sum_{i=1}^N P_{j,i}}$$

4: **until** convergenza

3.4 L'algoritmo realizzato

L'algoritmo che è stato studiato per essere applicato al localizzatore ricade nella classe di quelli basati su griglia e trae alcuni spunti da quello implementato nel localizzatore del framwork *Player-Stage*[16] per i robot e mette a disposizione una libreria per lo sviluppo di applicazioni di controllo. In *player* i punti vengono memorizzati in un *kd-tree* dopo essere stati discretizzati. L'insieme dei dati qui ntizzati forma una griglia sui cui nodi vengono a trovarsi tutti i punti. Segua la divisione in cluster secondo un criterio di adiacenza: se due nodi che contengono almeno un punto sono contigui vengono etichettati come appartenenti allo stesso cluster. Inoltre possono essere formati cluster con un solo elemento, cioè non viene applicato alcun fi ltraggio ai dati isolati. La ricerca delle adiacenze segue un tecnica ricorsiva: viene scelto un elemento a caso fra quelli che non sono stati mai esaminati; si cercano i suoi vicini, se ne ha; per ognuno di quelli non precedentemente esaminati si ripete la ricerca delle adiacenza con la stessa modalità.

La costruzione dei cluster avviene durante la ricerca dei vicini attraverso un sistema di etichette. Il risultato è un *kd-tree* nel quale ciascuna foglia possiede un'attributo che indica il cluster di appartenenza. Solo successivamente le statistiche di ogni

cluster vengono calcolate e memorizzate in strutture dati.

L'algoritmo studiato per il localizzatore ha in comune con quello di *Player* la costruzione dei cluster secondo un criterio di adiacenza e di richiedere, come requisito, il solo set di dati senza necessità di sapere il numero di cluster k a priori. Come ormai è chiaro, il contesto di applicazione di un algoritmo di clustering pone dei vincoli alla scelta o progettazione dello stesso. Ad esempio, il fatto di non sia necessario decidere a priori il numero di cluster in cui il set di dati deve essere diviso è un requisito fondamentale per un localizzatore particellare. Infatti il valore k rappresenta altrettante ipotesi di localizzazione e deve essere un *output* dell'algoritmo piuttosto che un *input*. L'algoritmo opera al momento della generazione dei cam-

Algoritmo 8 Algoritmo di clustering implementato

Require: $C_s; data; s_{new}; GridRes; map$

- 1: $x = discretizza(s_{new}, GridRes)$
- 2: $vicini = trova_vicini(data, x)$
- 3: **if** non ci sono vicini **then**
- 4: crea nuovo Cluster c_{new}
- 5: inserisci s_{new} in c_{new}
- 6: assegna un'etichetta $label_{new}$ a c_{new}
- 7: $C_s := C_s + c_{new}$
- 8: aggiungi a map la riga $label_{new} \mid label_{new}$
- 9: **else if** c'è un solo vicino **then**
- 10: aggiungi s_{new} al cluster a cui appartiene il vicino
- 11: **else**
- 12: $refLabel =$ label del cluster che contiene $vicini[0]$
- 13: **for** $i = 1$ to $(n^\circ vicini - 1)$ **do**
- 14: $l :=$ label del cluster che contiene $vicini[i]$
- 15: $tmpLabel := consulta_corrispondenze(map, l)$
- 16: **if** $tmpLabel \neq refLabel$ **then**
- 17: $C_{ref} := C_{refLabel} \cup C_{tmpLabel}$
- 18: elimina da map la riga $tmpLabel \mid tmpLabel$
- 19: aggiungi a map la riga $tmpLabel \mid refLabel$
- 20: **end if**
- 21: **end for**
- 22: **end if**

pioni e prima del ricampionamento (Algoritmo 2 al paragrafo 2.4.3). Mano a mano che i campioni vengono generati, l'algoritmo di clustering si occupa di elaborarli.

L'algoritmo 8 mostra in modo dettagliato la serie di operazioni che vengono svolte ogni qual volta viene generato un nuovo campione, il quale, come si può notare, fa gura nei requisiti col simbolo s_{new} . Gli altri ingressi rappresentano il set di campioni generati (*data*) fino all'iterazione corrente e il set di cluster in cui sono divisi (C_s); la risoluzione della griglia per ogni dimensione (*GridRes*) e una mappa di corrispondenza (*map*) (descritta in 3.4) che serve per il riconoscimento dei cluster di C_s .

La prima operazione svolta è una discretizzazione del campione (1) secondo la risoluzione della mappa in modo da poter sfruttare la griglia per individuare le adiacenze (2).

Dato l'insieme delle celle confinanti che contengono almeno un campione (*vicini*), si possono verificare tre casi:

1. *l'insieme dei vicini è vuoto*(3). In questo caso s_{new} è isolato. Viene creato un cluster con il solo campione. In questo caso nella mappa va aggiunta una riga che indichi la presenza di un nuovo cluster disgiunto da tutti gli altri e cioè chiave = valore = label del nuovo cluster.
2. *l'insieme dei vicini contiene un solo elemento*(9). Viene identificato il cluster di appartenenza del vicino utilizzando mappa e poi vi si aggiunge s_{new} .
3. *l'insieme dei vicini contiene più di un elemento*(11). In questo caso i vicini possono appartenere a più cluster o ad uno soltanto. Quello che accade è che s_{new} venga inglobato dal primo cluster che viene individuato e poi, se gli altri vicini appartengono a gruppi diversi, si procede all'unione dei cluster.

Mappa di corrispondenza

La mappa di corrispondenza è stata creata per annotare le associazioni fra le etichette (o *label*) dei cluster ottenuti dalle fusioni di cluster precedentemente etichettati in modo diverso. Per evitare di aggiornare le etichette di tutti gli elementi di un cluster e dei loro elementi ogni qual volta si verifica una fusione si inserisce nella mappa la coppia di etichette che da quel momento rappresentano il medesimo cluster. Ricapitolando quello che si nota nell'algoritmo 8, è che ogni qual volta viene

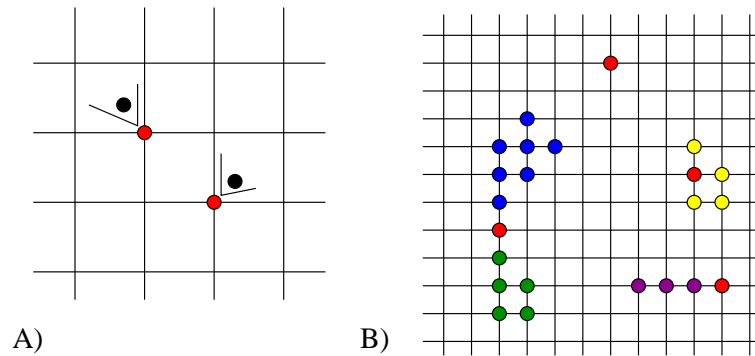


Figura 3.9: A) discretizzazione dei campioni nel caso 2D. B) I casi che si possono verificare con l'ingresso di una nuova particella (in rosso): particella isolata, adiacente ad una sola particella, adiacente a più particelle di un solo cluster, adiacente a più particelle di più cluster.

creato un nuovo cluster, una riga viene aggiunta alla mappa. Tale riga è una coppia <chiave, valore> in cui chiave = valore = label. La chiave contiene l'etichetta del primo cluster di partenza ed è la stessa con la quale sono contrassegnati i campioni. Il valore contiene l'etichetta del cluster finale di appartenenza. La clusterizzazione

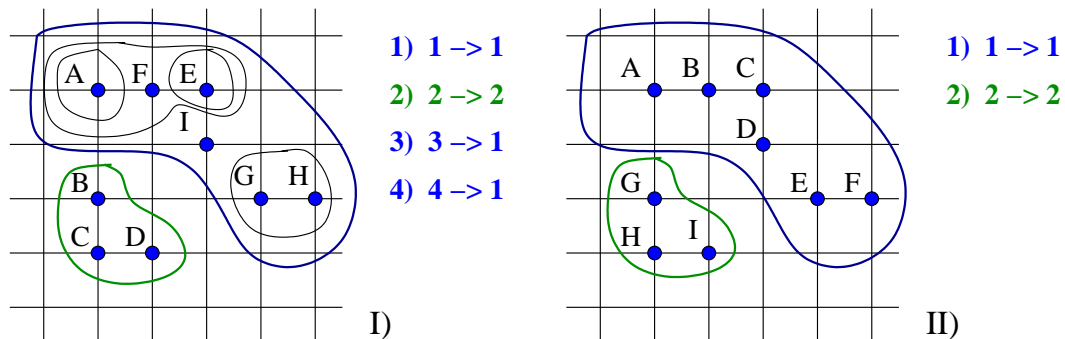


Figura 3.10: I) Mappa creata secondo l'ordine di inserimento dei punti da A a I. Si noti l'ordine in cui i cluster vengono aggregati. II) Stessi punti e, ovviamente, risultato finale. Cambiando l'ordine di arrivo dei punti, cambia anche la mappa ma non muta il risultato del clustering.

prevede che l'algoritmo 8 venga ripetuto per tutti gli N campioni. La struttura che è stata utilizzata per memorizzare i dati è un kd-tree, quindi la ricerca dei vicini ha un costo di $n \log n$ dove n è il numero di elementi memorizzati nell'albero. In conclusione il costo complessivo è minore di $O(N^2 \log(N))$ ed è approssimativa-

mente $\sum_i^N (i \log i)$ in quanto l'albero raggiunge gli N elementi progressivamente e non dalla prima iterazione.

Indubbiamente si tratta di un costo piuttosto elevato, ma questo è un male comune a tutti gli algoritmi di clustering. Se si osservano alcuni dei valori riportati nei paragrafi precedenti di questo capitolo, si nota immediatamente che i costi sono in generale più bassi. Attenzione però, in nessuno di quei casi si è fatta alcuna ipotesi sulla struttura dati utilizzata e sul costo di accesso ai suoi elementi. Alla luce di questa considerazione l'algoritmo implementato è da ritenersi di peso paragonabile alla maggior parte degli algoritmi che sarebbero stati applicabili al problema della localizzazione; in particolare quelli che non richiedono il numero di cluster come input.

Meglio potrebbe fare l'algoritmo presentato in 3.4.1 nell'ipotesi applicazione in ambiente multiprocessore.

3.4.1 Algoritmo di clustering con l'uso di una griglia a risoluzione variabile

Un'alternativa all'approccio utilizzato in questo lavoro di tesi (3.4) può essere quello di utilizzare una griglia a risoluzione variabile in modo simile a STING [14].

Lo spazio è diviso in celle rettangolari con un approccio top-down: le più grandi vengono suddivise a loro volta. Lo spazio risulta infine suddiviso in modo da presentare diversi livelli (layer) di risoluzione. Per ogni cella viene calcolata la densità delle particelle da usare come grandezza rappresentativa dei dati in quella regione. Questo approccio consente di svincolare la complessità dell'algoritmo dalla mole di dati, che diviene dipendente dalla sola risoluzione del livello più basso: $O(N_c)$, dove N_c è il numero di celle.

Anche la ricerca di cluster segue un approccio top-down. A partire da un layer predefinito fra quelli più in alto, si cercano le adiacenze fra le celle piene eliminando quelle ritenute ininfluenti (densità sotto soglia), poi si passa al layer immediatamente sottostante e si ripete il procedimento fino al raggiungimento di quello più in basso. In figura 3.11 è presentato un esempio di clustering basato su griglia. Un vantaggio tratto da STING è la possibilità di parallelizzazione: partendo dal layer più alto, ciascuna divisione può essere affidata ad un thread. Un secondo vantaggio

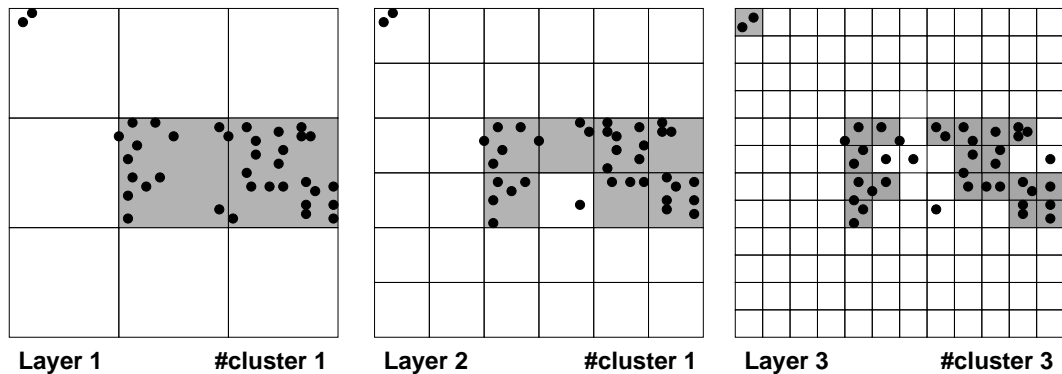


Figura 3.11: Esempio di clustering con tre layer. In particolare, si noti l'effetto del filtraggio che produce un cluster in più con i due punti in alto a sinistra. In questo esempio, la densità di particelle ai layer 1 e 2 con sole due particelle per cella è insufficiente a definire un cluster, mentre con la griglia del layer 3 avviene la formazione del cluster.

è quello della complessità che, come già illustrato, dipende dalla risoluzione della griglia. Infine, l'algoritmo non richiede il numero di cluster k a priori. Lo svantaggio è che i bordi dei cluster possono essere formati soltanto da segmenti verticali e orizzontali in quanto le celle sono definite da una griglia.

3.4.2 Implementazione dell'algoritmo di clustering

L'implementazione dell'algoritmo di clustering (3.4) ha richiesto di prestare attenzione alla struttura della libreria per la quale è stato concepito [11]. La libreria possiede la classe `LSOFT::RTSampleManager`, progettata per gestire l'aggiornamento dei campioni. Il metodo `LSOFT::RTSampleManager::evolve()` è il responsabile dell'update del set di campioni e della contestuale memorizzazione. Si è quindi pensato di realizzare una classe contenitore da sostituire a quella utilizzata in origine e capace di memorizzare i sample contestualmente all'applicazione dell'algoritmo di clustering (3.4).

