UNIVERSITA' DEGLI STUDI DI ROMA
"TOR VERGATA"

FACOLTA' DI INGEGNERIA

Tesi di Laurea in Ingegneria dell'Automazione

"Tecniche di esplorazione e pianificazione del moto per il problema dello SLAM"

Relatore
Ing. F. Martinelli

Laureando
Francesco A. Marino

Anno Accademico 2003/2004

Indice

Capitolo 1

Introduzione

Negli ultimi anni, la robotica mobile ha rivestito un interesse sempre maggiore nell'ambito dell'automazione. I motivi principali risiedono nell'enorme potenzialità che possiedono queste macchine a portare a termine processi che sono considerati difficili per l'uomo o che vengono svolti in ambienti ostili, come per esempio industrie nucleari. Un ulteriore campo di applicazione della robotica mobile è rappresentato da tutti quei lavori che prevedono una sorta di ripetitività e dipendenza da parte dell'uomo.
Una competenza necessaria che un robot deve avere è la capacità di muoversi evitando collisioni con eventuali ostacoli presenti nell'ambiente. Ciò implica che il robot deve avere una conoscenza dettagliata del dominio. Tuttavia in alcune applicazioni, come per esempio l'esplorazione extra-terrestre, ciò non può essere richiesto risultando necessario perciò l'intervento umano. In quasi tutte le applicazioni di questo tipo infatti i robot vengono comandati via radio da un operatore umano.
É proprio in questo contesto che si inserisce il *problema dello SLAM* (Localizzazione e acquisizione simultanea della mappa). Per navigare in un ambiente sconosciuto un robot autonomo ha necessità di costruire la mappa e contemporaneamente deve mantenere una corretta stima della propria posizione. Il problema risulta molto complesso a causa della natura circolare del problema. Infatti per esplorare e ricostruire la mappa di un ambiente sconosciuto, deve conoscere la propria posizione, ma contemporaneamente per una buona localizzazione, il robot richiede conoscenze specifiche del dominio sul quale opera. Legato al problema dell'acquisizione della mappa e della localizzazione vi è anche il problema dell'esplorazione dell'ambiente. L'acquisizione della mappa infatti è un processo che può avvenire solo grazie al fatto che il robot è in grado di esplorare correttamente e in modo esaustivo l'ambiente e viceversa essa è possibile solo se si riesce a distinguere con precisione una zona nota o ancora da esplorare. Inoltre al fine di una efficace

esplorazione del dominio è necessario che il robot si localizzi in modo corretto in quanto deve conoscere con una buona precisione la sua posizione. D'altro canto, una corretta localizzazione richiede buone tecniche di esplorazione al fine di acquisire il maggior numero di informazioni possibile. Già da una prima analisi, quindi, risulta evidente che i concetti principali sui quali si fonda il problema dello SLAM sono così intrinsecamente legati da renderlo un problema abbastanza complesso la cui soluzione prevede la messa a punto di tecniche tali da rendere il robot il più possibile autonomo e indipendente da qualsiasi tipo di assunzione o da una conoscenza innata. In questo lavoro di tesi vengono individuate alcune tecniche per l'auto-localizzazione e per l'esplorazione atte proprio a evitare qualsiasi tipo di conoscenza a priori sul dominio e sul modello del sistema in considerazione; vengono inoltre messe a punto tecniche basate su un comportamento reattivo e sull'apprendimento di metodologie di classificazione di oggetti caratterizzanti l'ambiente sulla sola base delle informazioni sensoriali.
Le uniche assunzioni che vengono fatte sono riguardo le dimensioni dell'ambiente nel quale opera il robot. Si considera infatti un ambiente di medie dimensioni (in particolare tutti gli esperimenti sono condotti nei locali del Dipartimento di Informatica e Sistemi di Produzione presso la facoltà di ingegneria di Tor Vergata). Tale assunzione è necessaria a causa dei forti limiti che si hanno nell'utilizzo dei sensori del robot Nomad 150 usato negli esperimenti.