La figura 3.12 mostra l'interfaccia di `LSOFT::ClusterContainer`, la classe preposta alla memorizzazione dei campioni e al clustering, e quella della classe `Cluster` che rappresenta il cluster attraverso i suoi parametri: media, matrice di covarianza e label. Per entrambe si è scelta un'implementazione a template che richiede due parametri: uno è la dimensione dello stato e l'altro il tipo di elementi

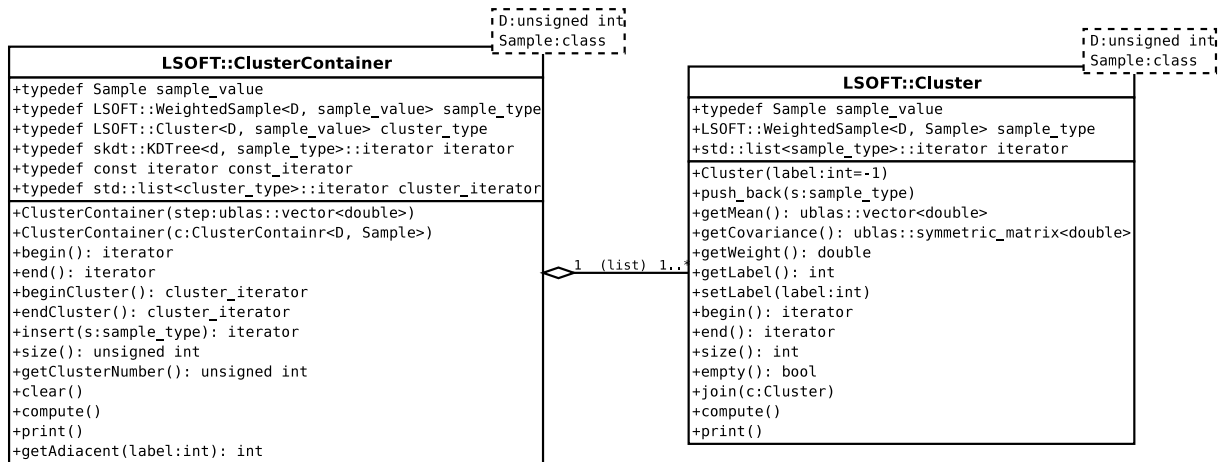


Figura 3.12: Diagramma delle classi che implementano il clustering.

che rappresentano i sample.

LSOFT::ClusterContainer utilizza un kd-tree per i campioni e una lista per memorizzare i cluster che vengono individuati di iterazioni in iterazione.

Il metodo LSOFT::ClusterContainer::insert(sample_type s) viene richiamato ogni qual volta è necessario memorizzare un campione e che, come accennato in precedenza, viene sottoposto all’algoritmo di clustering 8.

Il metodo LSOFT::ClusterContainer::compute() deve essere chiamato quando l’ultimo campione generato dal filtro partellare è stato inserito. Il suo compito è quello di calcolare le statistiche per ognuno dei cluster che sono stati trovati e che sono contenuti in una lista.

LSOFT::ClusterContainer::getAdiacent(int label) consente di consultare la mappa di corrispondenza (3.4) fornendo l’etichetta del cluster nel quale quello che possedeva il valore del parametro è stato inglobato.

L’accesso agli elementi del contenitore sono forniti mediante iteratori. In particolare è possibile ottenere un iteratore alla lista dei cluster e un iteratore al kd-tree che contiene i campioni.

LSOFT::Cluster rappresenta un cluster: contiene una lista con l’insieme dei campioni, la media, la covarianza, il peso e la label.

Il metodo LSOFT::Cluster::push_back(Sample s) inserisce un elemen-

to nella lista e aggiorna il peso del cluster.

Il metodo `LSOFT::Cluster::compute()` calcola la media e la matrice di covarianza relativa al cluster.

`LSOFT::Cluster::join(LSOFT::Cluster c)` ha il compito di unire il cluster a `c`. Tutti i parametri del cluster vengono aggiornati e il set di campioni unito a quello di `c` che viene eliminato assegnandogli una label negativa in modo che non venga più conteggiato.

L'accesso agli elementi contenuti nel cluster è garantito dai metodi `begin()` e `end()` che forniscono gli iteratori della struttura dati. Gli altri parametri sono accessibili dai corrispondenti metodi *get* e *set*.

Capitolo 4

Integrazione del localizzatore nel robot mobile

L'implementazione del localizzatore consente da subito di eseguire delle simulazioni ma per ottenere dei dati sperimentali occorre svolgere dei test in un contesto reale. In questo capitolo verranno introdotti gli strumenti utilizzati per realizzare un apparato che consenta l'utilizzo del localizzatore a bordo di un robot mobile, a cominciare dai componenti hardware che sono costituiti dal robot stesso e da un sensore laser scanner. Infine saranno descritti gli strumenti software utilizzati e sviluppati nel corso di questo lavoro di tesi.

4.1 Componenti Hardware del sistema

4.1.1 Il robot mobile

Il robot mobile utilizzato per i test del localizzatore è un Nomad 200 prodotto dalla statunitense *Nomadic Technologies*. si tratta di un robot di medie dimensioni dotato di numerosi sensori e di un sistema di elaborazione autonomo costituito da un PC. La figura 4.1 ritrae il Nomad: ha una forma pressoché cilindrica, un'altezza di circa 80cm e un diametro di 50cm.

L'apparato locomotore del robot è costituito da tre ruote di identiche dimensioni disposte ai vertici di un triangolo equilatero. L'avanzamento del Nomad è spinto da un motore elettrico che imprime una coppia equamente ripartita alle tre ruote per

mezzo di un sistema di cinghie. Un secondo motore è deputato allo sterzo che avviene per rotazione solidale delle ruote. Un terzo motore si occupa della rotazione della torretta costituita dalla parte superiore del robot, il cui movimento è completamente indipendente dalla base. Il Nomad non possiede un sistema di locomozione *olonomo* in quanto il moto traslazionale non è svincolato dalla variazione di direzione; tuttavia gli rimane la possibilità di ruotare da fermo intorno al proprio asse. Il Nomad possiede un ricco apparato sensoristico. I sensori *propriocezionali* sono co-



Figura 4.1: NOMAD 200 e SICK LMS 200.

stituiti da un apparato per l'odometria e da una bussola. L'odometria viene fornita in tempo reale in termini di spostamenti assoluti e velocità di traslazione e rotazione. Tali grandezze vengono ottenute dall'elaborazione della velocità degli assi dei tre motori. La bussola restituisce l'orientamento assoluto rispetto al nord magnetico della Terra.

La sensorialità *eterocettiva* si avvale di sedici sonar e altrettanti emettitori infrarossi per la misura delle distanze e di una cintura di bumper posta alla base per il rilevamento dei contatti.

Il sistema di elaborazione è costituito da un PC realizzato con componenti commerciali che si interfaccia ai vari apparati del Nomad mediante schede realizzate e tarate ad hoc per il robot. Il PC di bordo originariamente era dotato di un processore Intel 486. Presso il Laboratorio di Robotica (RIMLab) del Dipartimento di Ingegneria dell'Informazione è stato aggiornato in modo da conferire maggiore potenza di calcolo al robot di bordo [17]. Oggi il Nomad a disposizione del *RIMLab*, monta un *Intel Pentium III* con 1GHz di frequenza di clock. Visti i costi e le prestazioni dei componenti dei PC moderni la tentazione di un nuovo aggiornamento sarebbe forte se non fosse per le schede di interfacciamento con i sensori e gli attuatori che espongono un'interfaccia *ISA*[17] che è ormai obsoleta e difficilmente viene inserita nelle moderne schede madri. Inoltre è assai difficile sostituire tali schede con una versione più recente perché la *Nomadic Technologies* ha abbandonato il progetto Nomad già da molti anni.

4.1.2 Il laser scanner

Le misure di distanza sono l'unica sorgente eterocettiva impiegata dal localizzatore realizzato. Dal momento che il *RIMLab* dispone di un laser scanner, si è pensato di utilizzarlo per questo scopo, in quanto i sensori di prossimità di bordo del Nomad sono attualmente fuori uso; inoltre il laser ha una portata molto elevata ed è in grado di fornire misure più precise.

Il modello del laser è *SICK LMS 200*; esso emette fasci di infrarossi ed è prodotto dalla tedesca *SICK inc*[18] per scopi industriali. Il *SICK 200* si presenta con una robusta scocca in metallo che espone una finestra per l'emissione dei fasci laser. La figura 4.2 mostra un'immagine del dispositivo.

Il principio di funzionamento è basato sulla misura del tempo di volo del fascio: dopo lo sparo, viene misurato il tempo necessario perché il raggio riflesso da un oggetto ritorni indietro, e sulla base di questo, viene calcolata la distanza.

Il *SICK* è dotato di un'unica sorgente laser infrarossa pulsata che viene riflessa da uno specchio rotante in modo da scansare un'area compresa fra 0° e 180° . I dati ricavati dalla scansione vengono organizzati in record, memorizzati in un buffer FIFO e inviati a flusso continuo attraverso un'interfaccia seriale compatibile con gli standard RS-232 e RS-422.



Figura 4.2: SICK LMS 200

Alimentazione

Il laser richiede una tensione di $24V \pm 15\%$ e consuma una potenza di 17.5W. Con queste specifiche sui consumi, è uno strumento che si presta all'alimentazione tramite batterie.

Scansione

Il *SICK 200* prevede diverse modalità di scansione a diverse risoluzioni spaziali e angolari.

- *Ampiezza della scansione.* Può essere impostata a 180° oppure a 100° , come mostrato in figura 4.3.
- *Risoluzione angolare.* Può essere impostata a 1° , 0.5° oppure a 0.25° se l'ampiezza della scansione è 100° . A seconda della risoluzione cambia la mole di dati prodotta perché diverso è il numero di fasci sparati. Ciascuna misura è un intero memorizzato in 2 byte. Ad esempio, con una risoluzione di 1° e un'ampiezza di 180° vengono sparati 181 fasci e, quindi, altrettante saranno

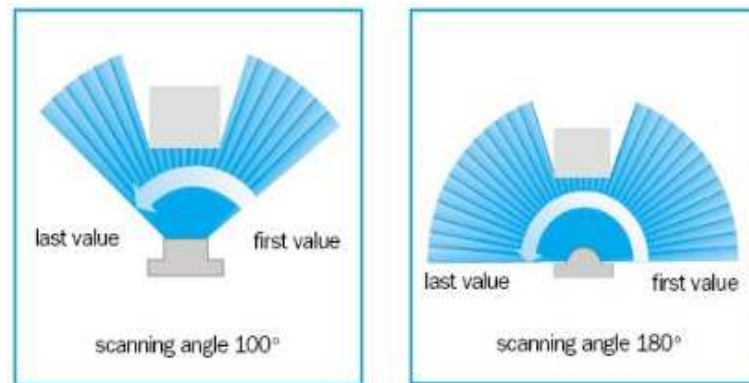


Figura 4.3: Ampiezza della scansione del SICK 200

le misure ottenute. Il massimo numero di valori rilevati ammonta a 401, che si ottengono con risoluzione 0.25° e ampiezza (obbligata) 100° .

- *Raggio massimo.* Il data-sheet del *SICK* riporta diversi valori per diverse condizioni. Infatti la misurazione è pesantemente condizionata dalla riflettività delle superfici. Il valore riportato per un normale oggetto opaco è di 30m, che scendono a 4 se la riflettività è prossima all'1.8%. È possibile avvalersi di particolari superfici riflettenti da applicare sugli oggetti per aumentare la portata fino a 150m.
- *Risoluzione della lunghezza.* È 10mm.
- *Incertezza.* Nel data-sheet è data a ± 15 mm entro gli 8 metri, e di ± 40 mm fino a 20 metri. La riflettività delle superfici gioca un ruolo fondamentale anche in questo caso perché tali dati sono validi se essa è almeno del 10% e 30%.

Formato e trasferimento dei dati

- *Tempo di risposta.* Dipendentemente dalla risoluzione angolare usata (cioè dal numero di fasci), il tempo richiesto fra due invii successivi è dato da 52/26/13 ms. I dati provenienti dal data-sheet sono stati confermati da un ciclo di prove sperimentali svoltosi presso il RIMLab.
Per una risoluzione di 0.25° su 100° , è stato rilevato un tempo medio fra due

acquisizioni successive pari a $13.6ms$ con una deviazione standard di $3.9ms$. Nel caso di risoluzione 0.5° su 180° le misure hanno dato un tempo medio di $26.7ms$ con una deviazione di $7.1ms$.

Infine, con 1° su 180° i risultati sono stati $13.2ms$ per la media e $7.64ms$ per la deviazione standard. Va sottolineato che questi dati sono stati ottenuti utilizzando la massima capacità trasmissiva dell'interfaccia seriale che ammonta a $500Kbaud$.

- **Formato dei dati.** Il laser comunica con l'uso di pacchetti del formato presentato in figura 4.4. L'header è di 4 byte che indicano l'inizio del pacchetto, la dimensione del payload e l'indirizzo che vale 0×00 per i pacchetti in ingresso e 0×80 per quelli in uscita. Segue un campo di controllo che indica la natura dei dati trasportati. Infine, il payload contiene i dati (2 byte per misura) e termina con CRC che consente di individuare i dati corrotti.

| | | | | | |
|-----|------|-----|----------------------|---------|-----|
| STX | ADDR | LEN | command/ response | PAYLOAD | CRC |
|-----|------|-----|----------------------|---------|-----|

Figura 4.4: Struttura dei pacchetti utilizzati dal protocollo del SICK 200.

- **Velocità di trasferimento e interfaccia seriale.** La comunicazione fra il SICK e l'esterno avviene tramite interfaccia seriale. I formati supportati sono due: RS232 e RS422. Il primo è il normale standard seriale, quello che si trova sulla maggior parte dei PC commerciali e consente un trasferimento alle velocità di 9.6, 19.2, 38.4 e $Kbaud$. L'RS422 è lo standard di comunicazione seriale più rapido e consente una velocità di trasferimento che arriva fino a $500 Kbaud$. L'aspetto più interessante di questi standard è quello di utilizzare un connettore identico ad eccezione di due collegamenti interni scambiati fra loro, il che consente di passare dall'uno all'altro con un semplice adattatore. Nel SICK il connettore è unico ed è quello dell'RS422, ma il collegamento può essere fatto utilizzando proprio un adattatore per modificare lo standard. La figura 4.5 mostra l'interfaccia seriale.

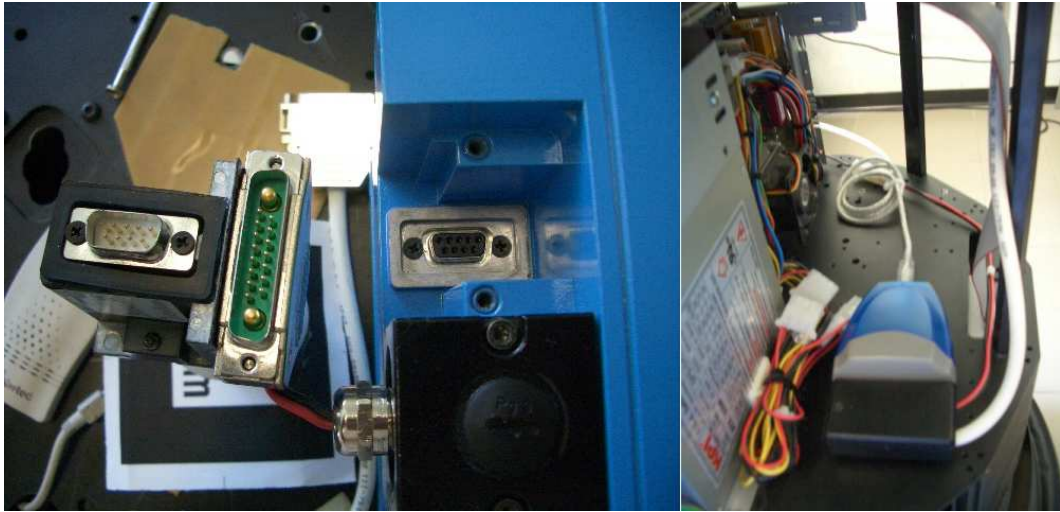


Figura 4.5: Connettore del SICK LMS 200 e l'adattatore USB-RS422 utilizzato per connetterlo al Nomad.

4.2 Componenti Software

4.2.1 Smartsoft

SmartSoft è un framework per lo sviluppo di applicazioni di controllo di robot che si devono interfacciare con sensori ed attuatori potenzialmente distribuiti. Le caratteristiche che lo contraddistinguono sono il ricorso a CORBA e l'approccio basato su componenti distribuiti che interagiscono fra loro secondo particolari pattern di comunicazione. Uno degli elementi chiave di Smartsoft è il concetto di *componente* (o *modulo*). Le applicazioni scritte usando SmartSoft hanno tutte una struttura modulare dove ciascun componente è autonomo e capace di sovrintendere ad una particolare funzione in un regime di concorrenza. In una tipica applicazione per la robotica potrebbero essere presenti diversi componenti, uno per ogni gruppo di sensori, coordinati assieme da un modulo preposto.

In questo trattato si farà riferimento all'implementazione di SmartSoft chiamata *OROCOS::Smartsoft*[19] che utilizza CORBA[20], ma va evidenziato che ne esistono anche altre versioni che sfruttano meccanismi di comunicazione diversi.

Benché sia già stato citato il concetto di *componente*, è bene fissare in modo definitivo il significato.

Componente

Un componente è una parte software che è in grado di svolgere una particolare funzione in modo autonomo e indipendente dal contesto in cui opera. Un componente può essere progettato e implementato senza tener conto dell'utilizzo che ne verrà fatto. L'attività di ogni componente è svincolata da quella degli altri, con i quali ha modo di interagire sfruttando i pattern di comunicazione di *OROCOS::SmartSoft*. Grazie alle *API*, i componenti hanno un'interfaccia e una struttura ben definita e ciò riduce la complessità della progettazione del sistema complessivo che risulta decisamente modulare.

Interazioni fra i componenti

Come ampiamente specificato, i componenti sono entità software piuttosto indipendenti e interagiscono grazie ad un sistema di comunicazione basato su *CORBA*.

CORBA rientra nella classe degli *ORB (Object Request Broker)*: i componenti comunicano attraverso lo scambio di oggetti dei quali chiamano poi i metodi necessari al compimento dell'attività. Inoltre è caratteristica comune degli *ORB* quella di possedere un protocollo (*IIOP* [21]) che consente la comunicazione interpiattaforma, caratteristica che è stata sfruttata appieno da *SmartSoft*. In sostanza è possibile realizzare applicazioni costituite da più componenti distribuiti fra più piattaforme in modo tale da sfruttare al meglio le risorse hardware a disposizione; molto spesso infatti, i robot non sono dotati di grandi capacità computazionali e quindi può essere vantaggioso spostare l'elaborazione su PC remoto che sia dotato delle risorse necessari all'ottimale esecuzione dell'applicazione.

Il netto disaccoppiamento delle varie parti del sistema si dimostra una caratteristica molto utile anche in fase di test poiché rende più semplice l'interazioni fra elementi simulati e reali (figura 4.6).

Schemi di comunicazione

In *SmartSoft* gli schemi di comunicazione (o *Communication Pattern*) descrivono il modo in cui avviene la comunicazione fra i componenti, cioè ne descrivono la dinamica. Parametrizzando un pattern con un *Communication Object* si realizza un servizio di comunicazione le cui modalità di accesso sono definite dal pattern stesso.

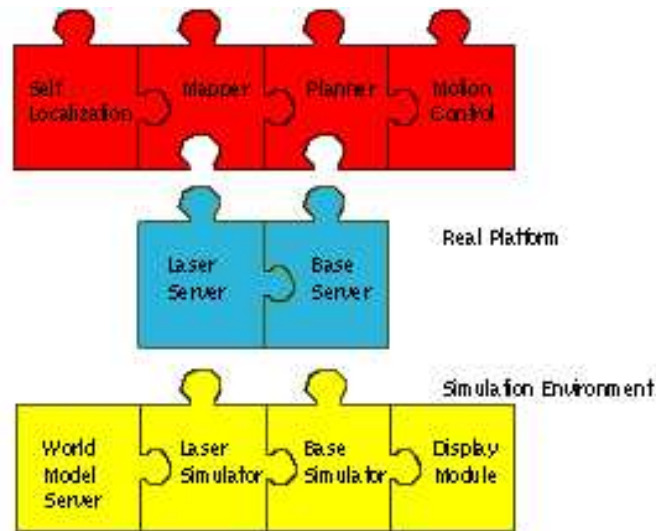


Figura 4.6: Sintesi del concetto di intercambiabilità fra componenti simulati e reali.

In termini di *CORBA* i *Communication Object* sono ciò che rappresenta il contenuto del messaggio della comunicazione.

I pattern che *OROCOS::Smartsoft* mette a disposizione sono i seguenti:

- **Send.** Invio senza risposta. Utilizzata in comunicazioni unidirezionali. Particolarmente utilizzato per inviare parametri di configurazione.
- **Query.** Si tratta di una Send bloccante. Il client rimane in attesa della risposta indeterminatamente.
- **Push Newest e Push Timed.** È una comunicazione uno a molti nella quale i client richiedono ad un server l'invio periodico di un dato. La differenza fra *Newest* e *Timed* è che nel primo caso avviene un invio ogni qual volta è disponibile un nuovo dato, nel secondo viene inviato il dato più recente tenendo conto del periodo richiesto da ogni client.
- **Event.** Tipo di comunicazione asincrona fra il generatore dell'evento e il server di gestione dello stesso. Gli eventi sono gestiti mediante appositi handler derivati dalle classi base del framework.
- **State.** Consente la gestione dello stato interno di un componente proteggendolo da interruzioni entro sezioni critiche, causate da richieste esterne. Questo

pattern è correlato alla comunicazioni in quanto la sospensione delle attività di un componente può richiedere la sospensione delle relazioni con i componenti esterni quali query pendenti, sottoscrizioni, etc. . . Fra le sue competenze si sono anche quelle di gestire l'accesso a risorse critiche e di garantire la consistenza dei dati. In sintesi, consente di implementare una serie di funzioni molto simili a quelle di un monitor.

- **Wiring.** Consente di gestire il controllo del flusso di comunicazione fra i componenti in modo dinamico durante l'esecuzione. Il *wiring pattern* mette a disposizione un serie di primitive per scambiare messaggi di controllo fra i componenti.

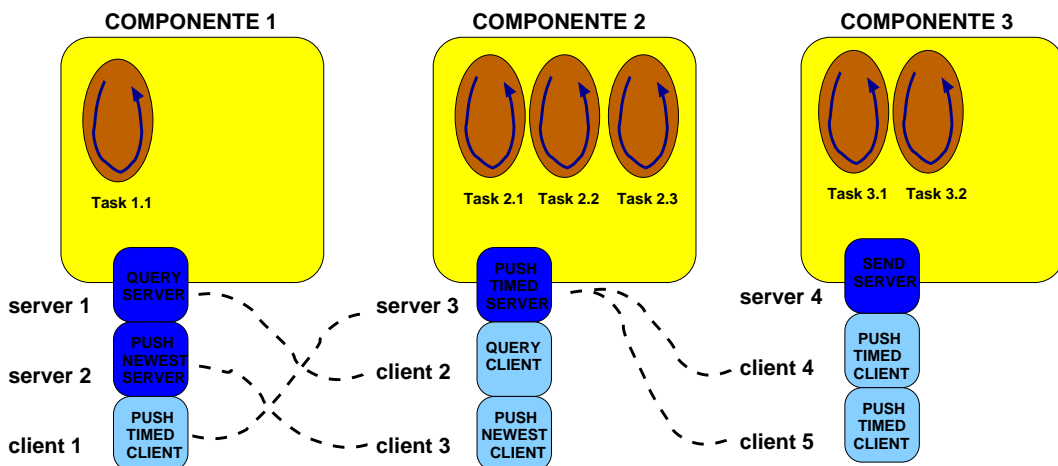


Figura 4.7: Un possibile sistema Smartsoft.

La figura 4.7 mostra un ipotetico sistema formato da tre componenti e dai rispettivi oggetti per la comunicazione che costituiscono l'unico strumento di interazione inter-componente. La comunicazione intra-componente avviene secondo un normale paradigma di chiamata a metodi. Un componente può contenere più oggetti di tipo *SmartTask*, ciascuno facente capo ad un thread del SO, il che consente di instaurare un regime di concorrenza all'interno di un singolo modulo. Il framework mette a disposizione alcuni strumenti per la sincronizzazione fra i task ma non quelli necessari a realizzare un'applicazione real-time.

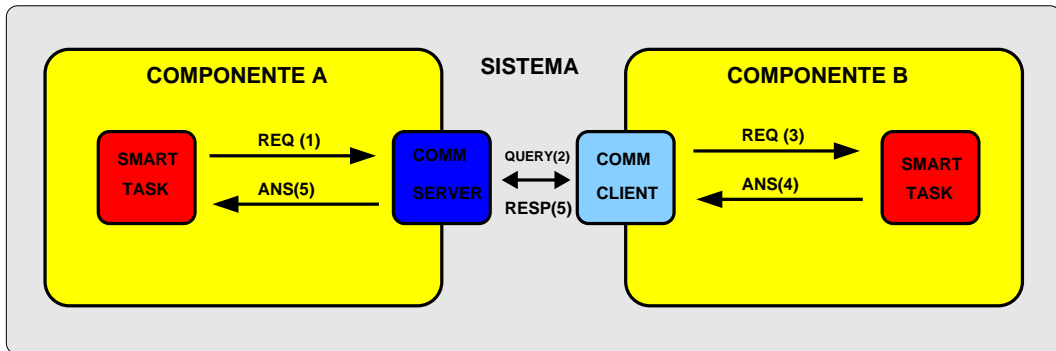


Figura 4.8: Schema di comunicazione in un sistema SmartSoft come somma di comunicazioni intra ed inter componente.

4.2.2 Il Name Service

Il *Name Service* è un'applicazione comune a tutti gli *ORB* (Object request Broker) e pertanto anche di *CORBA*. Essa ha il compito di pubblicare i servizi disponibili nel suo dominio. Una qualunque entità che desideri accedere ad un servizio, deve rivolgersi al *Name Service* di modo che, se la risorsa è effettivamente pubblicata, faccia da tramite con il fornitore. Nel caso di *OROCOS::SmartSoft* l'implementazione di riferimento del *Name Service* è quella di *CORBA*[22] il cui schema di funzionamento è sintetizzato in figura 4.9.

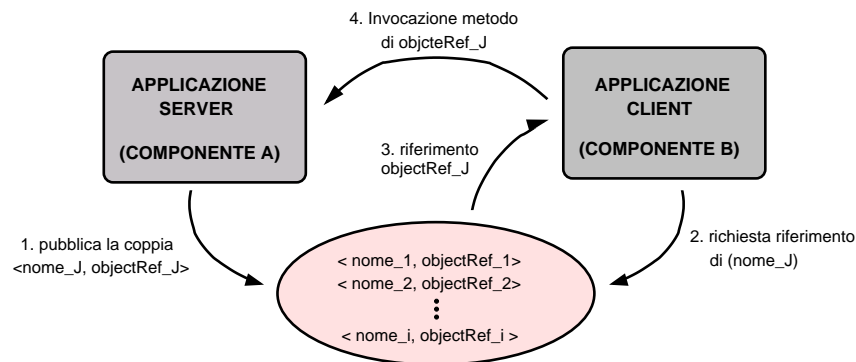


Figura 4.9: Meccanismo di funzionamento del *Name Service* di *CORBA*.

Struttura dei componenti

Cosa sia un componente e cosa più o meno debba contenere è già emerso nel corso dei precedenti paragrafi e dalle figure 4.7 e 4.8. Il framework mette a disposizione classi che realizzano i concetti e gli elementi precedentemente descritti e perciò definire un componente significa scrivere un'applicazione la cui struttura segue uno schema predefinito.