Il presente lavoro di tesi può essere schematizzato nel modo seguente: per quanto riguarda la localizzazione vengono utilizzati **algoritmi genetici** come filtro di stima della posizione del robot. Una descrizione dettagliata di tale tecnica viene presentata nel capitolo 4. Nel capitolo 3 viene offerta una panoramica sulle altre tecniche presenti in letteratura. Il capitolo 2 descrive il sistema hardware e software del robot utilizzato negli esperimenti, mentre nel capitolo 5 vengono descritte e analizzate tutte le metodologia ad alto e a basso livello utilizzate per l'esplorazione. Infine i capitoli 6 e 7 sono dedicati alla descrizione di quelli che sono stati i risultati degli esperimenti condotti e le conclusioni che ne derivano.

Capitolo 2

Nomad 150

Il lavoro simulativo e sperimentale è stato condotto con l'ausilio di un robot mobile *Nomad 150* , realizzato dalla Nomadic Software Inc. (vedi fig. 2.1).

Il sistema consta del robot stesso e delle librerie software necessarie per il suo funzionamento, oltre a un simulatore nel quale è possibile ricreare vari ambienti e situazioni.

Figura 2.1: Il robot *Nomad 150*

2.1 Hardware

Dal punto di vista meccanico, il *Nomad 150* è dotato di un sistema di tre ruote simmetriche azionate da due motori, uno per spostamenti traslazionali e uno per quelli di rotazione. Data la loro simmetria, le ruote non sono indipendenti ma si muovono insieme; costituiscono però un *zero gyro-radius*

system in grado di far ruotare il robot su sé stesso senza alcuna traslazione. Il sistema è *anolonomo*, in quanto presenta dei vincoli non integrabili nelle velocità. Per chiarire meglio tale concetto, prendiamo come esempio un sistema anolonomo come quello di una macchina: se si volesse parcheggiare non sarebbe possibile affiancarsi al posto libero e spostarsi ortogonalmente ad esso, bisognerebbe eseguire una serie di rotazioni concorrentemente a delle traslazioni. Un sistema olonomo permette di evitare tutto ciò, in quanto è in grado di portarsi in qualunque punto dello spazio di lavoro con uno spostamento unico; si pensi ad esempio ad un robot che abbia due motori traslazionali ortogonali oppure delle sfere.

Il controllo dei due motori delle ruote, più quello di un terzo per la rotazione della torretta, è affidato ad un sistema Motorola MC68008/ASIC. I due motori rotazionali così controllati hanno una velocità che va da 0 a 45 gradi/sec, mentre quello traslazionale va da 0 a 20 inch/sec ($1inch \approx 2.5cm$).

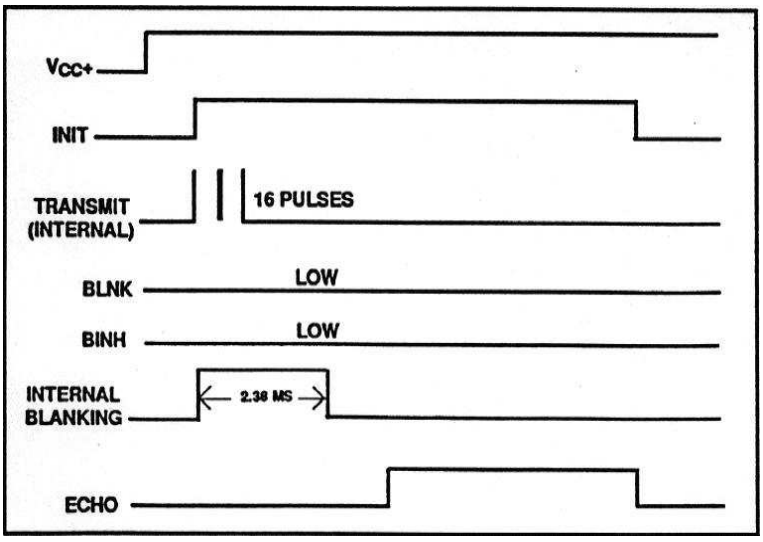

Figura 2.2: Diagramma temporale dei segnali logici di un trasduttore *Polaroid 6500*

Il *Nomad 150* è equipaggiato con tre sistemi di sensori: ultrasuoni, tattile e odometrico. Il *Sensus 200 Sonar System* consta di 16 sonars posti sulla torretta del robot a distanza di $22,5°$ l'uno dall'altro, in grado di rilevare eventuali ostacoli nel range [17,255] inch[1] con una precisione dell'1% sul range totale.