Nella funzione `main()` avviene la creazione di un oggetto della classe `CHS::SmartComponent` che rappresenta il vero e proprio componente nel sistema complessivo. Il suo costruttore richiede le generalità del *Name Server* e una stringa che rappresenta il nome con il quale vi si deve registrare. Vengono creati degli oggetti delle classi che implementano i *Communication Pattern* e quelli dei thread, le cui classi derivano da `CHS::SmartTask`. Dopo che gli oggetti necessari sono stati istanziati, vengono avviati i task e il componente stesso e così, l'applicazione entra in funzione.

4.3 Installazione del SICK 200 sul Nomad

La figura 4.1 mostra il Nomad completo del laser scanner. Il SICK è stato installato sulla parte superiore del robot con la parte anteriore orientata in modo concorde al senso di marcia. Le batterie sono state alloggiare in uno scomparto ricavato nel telaio del Nomad e conferiscono al laser un'autonomia stimata in circa dieci ore. Il *SICK 200* possiede un'interfaccia seriale compatibile con i due standard *RS232* e *RS442*. Le scelte possibili erano quelle di collegare il Nomad utilizzando l'interfaccia seriale *RS232* oppure quella di utilizzare un convertitore *RS422 - USB*. È stata preferita la seconda in virtù della maggiore velocità di trasmissione raggiungibile: *500 Kbaud* contro i *38.4 Kbaud* dello *RS232*. Il convertitore utilizzato è dell'*Easy Sync* ed è dotato di un chip *FTDI* il cui modulo (*ftdi_sio*) è presente nei kernel linux almeno dalla versione 2.6.12. La figura 4.10 lo mostra installato a bordo del Nomad; l'interfaccia *USB* è ben visibile mentre quella seriale proveniente dal laser è inserita nella scatola nera.



Figura 4.10: RS422 - USB installato a bordo del *Nomad*.

4.4 La piattaforma software

Il sistema complessivo realizzato comprende tre componenti *SmartSoft*, il *Nomad* equipaggiato con uno scanner laser *SICK LMS 200* e un calcolatore. La figura 4.11 mostra la piattaforma in tutte le sue parti, software e hardware. Il PC di bordo del robot viene utilizzato per il componente che ne gestisce il controllo dei motori, la lettura dell'odometria e le scansioni del laser. L'elaboratore remoto ospita i componenti per il controllo del movimento e del localizzatore.

4.4.1 Il localizzatore

Il componente del localizzatore contiene un unico *SmartTask* ciclico che si occupa della lettura dei rilevamenti sensoriali e della localizzazione. La comunicazione con lo *Smart Nomad Server* avviene per mezzo di due diversi pattern (4.2.1): un *query pattern* per la scansione laser e un *push timed pattern* per l'odometria. Il componente contiene, infatti, gli oggetti che costituiscono uno degli estremi del canale di comunicazione e cioè i client dei pattern parametrizzati con gli opportuni comunicatori:

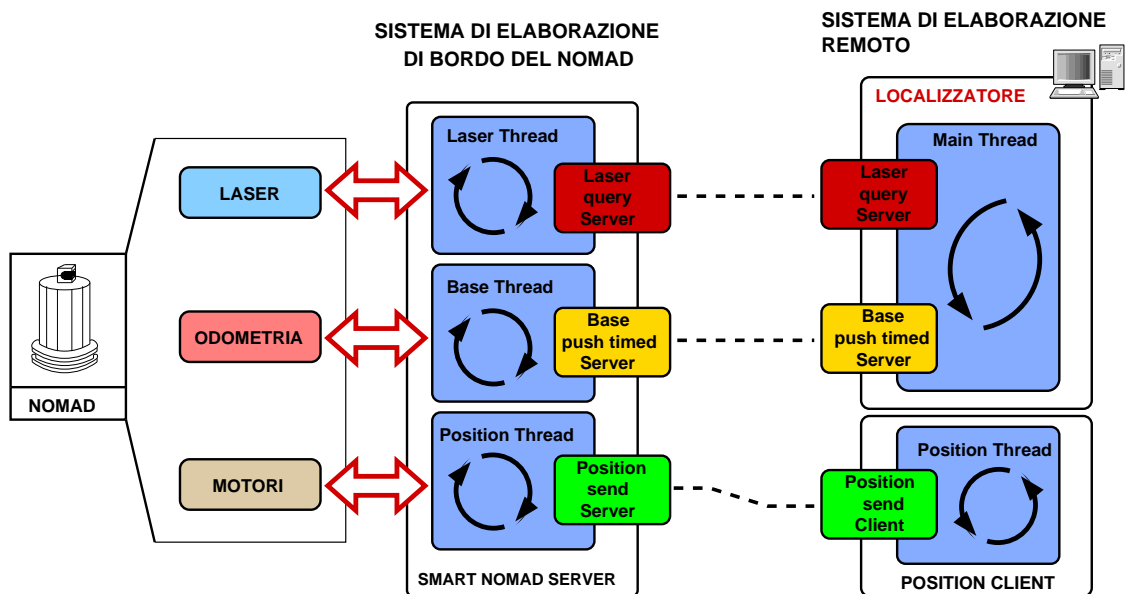


Figura 4.11: Sistema completo con il localizzatore posto su una piattaforma remota.

```
CHS::PushTimedClient<Smart::CommBaseState>
```

```
CHS::QueryClient<Smart::CommMobileLaserScan>
```

Il servizio *Push Timed* richiede una sottoscrizione al server che periodicamente invia un oggetto comunicatore `Smart::CommBaseState` che contiene i dati odometrici. La scelta del periodo di sottoscrizione non ha comportato particolari difficoltà in quanto è stato sufficiente scegliere un valore di tempo molto inferiore a quello richiesto dal compimento di un'iterazione del filtro particellare: 10 ms . In questo modo è sempre possibile avere dati odometrici sufficientemente aggiornati. Il servizio *Query* per il laser funziona secondo un paradigma *send - receive* bloccante per cui a seguito di una richiesta, il client attende i dati della scansione che verranno inviati mediante un `Smart::CommMobileLaserScan`. Il collo di bottiglia è localizzato nella trasmissione perchè il *SICK 200* è in grado di produrre 75 scansioni al secondo alla risoluzione di 1° con cui viene utilizzata. Tuttavia il tempo necessario alla trasmissione dei dati dal laser al PC è esigua rispetto al tempo richiesto dal localizzatore perfino alla velocità di 9.6 kbaud .

Dopo ogni lettura sensoriale, i dati vengono elaborati per essere successiva-

Algoritmo 9 Main Thread del Componente Localizzatore

- 1: inizializzazione del localizzatore
 - 2: **while** *true* **do**
 - 3: Invia una richiesta al laser-server attraverso il query client.
 - 4: attendi la risposta.
 - 5: seleziona i beam corrispondenti a 0° , 90° e 180° .
 - 6: rileva i dati odometrici dal push-timed client.
 - 7: converti tutte le lunghezze in metri e gli angoli in radianti.
 - 8: esegui una iterazione dell'algoritmo di localizzazione utilizzando i dati sensoriali appena acquisiti.
 - 9: scrivi il log fi le dei risultati e disegna i campioni sulla mappa.
 - 10: **end while**
-

mente forniti all'algoritmo di localizzazione. Il condizionamento consiste semplicemente nella conversione delle lunghezze in metri, degli angoli in radianti e nella selezione di sole tre fra tutte le 181 misure fornite dal laser: quella corrispondente a 0° , quella a 90° e quella a 180° . La ragione di questo drastico sfooltimento dei dati provenienti dal laser, è il costo computazionale dell'algoritmo di localizzazione, che cresce molto rapidamente con la quantità di informazioni sensoriali da trattare. In letteratura sono riportati diversi esempi di come pochi fasci consentano comunque una buona accuratezza nella localizzazione. Ad esempio in [4] ne vengono utilizzati soltanto due. Nella versione del componente creata per i test, una volta che il localizzatore ha generato l'ipotesi per l'iterazione corrente, i campioni vengono disegnati sulla mappa dell'ambiente in modo da avere una visione d'insieme sull'operato del localizzatore.

4.4.2 Il Nomad Server

Il compito di questo componente è quello di interfacciare l'hardware del *Nomad* con il resto del sistema. Esso Incorpora tre *SmartTask*: uno per la gestione del laser, uno per i sensori odometrici e uno per il controllo dei motori. Il componente mette a disposizione tre servizi: un *query server* per la gestione delle richieste per le scansioni laser e l'invio delle relative risposte; un *push timed server* per l'invio dei dati odometrici ai client che hanno effettuato la sottoscrizione; infine, un *send server* per gestire l'arrivo di comandi per la navigazione in forma di

< *spostamento, rotazione* >. Dal momento che si è ritenuto di non utilizzare la torretta del Nomad, questa viene ruotata in modo solidale alla base inviandole lo stesso angolo di rotazione della base. I thread accedono all'hardware utilizzando i driver distribuiti con *OROCOS::SmartSoft*; va notato che quello per il *SICK 200* non è in grado di gestire la connessione a 500 *Kbaud* consentita dell'interfaccia seriale *RS422*.

4.4.3 Position Client

Il Position Client è il componente preposto all'invio di comandi di navigazione al *Nomad Server*. Per la comunicazione utilizza un *Send Client*:

```
CHS::SendClient<Smart::CommNavigationPosition>
```

I comandi vengono inviati con il comunicatore in coppie < *spostamento, rotazione* >. Il componente espone all'utente un'interfaccia testuale e la possibilità di scegliere fra diverse modalità di funzionamento durante l'esecuzione con la pressione di alcuni tasti della tastiera.

Modalità di funzionamento e interfaccia utente

Si procede alla descrizione delle varie possibilità del *Position Client*:

- *Navigazione per step*. È la modalità predefinita. Con la croce direzionale formata dai tasti 'd', 'f', 'g', 'r' si invia al *Nomad Server* un comando di movimento in avanti ('r'), indietro ('f'), rotazione positiva ('g') o negativa ('d'). L'ampiezza dei movimenti è calibrata con la croce 'h', 'j', 'k', 'u' con logica identica a quella del movimento: 'h' e 'k' riducono e aumentano l'ampiezza della rotazione, 'j' e 'u' fanno lo stesso con quella del movimento. La convenzione adottata è che i valori positivi significano movimenti in avanti e rotazioni in senso antiorario. A questa modalità si accede premendo il tasto '4' oppure inserendo < 0, 0 > dalla modalità *Navigazione per coppie di valori*.
- *Navigazione per coppie di valori*. Al prompt della shell vengono richiesti due valori, il primo è un movimento e il secondo un angolo. La convenzione dei segni è la stessa della *Navigazione per step*.

Si accede a questa modalità con la pressione del tasto '5' e la si termina inserendo la coppia $\langle 0, 0 \rangle$.

- *Navigazione programmata.* I dati da inviare al *Nomad Server* vengono prelevati da un file. Alla pressione del tasto '3' viene richiesto il file da aprire. Da questa modalità si passa alla *Navigazione per step* al termine del file.
- *Registrazione.* I comandi inviati al *Nomad Server* vengono anche registrati su un file del quale viene anche fatta una copia con estensione *.reversed* a comandi invertiti in modo da poter guidare il robot alla posizione di partenza. Premendo il tasto '1' i comandi vengono dati nella modalità *Navigazione per step*, mentre con '2' nella modalità *Navigazione per coppie di valori*.

4.5 Soluzione alternativa per la Piattaforma Software

La soluzione presentata in 4.4 ha un'alternativa che prevede l'integrazione della gestione del laser direttamente all'interno del componente del localizzatore. Tale scelta ha uno svantaggio fondamentale che sta nell'obbligo di eseguire il componente sul sistema a cui è collegato il laser e cioè il *Nomad*. Ci sono stati due validi motivi per cui è stata provata questa soluzione. Uno è la disponibilità di un driver che a differenza di quello di *SmartSoft*, consente l'utilizzo di tutta la banda messa a disposizione dell'*RS422*. Il secondo motivo è stato il comportamento mostrato da un'applicazione di prova del laser, creata con *SmartSoft* che aveva messo in evidenza delle irregolarità non sistematiche nel trasferimento dei dati dal laser al client: all'invio continuo di dati si alternavano pause di alcuni secondi. Così, la soluzione presentata in 4.4 era stata momentaneamente messa da parte in favore di quella presentata in questa sezione.

Il motivo per cui si è poi ritornati alla decisione di svincolare il laser dal localizzatore, è che il *Nomad*, con il suo Intel Pentium III a 1GHz, si è rilevato inadeguato per sostenere il peso computazionale dell'algoritmo di localizzazione. La decisione di ritornare al precedente approccio è stata presa solo successivamente a nuovi test sulla gestione del laser in *SmartSoft* dai quali è emerso che il comportamento anomalo precedentemente citato non si presenta o è trascurabile nel sistema completo.

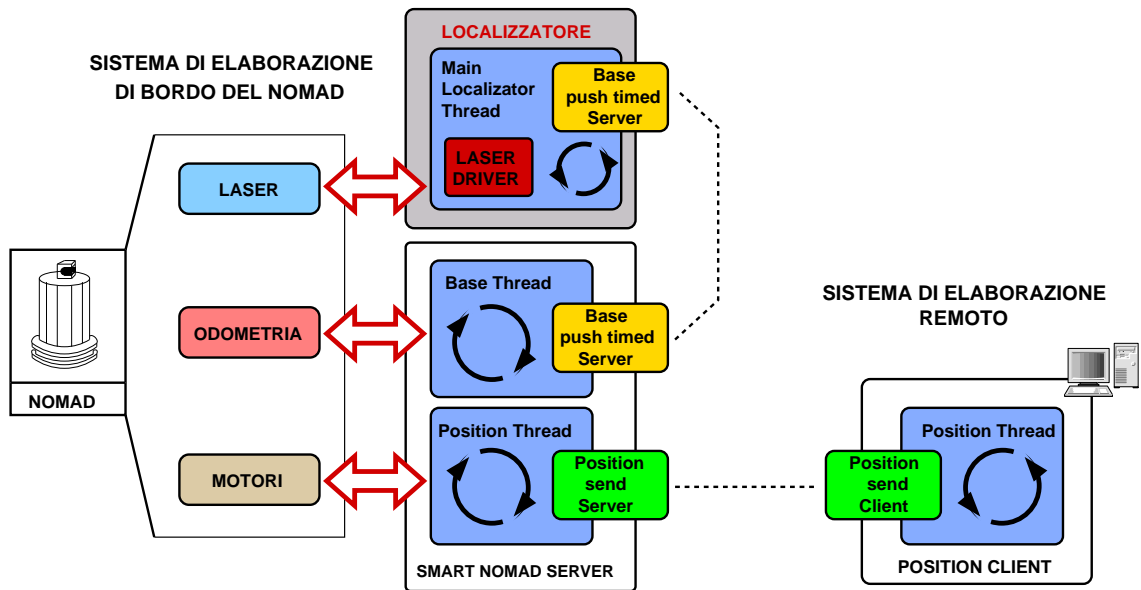


Figura 4.12: Sistema completo con il localizzatore con driver veloce per il laser scanner.

Probabilmente ciò è dovuto alla minore frequenza di richieste che il *Nomad Server* deve gestire grazie al tempo di esecuzione del localizzatore.

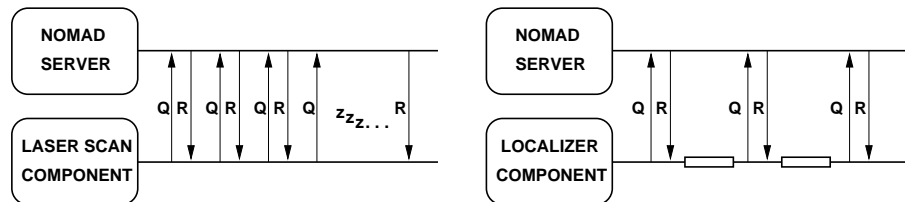


Figura 4.13: a. Anomalie nella comunicazione fra Nomad Server e un client che richiede misure dal laser. b. Con un opportuno ritardo le anomalie diventano trascurabili.

Driver del Laser

Il driver impiegato[23] consente di impostare la comunicazione fra laser e PC a tutte le velocità da 9.6 *Kbaud* fino a 500 *Kbaud*. Il file `Driver.h` definisce l'interfaccia della classe `Driver` e la struttura dati che serve per configurare il `SICK`. Al momento della costruzione il laser viene avviato e configurato. Durante tutta l'at-

tività esso scansisce l'ambiente ininterrottamente e memorizza i dati in un buffer FIFO. Ad ogni richiesta di lettura da parte del driver, viene restituito il primo pacchetto di misure dalla coda. Solo alla distruzione dell'oggetto `Driver` il laser si ferma.

Il driver funzionava correttamente ma non consentiva lo svuotamento del buffer e questo determinava l'impossibilità di prelevare la misura più aggiornata. Il localizzatore, invece, necessita di dati sensoriali molto recenti. È stato sufficiente modificare il metodo `Driver::Read()` in modo che, a seconda della configurazione, svuoti o meno il buffer prima di effettuare la lettura.

Capitolo 5

Il collaudo del sistema di localizzazione

In questo capitolo vengono illustrati i risultati delle prove ottenute in simulazione e con esperimenti nel mondo reale.

L'obiettivo delle simulazioni è stato, da un lato la verifica della correttezza funzionale del localizzatore a seguito delle estensioni introdotte (in particolare l'algoritmo di clustering), dall'altro le prestazioni al variare di alcuni parametri. L'ambiente scelto per svolgere le prove è il corridoio del piano terra della *Palazzina 1 del Dipartimento di Ingegneria dell'Informazione* la cui mappa è mostrata in figura 5.1. Un limite di questo ambiente è l'assenza di percorsi chiusi. Tale caratteristica, sebbene sia significativa soprattutto per i problemi di mapping, compare molto spesso nei test di localizzazione presentati in letteratura; la presenza di un anello chiuso introduce elementi di simmetria che creano ambiguità nei dati sensoriali. Tuttavia anche nella mappa utilizzata la relativa similitudine dei due tratti di corridoio costituisce un elemento sufficientemente critico che mette in evidenza la formazione di ipotesi multiple. Si noti inoltre che la mappa comprende esclusivamente il corridoio, ossia un locale caratterizzato dall'assenza di oggetti ed ostacoli temporanei. Si ricordi, infatti, che un filtro bayesiano può operare correttamente solo se rimane valida l'ipotesi di stato completo (2.1).



Figura 5.1: Mappa della Palazzina 1 del Dipartimento di Ingegneria dell'Informazione. Dimensioni massime 34.04 x 6.88 metri. Risoluzione: 1 pixel = 1 mm.

5.1 Prove di simulazione

Per le prove di simulazione si è fatto percorrere al robot virtuale tre differenti percorsi ricavati nella mappa della palazzina, ritratti in figura 5.2. I percorsi sono stati scelti in modo tale da porre il localizzatore in situazioni di ambiguità sfruttando gli elementi di simmetria costituiti dai due corridoi laterali. Il sistema di localizzazione utilizzato è quello messo a disposizione dalla libreria di localizzazione [11] con algoritmo RTPF. Per tutte le prove sono state utilizzate sempre due partizioni. Il numero di particelle influenza la capacità di un filtro particellare di convergere al risultato corretto, in particolare maggiore è il loro numero e migliori sono le prestazioni in tal senso. Tuttavia anche il costo computazionale è fortemente legato alla quantità di particelle utilizzata. Per questa ragione le varie prove sono state eseguite utilizzando quantità differenti di particelle: 1200, 2000, 4000. La valutazione è stata fatta sulla base del numero di volte in cui la localizzazione è riuscita e sulla media dell'errore minimo commesso sulla stima della posizione, calcolata utilizzando i dati ricavati dalle simulazioni in cui la localizzazione è avvenuta correttamente.

Va definito cosa si intende per *localizzazione corretta*. Visivamente si osserva l'andamento della distribuzione dei campioni: in caso di successo si addensano in una nuvola nei pressi della posizione del robot per poi seguirlo lungo il suo percorso (figura ??). Il risultato dell'esame visivo viene poi convalidato dall'analisi dell'andamento dell'errore che tende ad essere elevato all'inizio per poi ridursi e mantenersi basso durante la fase di tracking che segue il momento in cui il localizzatore individua la posizione del robot.

La tabella 5.1 mostra un riepilogo delle simulazioni svolte e riporta il numero di

successi conseguiti dal localizzatore rispetto alle simulazioni effettuate. Sulle colonne è riportato il numero di punti utilizzati. Si tratta di un parametro importante perché determina il tempo richiesto da ciascuna iterazione del localizzatore. Le misure effettuate indicano che se la durata di una iterazione con 1200 campioni è 1, con 2000 campioni il tempo necessario è mediamente 2.5 superiore e cresce di un fattore 10.2 con 4000 punti.

Il peso computazionale influenza la capacità del localizzatore di convergere nel modo reale. Si pensi ad un robot che si muove a velocità costante: se questa è eccessiva rispetto a quella d'esecuzione di una iterazione, il localizzatore dovrà trattare uno spostamento molto ampio e, quindi, caratterizzato da una maggiore incertezza che non verrà compensata dalle informazioni sensoriali ottenute. Lo scopo del RTPF è proprio quello di ottenere un compromesso fra i tempi di elaborazione e la frequenza con cui le osservazioni vengono ricevute. In simulazione questo problema non è presente in quanto la posizione del robot virtuale viene aggiornata prima di ogni iterazione, il che equivale ad adattarne la velocità a quella del localizzatore. Per questa ragione in simulazione l'aumento del numero dei campioni apporta sempre benefici.

Sulle righe 5.1 della tabella è indicata la traiettoria alla quale i dati sono riferiti. La figura 5.2 mostra i percorsi fatti dal robot nel corso delle simulazioni. Si noti come la traiettoria numero due sia caratterizzata da due curve strette in una zona che provoca grandi variazioni alle misure rilevate dai tre beam del laser. Infatti questo è il caso più critico per il localizzatore realizzato, che è stato in grado di produrre un'ipotesi vicina alla realtà solo in un paio di casi tra tutte le trenta simulazioni complessive.

La tendenza generale è che con l'aumento dei campioni si verifica un miglioramento nelle prestazioni del localizzatore. Questo risultato è in accordo con quanto spiega-

| <i>Mapa: Palazzina 1 Dipartimento di Ingegneria dell'Informazione</i> | | | |
|---|--------------------|--------------------|--------------------|
| | 1200 sample | 2000 sample | 4000 sample |
| <i>Percorso 1</i> | 5 successi su 10 | 7 successi su 10 | 8 successi su 10 |
| <i>Percorso 2</i> | 5 successi su 10 | 4 successi su 10 | 9 successi su 10 |
| <i>Percorso 3</i> | 5 successi su 10 | 8 successi su 10 | 10 successi su 10 |

Tabella 5.1: Percentuale di successi nella localizzazione.

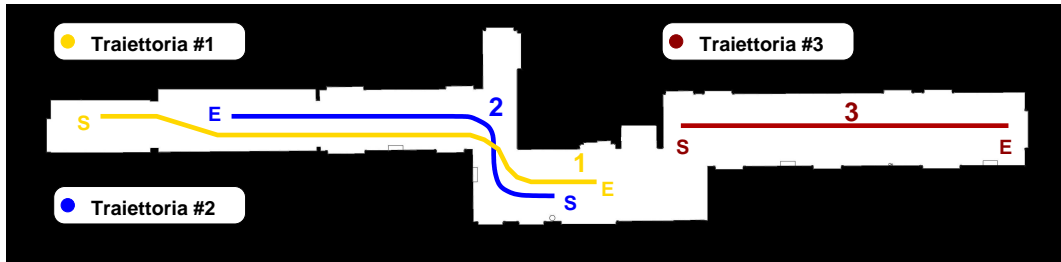


Figura 5.2: I tre percorsi fatti dal robot nel corso delle simulazioni. *S* indica la configurazione iniziale e *E* quella finale.

to in 2.4.1 a proposito della necessità di avere ipotesi vicine alla posizione corretta per la convergenza del filtro.

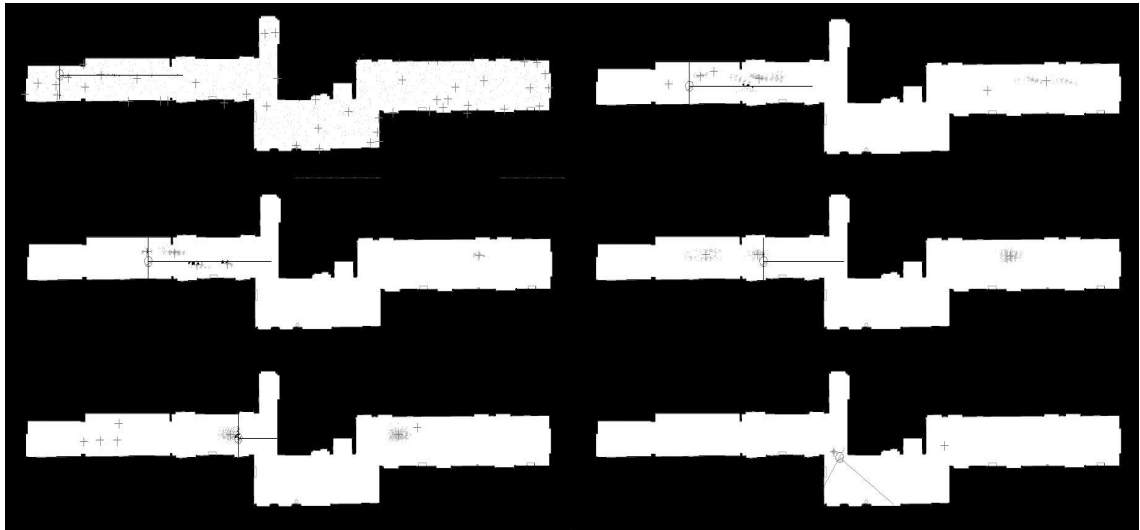


Figura 5.3: Sei iterazioni del localizzatore, risultato di una simulazione con 1200 sample. Si noti la distribuzione uniforme del primo frame e come i campioni tendano ad aggregarsi in distinti cluster dei quali rimane solo quello centrato sulla posizione reale del robot.

5.1.1 Formazioni di artefatti nella distribuzione di campioni

Nel corso dello svolgimento delle simulazioni è talvolta comparsa una distribuzione di campioni lungo linee verticali. La formazione di tali artefatti sembra essere dovuta alla gestione dell'odometria poiché si osserva in presenza di un avanzamento

nullo fra una iterazione e l'altra. La presenza di artefatti non sembra aver compromesso la localizzazione, ma richiede comunque attenzione in quanto, probabilmente, è causato da qualche errore nel codice. I risultati presentati fino ad ora sono stati ottenuti mantenendo il robot virtuale sempre in movimento. La figura 5.4 mostra il fenomeno.

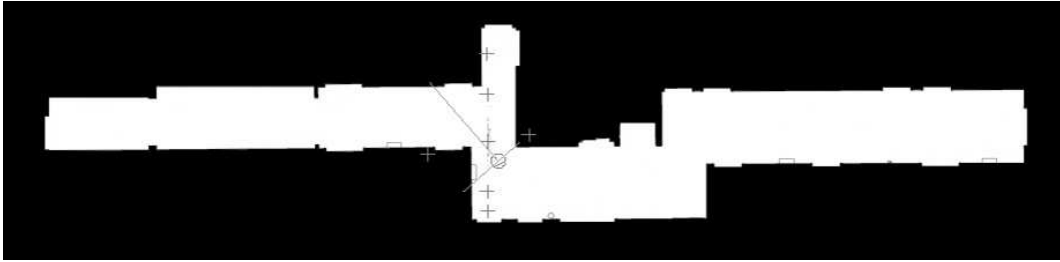


Figura 5.4: Formazione di artefatti nella distribuzione dei campioni.