[1]Negli esperimenti condotti i sonars sono riusciti a rilevare con precisione oggetti fino a 6 inch.

I sonar sono dei trasduttori *Polaroid 6500* che emettono 16 impulsi a 49,4KHz, alimentati con una tensione pari a circa 5V, e dotati di un sistema che evita che il ronzio stesso del sonar all'atto dell'emisione dell'impulso sia captato come segnale di ritorno; per questo motivo nel primi millisecondi viene disattivato il circuito di *echo* (vedi fig. 2.2) e la distanza minima di rilevamento è di 6 inch. Il secondo sistema, il *Sensus 100 Sonar System*, è composto da 20 sensori di contatto (detti *bumper* o *switch*) disposti su due corone circolari (come mostrato in figura 2.3), ognuna con 10 di essi, e posti in modo asimmetrico, disposti quindi a 18° l'uno dall'altro. Ciascun sensore ha una soglia di sensibilità pari a circa 226 grammi.

Figura 2.3: I bumper nel robot *Nomad 150*

Entrambi i sistemi sono gestiti da un controller *Motorola 68HC11* (fig. 2.4), che dispone di una CPU programmata per lavorare su dati ad 8 bit, grazie a due accumulatori che si possono anche unire in uno unico a 16 bit; inoltre il set di istruzioni comprende operazioni logico-aritmetiche per interi sia a 8 che a 16 bit. Dal punto di vista energetico il microcontroller in questione richiede risorse minime (meno di 10mA) e supporta delle primitive STOP e WAIT di risparmio energetico. Infine, il sistema odometrico è composto da encoder per l'acquisizione dei dati sul movimento dei tre motori.

Il *Nomad 150* viene alimentato per mezzo di tre batterie *YUASA NP 12-12* che erogano una tensione di 12V ed una corrente di 12Ah, con una durata massima di circa 3 ore.

Il robot non è dotato di alcun computer di bordo, è necessario dunque interfacciarlo con un PC tramite due possibili modalità: cavo seriale o collegamento wireless. Nel primo caso il problema principale è la necessita di dover porre il PC sulla torretta, senza la possibilità di poterlo collegare alla

rete elettrica e quindi con una durata massima data dall'autonomia delle sue batterie. Nel secondo caso invece, si fa uso di un radio-modem *Mercury-EN* prodotto dalla *Nomadic Communications, Inc.*, che si collega alla porta seriale del PC e lavora ad una frequenza di 2.4GHz con una connessione TCP. La limitazione in questo caso è data dalla portata massima del segnale (valutata in circa 8 metri) e dalla perdita di pacchetti riscontrata durante i test.

Figura 2.4: Il microcontroller *Motorola 68HC11*

2.2 Software

Il *Nomad 150* ha una libreria di funzioni scritte in C (ma utilizzabili anche in C++) che forniscono un'architettura di sistema a basso livello, permettendo di eseguire alcuni semplici movimenti di traslazione, rotazione, nonché di gestione dei sensori.

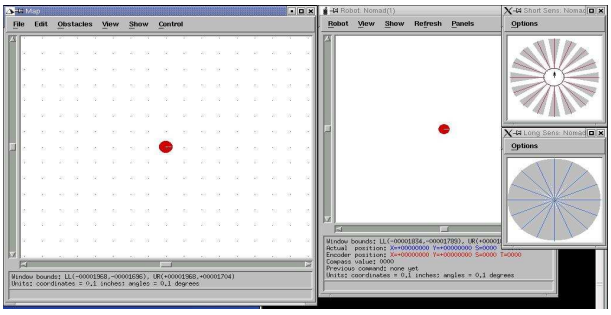

Figura 2.5: L'ambiente simulativo *Nserver*

In aggiunta viene fornito anche un programma, *Nserver* (mostrato in fig. 2.5), che ha due funzionalità: permette di vedere a schermo i dati che il

robot ottiene dall'ambiente, quali la sua posizione e le rilevazioni dei sensori, e agisce anche da simulatore per testare il robot senza doverlo far muovere realmente.