5.2 Sperimentazione sul Robot Mobile

Lo svolgimento dei test sperimentali richiede l'utilizzo del localizzatore con il robot mobile Nomad. Nel capitolo 4 è stata descritta nei dettagli la piattaforma software che è stata progettata proprio per realizzare questo genere di test.

Il Nomad è stato fatto muovere lungo il corridoio della Palazzina 1 del Dipartimento di Ingegneria dell'Informazione (figura 5.1).

5.2.1 Rilevamento della posizione reale del robot

Per poter valutare i risultati della sperimentazione, occorre confrontare l'output del localizzatore con quelli ottenuti mediante un sistema di misura indipendente e sufficientemente sicuro. Il rilevamento della posizione del robot non può essere svolto internamente al localizzatore e nemmeno sfruttandone le stesse sorgenti eterocettive perchè tale operazione richiederebbe la disposizione nell'ambiente di particolari punti di riferimento che finirebbero con il condizionare i dati sensoriali utilizzati nella localizzazione. Un altro fattore importante da considerare è il tempo: la stima del localizzatore e il risultato della misura di posizione, essendo grandezze ottenute

indipendentemente, sono confrontabili solo se ricavate nello stesso istante temporale (a meno di un errore ragionevole).

Il sistema esterno impiegato per questo scopo è stato costruito utilizzando un framework opensource per la visione artificiale chiamato *ARToolkit*[24].

5.2.1.1 ARToolkit

ARToolkit[24] è stato sviluppato presso l'Università di Washington per applicazioni capaci di sovrapporre modelli grafici tridimensionali alle immagini del mondo reale (Augmented Reality Application). Per poter realizzare applicazioni di questo tipo è necessario avere dei punti di riferimento in uno stream video dei quali sia possibile rilevare la posizione. La libreria di *ARToolkit* mette a disposizione delle primitive che, sfruttando la visione artificiale, consentono di individuare degli elementi predefiniti (i marker) in uno stream video e di calcolarne la posizione relativa rispetto alla telecamera. La figura 5.5 mostra uno schema del funzionamento complessivo di *ARToolkit*.

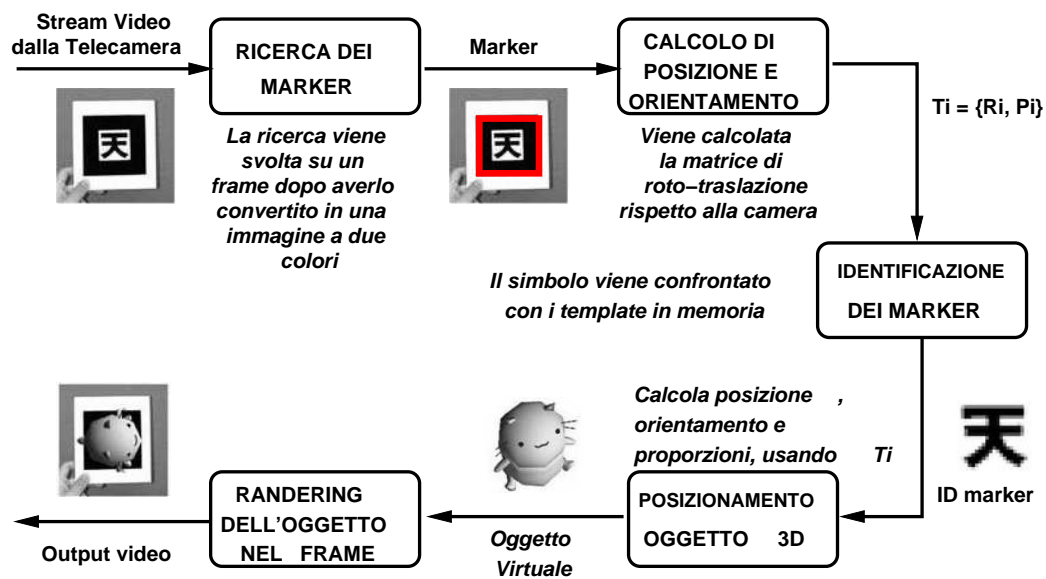


Figura 5.5: Schema del funzionamento di ARToolkit..

I marker previsti da *ARToolkit* sono delle immagini formate da una cornice quadrata al cui interno si trova il simbolo che lo contraddistingue. Maggiori sono le dimensioni del marker e maggiore è la distanza di lettura affidabile alla quale viene

rilevato e riconosciuto con successo. Anche la complessità del simbolo gioca, in tal senso, un ruolo importante: il riconoscimento è più agevole per le figure semplici. In [24] sono riportati alcuni dati che indicano che il rapporto fra la distanza e il lato del marker è circa pari a sei per pattern relativamente semplici come lo sono le lettere dell'alfabeto e i numeri.

Le condizioni di luminosità possono influenzare in senso negativo le capacità di riconoscimento dell'algoritmo: riflessi sulla superficie della carta, bassa intensità e colore possono essere fonti di errore. Ma il principale limite di *ARToolkit* è una forte sensibilità all'occlusione: è infatti sufficiente coprire una piccola porzione del marker per impedirne il riconoscimento.

Le prestazioni possono essere migliorate calibrando la videocamera per far sì che l'algoritmo di riconoscimento tenga conto delle eventuali aberrazioni dell'ottica. Il procedimento viene svolto sfruttando un'applicazione distribuita con il framework e consiste nell'inquadrare da più angolazioni particolari schemi di linee e punti coadiuvando manualmente le operazioni di riconoscimento.

Le applicazioni sviluppate con *ARToolkit* tendono a seguire uno schema fisso formato da una fase di inizializzazione, da un ciclo che ripete indefinitamente le operazioni di figura 5.5 e da una fase di arresto. La fase di inizializzazione comprende il caricamento dei pattern dei marker da riconoscere che sono memorizzati in forma di griglia nelle cui celle sono memorizzati i valori che rappresentano la luminanza associata ad una data posizione. Gli sviluppatori di *ARToolkit* hanno incluso nel pacchetto software anche un applicativo per la memorizzazione di pattern definiti dall'utente.

Sistema di coordinate di ARToolkit

La posizione dei marker è espressa dalla matrice di rototraslazione rispetto al sistema di coordinate della telecamera. Quando un marker viene individuato, ne viene calcolata la distanza utilizzando il dato sulla sua dimensione e dal fattore di forma dell'immagine che lo riassume nel frame, e ne vengono ricostruiti gli assi. L'immagine 5.6 mostra il funzionamento dello sistema di coordinate di *ARToolkit*. Si noti come alle coordinate del marker venga applicata la funzione di distorsione della teleca-

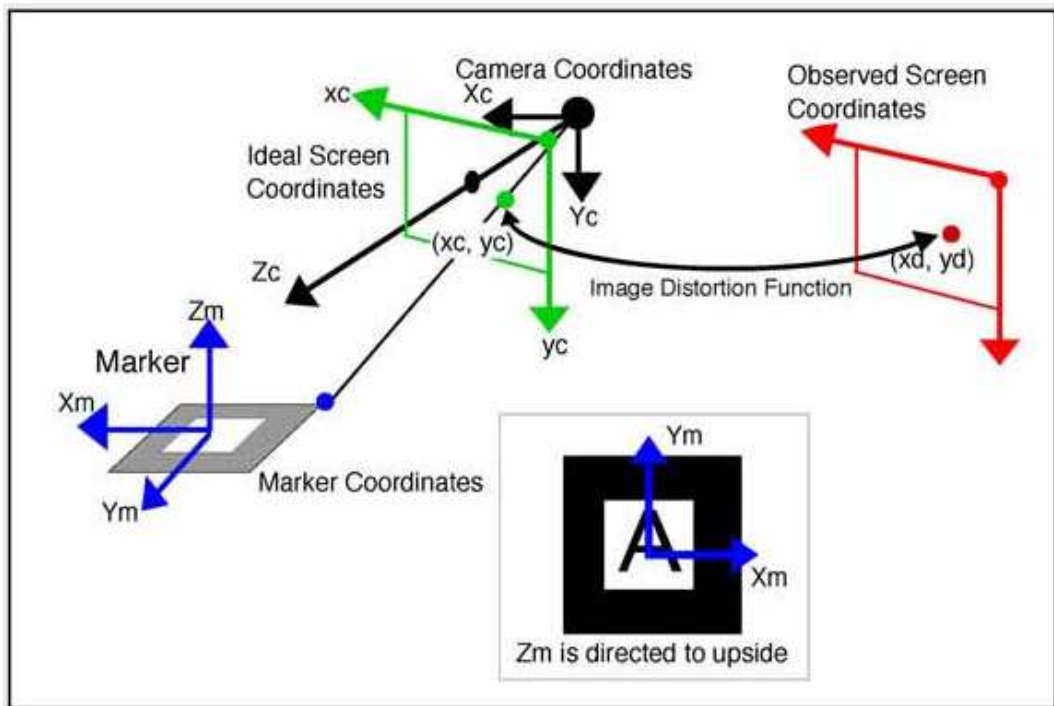


Figura 5.6: Sistema di coordinate si ARToolkit

mera prima della visualizzazione. La funzione di distorsione serve per correggere i difetti visivi indotti dall'aberrazione dell'ottica della camera e può essere calcolata con le funzione di calibrazione di cui si è già parlato in precedenza.

5.2.2 Il sistema di tracking realizzato

Per rilevare la posizione del robot mobile è stata realizzata un'applicazione capace di rilevarne la posizione in tempo reale a partire dalla matrice rototraslazionale fra il marker installato su di esso e quelli di riferimento disposti nell'ambiente. Il meccanismo di rilevamento della posizione si basa sul calcolo della matrice di rototraslazione T fra il marker a bordo del robot e quello di riferimento. Di quest'ultimo sono note posizione ed orientamento rispetto alla terna di riferimento ambientale. Con questi dati è possibile capire dove si trovi il robot rispetto all'origine. *ARToolkit* espone la funzione `arGetTransMat` che restituisce la matrice T_{cam}^{mark} per assi

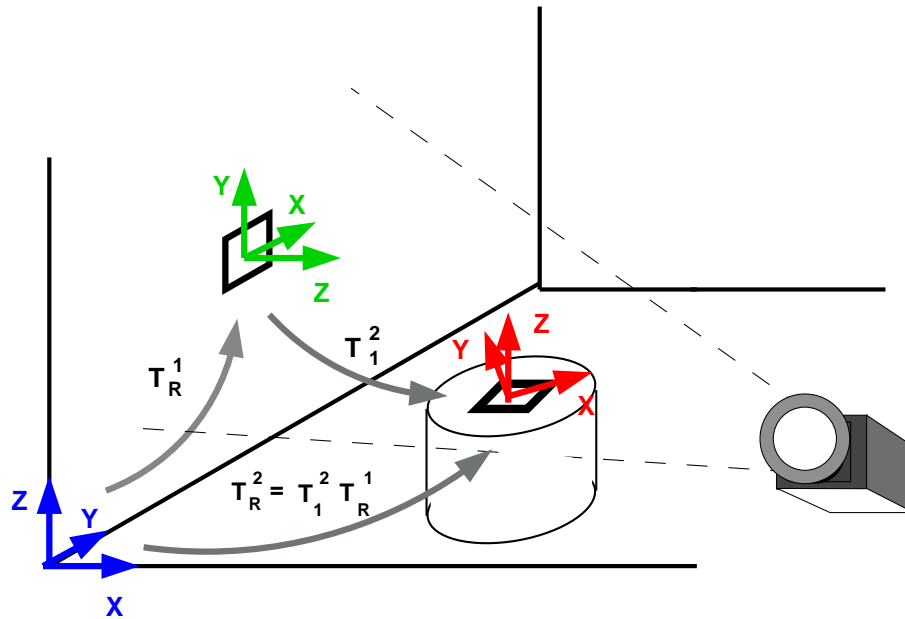


Figura 5.7: L'applicazione calcola T_1^2 e successivamente posizione e orientamento del marker sul robot sfruttando i dati riguardanti il marker appeso al muro

mobili. Per ottenere la matrice fra due marker, è sufficiente sfruttare le proprietà delle matrici T [25]:

$$T_{mark_2}^{mark_1} = T_{mark_2}^{cam} (T_{mark_1}^{cam})^{-1} \quad (5.1)$$

Dove la matrice di roto-traslazione ha la forma generale:

$$\mathbf{T}_A^B = \left(\begin{array}{c|c} R_A^B & p^A \\ \hline 0 & 1 \end{array} \right) \quad (5.2)$$

R_A^B rappresenta la matrice di rotazione fra le terne A e B e p^A il vettore che ne congiunge le origini espresso secondo il riferimento A . α , β e γ sono gli angoli delle tre rotazioni intorno agli assi X , Y e Z . In particolare la 5.3 mostra l'espressione della matrice di rotazione per assi mobili e la 5.4 quella per assi fissi [25]. *Artoolkit*, in particolare, utilizza la notazione per assi mobili, quindi da ora in avanti quando si parlerà di matrici di rotazione (roto-traslazione), sarà sottintesa la notazione 'per

assi mobili'.

$$\mathbf{R}_A^B = \begin{pmatrix} C_\gamma C_\beta & -C_\beta S_\gamma & S_\gamma \\ C_\gamma S_\beta S_\alpha + S_\gamma C_\alpha & -S_\gamma S_\beta S_\alpha + C_\gamma C_\alpha & -C_\gamma S_\alpha \\ -C_\gamma S_\beta C_\alpha + S_\gamma S_\alpha & S_\gamma S_\beta C_\alpha + C_\gamma S_\alpha & C_\gamma C_\alpha \end{pmatrix} \quad (5.3)$$

$$\mathbf{R}_A^B = \begin{pmatrix} C_\gamma C_\beta & -C_\alpha S_\gamma + S_\alpha S_\beta S_\gamma & S_\alpha S_\gamma + S_\beta S_\alpha S_\gamma \\ S_\gamma C_\beta & -C_\alpha C_\gamma + S_\alpha S_\beta S_\gamma & -S_\alpha S_\gamma + S_\beta S_\alpha S_\gamma \\ -S_\beta & C_\beta S_\alpha & C_\beta C_\alpha \end{pmatrix} \quad (5.4)$$

$$p^A = [x^A, y^A, z^A] \quad (5.5)$$

La figura 5.7 mostra un metodo generale per ricavare la matrice T_R^2 dalla quale sarebbe possibile ottenere i dati cercati sulla posizione e l'orientamento del robot. Tuttavia, scegliendo opportunamente la terna di riferimento e il posizionamento dei marker, è possibile raggiungere una situazione particolarmente comoda per lo svolgimento dei calcoli e soprattutto tale da consentire l'unicità della soluzione del calcolo degli angoli.

Il marker sul robot giace su un piano parallelo al piano X-Y della terna di riferimento e l'asse Y è concorde con il senso di marcia. Inoltre, il robot si muove su un piano e quindi la sua posizione è un vettore bidimensionale per cui la coordinata Z non è rilevante. Il calcolo della posizione è semplice, in quanto la matrice di rototraslazione fornisce la posizione relativa fra i due marker e quindi si ottiene la posizione del robot per semplice differenza del vettore che identifica la posizione del marker di riferimento con quello della matrice T .

Più complesso si può rivelare il calcolo dell'orientamento in quanto le funzioni trigonometriche inverse non danno risultati univoci senza gli opportuni vincoli. Fortunatamente è possibile disporre i marker in modo tale da ricadere in alcuni casi notevoli:

- *Parallelamente al piano X-Y della terna di riferimento.* Questa è la disposizione che semplifica maggiormente i calcoli in quanto la terna del robot si sovrappone a quella dei marker ambientali per una semplice rotazione intorno

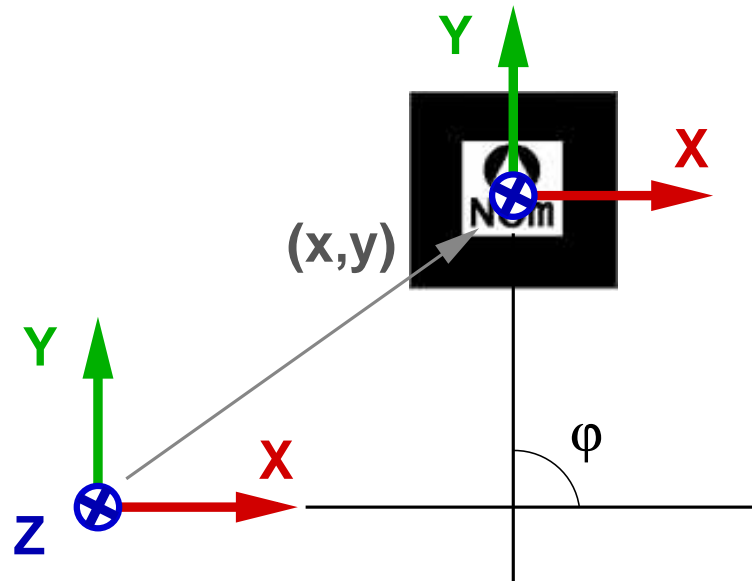


Figura 5.8: Posizione e orientamento del marker del robot.

all'asse Z di un angolo γ . La matrice di è molto semplice.

$$\mathbf{R}_A^B = \begin{pmatrix} \cos \gamma & \sin \gamma & \sin \gamma \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & \cos \gamma \end{pmatrix} \quad (5.6)$$

L'orientamento può essere ottenuto da γ . Purtroppo questa disposizione si rivela poco pratica in quanto i marker finirebbero con l'essere disposti sul pavimento e questo renderebbe difficile l'inquadratura con la telecamera. Inoltre sarebbero soggetti a macchiarsi e questo potrebbe comprometterne il riconoscimento.

- *Parallelamente al piano X - Z della terna di riferimento.* In questo caso l'orientamento complessivo si compone di una rotazione intorno a Z di ampiezza γ e una intorno a X di $\frac{\pi}{2}$.

$$\mathbf{R}_A^B = \begin{pmatrix} \cos \gamma & -\sin \gamma & \sin \gamma \\ 0 & 0 & -\cos \gamma \\ \sin \gamma & \cos \gamma & 0 \end{pmatrix} \quad (5.7)$$

Con questa disposizione il calcolo di β è semplice sebbene non univoco ma non è semplice disporre i marker se non si hanno a disposizione dei cartelli da inchiodare alle pareti o degli striscioni da appendere al soffitto.

- *Disposizione sulle pareti, parallelamente al piano Y-Z della terna di riferimento.* Questa è la disposizione scelta. Benché sia la più complicata da gestire analiticamente è quella che crea meno difficoltà ad essere messa in pratica. I marker vengono disposti sulle pareti per essere inquadrati al passaggio del robot nei loro pressi. Con una rotazione intorno a Z pari a γ e una di $\frac{\pi}{2}$ intorno a X , la matrice di rotazione è la seguente:

$$\mathbf{R}_A^B = \begin{pmatrix} \cos \gamma & -\sin \gamma & \sin \gamma \\ 0 & 0 & -\cos \gamma \\ \sin \gamma & \cos \gamma & 0 \end{pmatrix} \quad (5.8)$$

Ancora una volta è possibile ricavare l'angolo γ agevolmente per poi risalire all'orientamento. La maggiore difficoltà nella gestione di questa disposizione deriva dalla necessità di trattare diversamente i marker che sono interni alla traiettoria da quelli che sono esterni. La figura 5.9 mostra un'ipotetica disposizione dei marker sulle pareti di un corridoio chiuso. Si prendano come esempio, i marker 1) e 7). Supponendo che lo spostamento avvenga lungo la linea tratteggiata in senso orario, il loro orientamento rispetto al riferimento è identico ma devono dare informazioni diverse riguardo il verso del movimento del robot. Quando il robot è nei pressi di 1), la direzione rilevata deve dare un angolo di $\frac{\pi}{2}$, vice versa sarà di $-\frac{\pi}{2}$ al passaggio nella zona di 7). In conclusione, è necessario tenere conto nel calcolo dell'orientamento del robot del lato della traiettoria in cui è piazzato il marker.

5.2.2.1 Calcolo della posizione e dell'orientamento del robot

Quando l'applicazione individua sullo stesso frame il marker installato sul robot e uno o più di quelli disposti nell'ambiente mediante l'utilizzo della primitiva `arGetTransMat` ricava tutte le matrici di rototraslazione rispetto alla telecamera. Quella corrispondente al marker del robot viene invertita e poi moltiplicata con quella del marker più vicino, ottenendo la matrice che esprime la rototraslazione fra i due. A questo

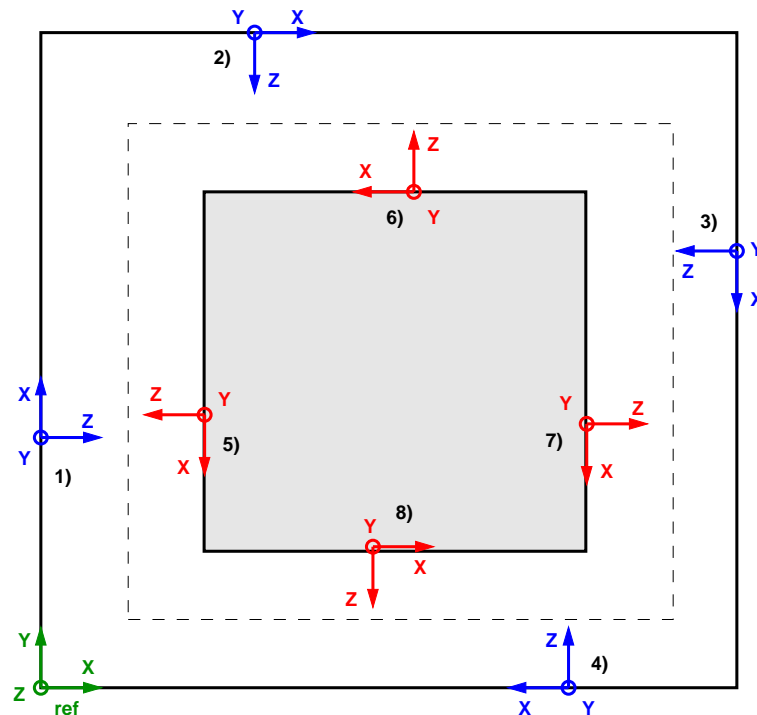


Figura 5.9: Ipotetico posizionamento dei marker in un corridoio chiuso.

punto, sfruttando le informazioni su posizione e orientamento del marker di riferimento rispetto alla terna di origine, viene applicato l'algoritmo 10 per il calcolo della posizione del robot.

5.2.2.2 Interfaccia dell'applicazione di tracking

L'interfaccia utente è costituita da una finestra che mostra l'output video del software: lo stream video originale, proveniente dalla telecamera, con la sovrapposizione di un modello 3D al centro di ogni marker riconosciuto che ne rappresenta gli assi. Quando viene svolto il calcolo della posizione del robot, il risultato viene stampato a video e scritto su un file. I dati riguardanti la posizione dei marker di riferimento disposti nell'ambiente, vengono caricati da un file di testo. La figura 5.10 mostra un frame proveniente dalla telecamera prima e dopo l'elaborazione da parte del tracker. Nell'immagine 5.10.B si nota come intorno ai marker sia stato disegnato un contorno rosso e nella posizione di quelli con un pattern noto al software è stata disegnata anche la terna di riferimento orientata in modo concorde al marker.

Algoritmo 10 Calcolo della posizione del marker posto sul robot rispetto all'origine

Require: matrice di rototraslazione T fra il marker di riferimento e il marker del robot; posizione e orientamento nello spazio del marker di riferimento rispetto all'origine.

- 1: $x_r = x_m - T_{1,4}$
 $y_r = y_m - T_{2,4}$
 - 2: $\varphi = \phi_m + \frac{\pi}{2} + \text{sign}(T_{1,1})[\arcsin T_{1,3}]$
 - 3: **if** il marker è interno **then**
 - 4: $\varphi + = \pi$
 - 5: **end if**
 - 6: **if** $\varphi < 0$ **then**
 - 7: $\varphi + = 2\pi$
 - 8: **end if**
 - 9: **return** $[x, y, \varphi]$
-

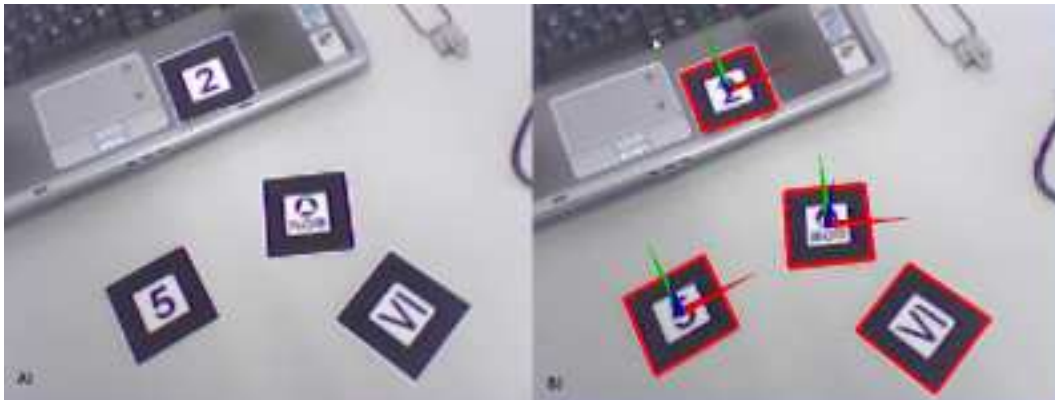
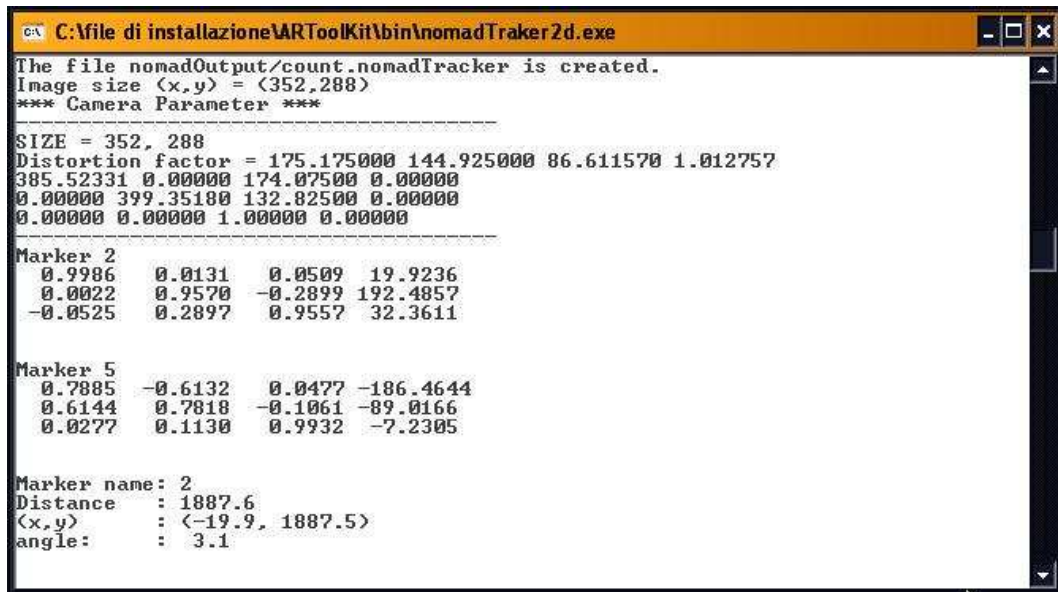


Figura 5.10: A) frame video; B) frame elaborato dal tracker.

L'output generato dal software è indirizzato sia sul video (figura 5.11) che su un file di testo con un timestamp associato ad ogni rilevamento. Il formato è quello di una matrice di rototraslazione seguita dal rilevamento delle coordinate assolute del riferimento rispetto al più vicino. Nell'esempio di figura 5.10 il marker che rappresenta il robot è *Nom* e il più vicino è il 2.