Esistono due possibili modi di eseguire un programma sul robot:

1. utilizzare un terminale di emulazione (*minicom* sotto Linux o *hyperterminal* sotto Windows) per far eseguire al robot semplici operazioni;
2. eseguire operazioni complesse (un programma C/C++) grazie alle librerie fornite o ad *Nserver*, con due diversi metodi:
   - creare un sistema *client/server* tra il l'eseguibile e *Nserver*, utilizzando quest'ultimo come interfaccia con il robot stesso (ciò permette inoltre di visualizzare a schermo in *real-time* le operazioni che esso svolge);
   - interfacciarsi direttamente al *Nomad 150* senza dover avviare alcun server.

È preferibile usare *Nserver* quando si è ancora in fase di sviluppo dei propri programmi, grazie alla possibilità di ricevere dati utili a schermo; in seguito si può passare alla modalità diretta, diminuendo in questo modo il carico di dati tra il programma ed il robot. Nel primo caso, è necessario compilare il proprio programma includendo il file *Nclient.o*:

gcc -o prog prog.c Nclient.o

Nel secondo invece si include *Ndirect.o*:

gcc -o prog prog.c Ndirect.o

Capitolo 3

Letteratura precedente inerente allo SLAM

Un approccio classico alla soluzione del problema dello SLAM è dato dall'utilizzo del ***filtro di Kalman***, un filtro lineare ricorsivo che stima la posizione assoluta del robot. Il modello analitico viene descritto dalle seguenti equazioni:

$$\begin{cases} \dot{x} &= A(t)\,x(t) + B(t)\,U(t) + w(t) \\ y(t) &= C(t)\,x(t) + v(t) \end{cases} \tag{3.1}$$

dove $v(t)$ e $w(t)$ sono processi Gaussiani bianchi, mutuamente incorrelati con matrici di covarianza note:

$$E\left[\ w(t)v^T(\tau)\ \right] = 0 \quad \forall\ t,\tau \geq 0$$
$$E\left[\ w(t)w^T(\tau)\ \right] = Q(t)\delta(t-\tau)$$
$$E\left[\ v(t)v^T(\tau)\ \right] = R(t)\delta(t-\tau)$$

Si può considerare ora l'osservatore identità per il modello (3.1):

$$\dot{\hat{x}}(t) = A(t)\,\hat{x}(t) + B(t)\,u(t) + K(t)[y(t) - C(t)\,\hat{x}(t)] \tag{3.2}$$

con stato iniziale $\hat{x}(t_0) = \hat{x}_0$. Si può definire, quindi l'errore di stima $e(t) = x(t) - \hat{x}(t)$ e l'errore quadratico medio di stima

$$E\left[\ e^T(t)e(t)\ \right] \tag{3.3}$$

L'errore quadratico medio di stima all'istante t dipende dalla scelta di $\hat{x}_0$ e della matrice $K(t)$ e fornisce una misura dell'attendibilità della stima $\hat{x}(t)$ all'istante t.

Il progetto dell'osservatore ottimo consiste nel scegliere $\hat{x}_0$ e la matrice $K(t)$ in modo da minimizzare l'errore quadratico medio di stima (3.3). Se la matrice $R(t)$ è definita positiva (se cioè tutte le misure delle componenti del vettore di uscita sono affette da rumore) la soluzione del problema è data da:

$$K(t) = P(t)\,C^T(t)\,R^{-1}(t) \tag{3.4}$$

$$\hat{x}_0 = \bar{x}_0 \tag{3.5}$$

ove $P(t)$ è la soluzione dell'equazione di Riccati

$$\dot{P}(t) = A(t)\,P(t) + P(t)\,A^T(t) - P(t)\,C^T(t)\,R^{-1}(t)\,C(t)\,P(t) + Q(t) \tag{3.6}$$

con $P(t_0) = P_0$

La stima fornita dall'osservatore ottimo detto ***filtro di Kalman***, è non polarizzata cioè il valor medio dell'errore di stima è nullo

$$E[e(t)] = 0$$

mentre la sua matrice di covarianza coincide con $P(t)$:

$$E\left[\ e(t)e^T(t)\ \right] = P(t).$$

Il valore dell'errore quadratico medio di stima è dato da

$$E\left[\ e^T(t)e(t)\ \right] = tr\left[\ P(t)\ \right]$$

La matrice $K(t)$ è detta matrice dei guadagni del filtro. Il filtro di Kalman è il dispositivo che fornisce la stima con errore qudratico medio minimo: non esiste cioè alcun altro dispositivo algebrico o dinamico che, elaborando le misure $y(t)$ e $u(t)$, fornisca una stima dello stato con errore quadratitico medio di stima minore. L'errore di stima $y(t) - C(t)\hat{x}(t)$ è detto *innovazione* ed è un processo stocastico bianco con matrice di covarianza uguale a quella di $v(t)$.