5.2.2.3 Problema della sincronizzazione

Per la valutazione del localizzatore a bordo del robot è necessario confrontare la posizione stimata dal localizzatore con quella rilevata mediante l'applicazione di



```
C:\file di installazione\ARToolKit\bin\nomadTracker2d.exe
The file nomadOutput/count.nomadTracker is created.
Image size (x,y) = (352,288)
*** Camera Parameter ***
-----
SIZE = 352, 288
Distortion factor = 175.175000 144.925000 86.611570 1.012757
385.52331 0.000000 174.075000 0.000000
0.000000 399.351800 132.825000 0.000000
0.000000 0.000000 1.000000 0.000000
-----
Marker 2
0.9986    0.0131    0.0509    19.9236
0.0022    0.9570   -0.2899    192.4857
-0.0525   0.2897    0.9557    32.3611

Marker 5
0.7885   -0.6132    0.0477   -186.4644
0.6144    0.7818   -0.1061   -89.0166
0.0277    0.1130    0.9932   -7.2305

Marker name: 2
Distance   : 1887.6
(x,y)     : (-19.9, 1887.5)
angle:    : 3.1
```

Figura 5.11: Output video del tracker.

tracking. Pertanto è necessario aggiungere ai dati prodotti da entrambe le applicazioni un timestamp che consenta di effettuare un confronto.

L'informazione prodotta dal componente localizzatore è simile a quella fornita dal simulatore: un log file che riporta le coordinate dei sample e una mappa che ne mostra graficamente la posizione. Ciò che manca rispetto alla simulazione è il calcolo dell'errore del localizzatore al termine di ogni iterazione, in quanto tale calcolo richiede la conoscenza della posizione reale del robot in quel preciso istante. Il software di tracking deve fornire tale informazione ma affinché ciò sia possibile è necessario sincronizzarlo con il componente di localizzazione.

La soluzione scelta per realizzare la sincronizzazione è particolarmente semplice e consiste nell'utilizzare dei segnali inviati tramite socket. Essa, inoltre coinvolge anche il controllo del movimento in modo che il robot cominci a muoversi solo quando effettivamente necessario. Il localizzatore funziona da server e attende un messaggio dal tracker e dal componente di navigazione, i quali dopo l'invio del loro pacchetto si mettono in attesa di un segnale di conferma che il server invia solo quando entrambi lo hanno contattato. La figura 5.12 mostra il diagramma di funzionamento.

Una possibile evoluzione di questo meccanismo di sincronizzazione consiste

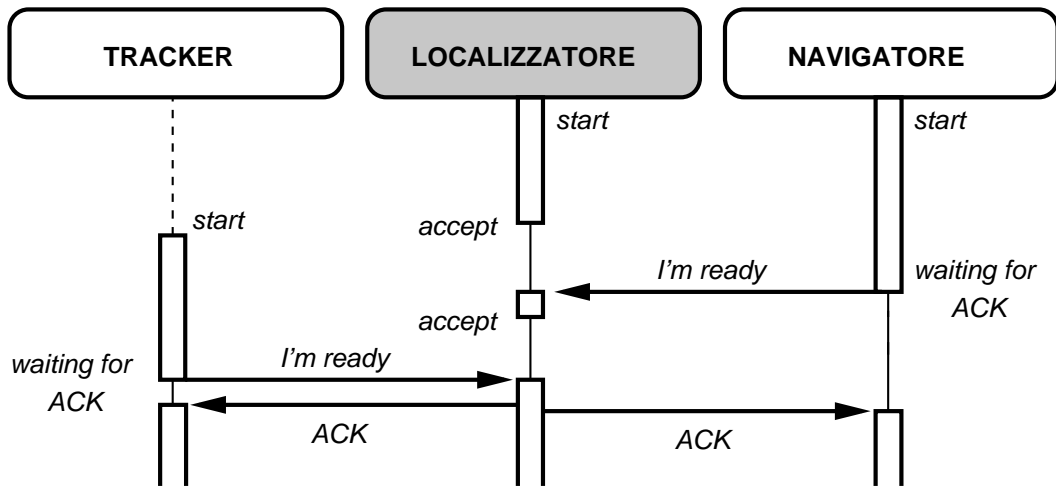


Figura 5.12: Sequenza di messaggi fra i tre processi: localizzatore, tracker e navigatore.

nell'includere una comunicazione fra localizzatore e tracker di tipo *query-send*, in modo che i dati sulla posizione del robot vengano richiesti al termine di ogni iterazione. Il principale vantaggio di questa soluzione è un maggior grado di sincronismo fra localizzatore e strumento di misura in quanto non vengono generati dei timestamp da confrontare a esecuzione ultimata, ma i dati sulla posizione del robot vengono rilevati su richiesta del localizzatore e immediatamente utilizzati per valutare l'errore commesso al termine di ogni iterazione. Nel caso del sistema realizzato invece, è necessario elaborare i dati prodotti dal localizzatore e confrontarli con i rilevamenti del tracker sulla base dei timestamp solo ad esecuzione ultimata. In figura 5.13 è mostrato un diagramma che descrive lo scambio di messaggi relativo all'approccio appena descritto.

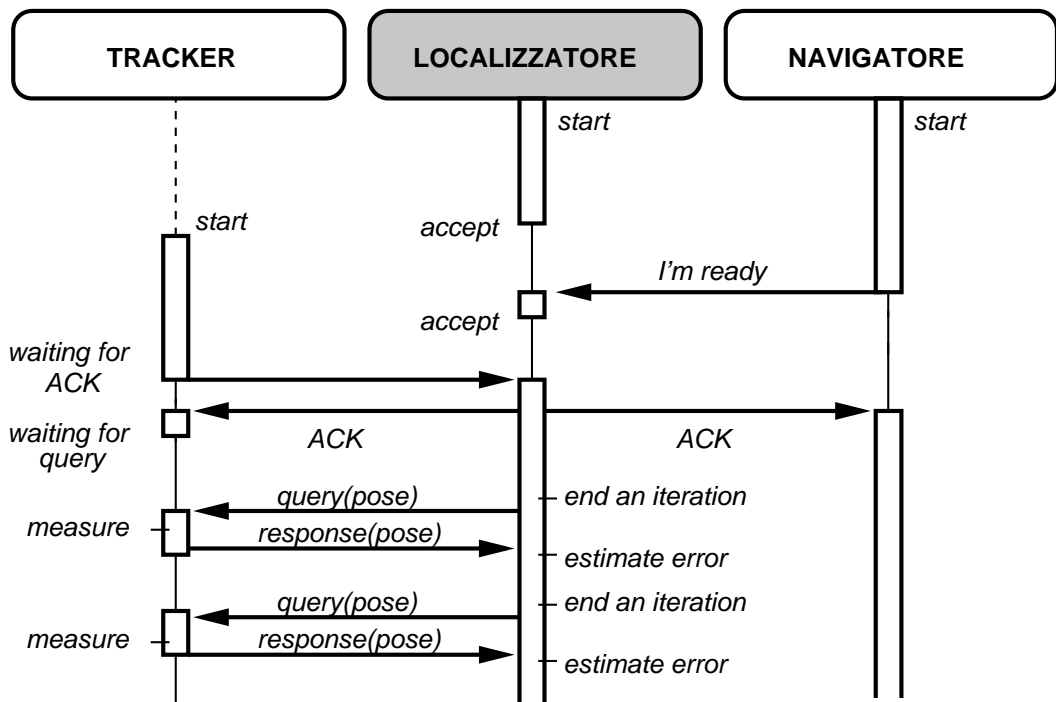


Figura 5.13: Sincronizzazione e paradigma query-send fra il localizzatore e il tracker.

Capitolo 6

Conclusioni

In questo lavoro di tesi è stato esteso un sistema di localizzazione basato su filtri particellari integrando in esso un algoritmo di clustering che viene applicato ai set di campioni generati ad ogni iterazione. Tale estensione è concepita per migliorare la convergenza del localizzatore. L'algoritmo di clustering si è dimostrato capace di assolvere allo scopo previsto consentendo al localizzatore di individuare gli agglomerati di particelle nel set di campioni e quindi di estrarre agevolmente informazioni sugli stati a più elevata verosimiglianza.

La verifica in simulazione eseguita sul localizzatore integrato dall'algoritmo di clustering, ha evidenziato un comportamento soddisfacente nella maggior parte dei casi e ha consentito di portare alla luce alcune situazioni critiche per la convergenza.

È stata, inoltre, progettata una piattaforma software per l'integrazione del localizzatore su robot mobile basata su SmartSoft e dotata di una architettura distribuita tale da consentire l'esecuzione dell'algoritmo di localizzazione su un elaboratore remoto. La realizzazione della piattaforma ha richiesto una progettazione su più livelli, richiedendo anche fasi di analisi e sviluppo dei driver per il controllo dei dispositivi hardware. L'architettura realizzata comprende un server per la gestione della piattaforma robotica che abbia accesso all'apparato sensoriale e agli attuatori dei motori; una unità che incapsula il localizzatore che elabora i dati provenienti dai sensori; un sistema di controllo del movimento del robot che sia in grado di inviare i comandi necessari per la navigazione.

Pur non essendo stato possibile completare le verifiche sperimentali sul robot reale,

sono stati realizzati tutti i componenti di controllo e monitoraggio necessari per i test. Tali test potranno essere completati una volta superati i problemi di interfacciamento del laser scanner con SmartSoft. Il lavoro svolto in questo progetto di tesi fornisce alcuni importanti spunti per sviluppi futuri:

- Le simulazioni svolte sul localizzatore hanno portato alla luce alcune situazioni critiche. L'ulteriore sviluppo della libreria deve essere rivolto all'irrobustimento dell'algoritmo di localizzazione tenendo conto di tali situazioni.
- Implementazione dell'algoritmo di clustering alternativo basato su griglia descritto nel capitolo 3 da sostituire eventualmente a quello realizzato. Tale algoritmo potrebbe rivelarsi più efficiente in quanto, sotto opportune condizioni, ha una complessità inferiore di quello integrato nel localizzatore..
- Il driver del laser scanner utilizzato da SmartSoft si è dimostrato inefficiente nell'impiego con il convertitore USB-RS422 e pertanto potrebbe essere sostituito con quello sviluppato dal RIMLab e modificato nel corso di questo lavoro di tesi. In questo modo anche con SmartSoft diverrebbe possibile beneficiare di tutta la banda offerta dallo standard veloce RS-422.
- Revisione del sistema di acquisizione della posizione del robot in modo che le misurazioni avvengano solo al bisogno e in modo sincronizzato con il termine di ognuna delle iterazioni del localizzatore. In queste condizioni, la sincronizzazione fra misure e risultati di localizzazione sarebbe sempre garantita.

Bibliografia

- [1] W. Burgard S. Thrun, D. Fox and F. Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence Magazine*, 2001.
- [2] M. Montemerlo. Fastslam: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association. PhD Thesis, Carnegie Mellon University, Pittsburgh, 2003.
- [3] D. Fox. Adapting the Sample Size in Particle Filters through KLD-Sampling. *International Journal of Robotics Research*, 22(12), 2003.
- [4] D. Fox. Real-Time Particle Filters. *Proceedings of the IEEE*, 92(2), 2004.
- [5] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and Thrun S. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.
- [6] A. Doucet, N. de Freitas, K. Murphy, and S. Russel. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2000.
- [7] A. Doucet, S.J. Godsill, and C. Andrieu. On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10(3), 2000.
- [8] J. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443), 1998.
- [9] W. Poland and R. Shachter. Mixture of gaussians and minimum relative entropy techniques for modeling continuous uncertainties. *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 1993.
- [10] J.N. Williamson E.T. Milstein, A. Sánchez. Robust Global Localization Using Clustered Particle Filtering.
- [11] D. Lodi Rizzini. Progettazione di una libreria per la localizzazione e fusione sensoriale, basata su filtri particellari. Tesi di Laurea Specialistica in Ingegneria Informatica, Università degli Studi di Parma, 2005.
- [12] Anil K. Jain, Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.
- [13] Osmar Zaiane. Data Clustering.
www.cs.ualberta.ca/~Ezaiane/courses/cmput690/slides/Chapter8.
- [14] Unknown. Grid-Based Cluster Algorithm.
<http://www.cs.unc.edu/Courses/comp290-90-f03/clustering3.pdf>.
- [15] Salvatore Orlando. Gaussian Mix Model for Clustering.
<http://www.stat.psu.edu/~jiali/course/stat597e/notes2/mix.pdf>.

- [16] Player/Stage. <http://playerstage.sourceforge.net>.
- [17] F. Monica. Progettazione di una Architettura Modulare, Aperta ed in Tempo Reale per un Robot Mobile. Tesi di Laurea Magistrale in Ingegneria Informatica, Università degli Studi di Parma, 2003.
- [18] Sick inc. www.sick.com/home/en.html.
- [19] orocos-smartsoft home page. <http://www.rz.fh-ulm.de/~cschlege/orocos/>.
- [20] Corba home page. www.corba.org/.
- [21] Iiop spec. iiop-net.sourceforge.net/.
- [22] Corba name service. edocs.bea.com/wle/naming/over.htm.
- [23] F. Pavino and L. Tiso. Realizzazione di una Applicazione Software In Grado di Pilotare un Laser Scan. Tesina per il Corso di Sistemi in Tempo reale, 2005.
- [24] Artoolkit home page. www.hitl.washington.edu/artoolkit/index.html.
- [25] C. Guarino Lo Bianco. *Cinematica dei Manipolatori*. Pitagora Editrice Bologna, 2003.
- [26] Athanasios Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 1965.
- [27] Data Clustering Tutorial.
http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html.
- [28] Salvatore Orlando. Data Clustering.
<http://www.dsi.unive.it/~dm/Cluster.pdf>.
- [29] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Pattern Applied*. Addison-Wesley, 2001.
- [30] Boost. <http://www.boost.org>.
- [31] Datasheet sick lms 200.
www.robosoft.fr/SHEET/02Local/1004SickLMS200/SickLMS200.html.
- [32] Ace-tao home page. <http://www.cs.wustl.edu/~schmidt/TAO.html>.
- [33] E. Gamma, R. Helm, R. Jhonson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [34] N.M. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Addison-Wesley, 1999.