Il filtro di Kalman è un osservatore identità nel quale, anche per sistemi stazionari, la matrice dei guadagni varia nel tempo. Nell'osservatore identità la differenza tra la misura dell'uscita del sistema e la sua stima fornita dall'osservatore è un errore (innovazione) utilizzato per la correzione della stima cioè per estinguere l'errore iniziale. In ambiente stocastico l'innovazione deve essere usata anche per correggere gli effetti di previsione; inoltre a causa degli errori modellati tramite il processo $v(t)$, l'innovazione può essere non nulla anche quando la stima del vettore di stato è esatta. La matrice dei guadagni, $K(t)$, costituisce un compromesso tra due esigenze distinte: l'opportunità di

utilizzare le misure disponibili per correggere la stima dello stato e la necessità di non peggiorare la stima corrente a causa degli errori sulla misura dell'uscita. Si può osservare come nell'osservatore ottimo la proporzionalità tra la matrice $K(t)$ e la matrice di covarianza dell'errore di stima $P(t)$ renda i guadagni dell'osservatore tanto più elevati quanto più elevato è, a parità di altre condizioni, l'errore sulla stima corrente. La proporzionalità tra la $K(t)$ e il prodotto $C(t)\,R^{-1}(t)$ può essere interpretata come proporzionalità tra i guadagni dell'osservatore e l'affidabilità delle misure sull'uscita. Infine per quanto riguarda l'errore di stima, esso è descritto dal seguente modello:

$$\begin{cases} \dot{e}(t) &= [A(t) - K(t)\,C(t)]e(t) + w(t) - K(t)\,v(t) \\ e(t_0) &= x_0 - \bar{x}_0 \end{cases} \tag{3.7}$$

Indicando con $P(t) = E[e(t) - \bar{e}(t)][e(t) - \bar{e}(t)]^T$ la matrice di covarianza dell'errore di stima, si ottiene, dal modello (3.7)

$$\dot{P}(t) = [A(t)-K(t)C(t)]P(t)+P(t)[A(t)-K(t)C(t)]^T+Q(t)+K(t)R(t)K^T(t) \tag{3.8}$$

con la condizione iniziale $P(t_0) = P_0$ cioè, considerando la relazione (3.4), l'equazione di Riccati (3.6). L'equazione (3.8) può venire utilizzata anche per calcolare la matrice di covarianza dell'errore di stima utilizzando una matrice dei guadagni $K(t)$ diversa da quella ottima (3.4).

La popolarità di questo approccio è dovuta a due importanti fattori. Per prima cosa, offre direttamente una soluzione ricorsiva al problema della navigazione e una stima delle posizioni del veicolo e degli oggetti nella mappa sulla base di modelli statistici. In secondo luogo è stata sviluppata una parte consistente del metodo in ambiti aereospaziali, marittimi o in altre applicazioni per la navigazione autonoma.

Lo studio di soluzioni di questo tipo al problema dello SLAM è molto diffuso in letteratura. Nel lavoro introduttivo di Smith *at al.* [4] e Durrant-Whyte [9] vengono stabilite le basi statistiche per descrivere le relazioni tra gli ostacoli e l'incertezza geometrica. Un elemento chiave di questo lavoro è stato mostrare l'esistenza di un alto grado di correlazione tra le stime della posizione dei differenti ostacoli nella mappa; inoltre hanno mostrato che in realtà queste correlazioni crescono ad ogni osservazione successiva. Allo stesso tempo Ayache e Faugeras [1] e Chatila e Laumond [3] hanno indirizzato i loro primi lavori allo studio della navigazione di robot mobili usando algoritmi che si basano sul filtro di Kalman. Questi due lavori hanno molto in comune con il lavoro di Smith, Self e Cheeseman [24]. Viene mostrato che, nel momento in cui un robot si muove attraverso un ambiente sconosciuto prendendo le rilevazioni relative agli ostacoli, le stime di questi ostacoli sono

tutte necessariamente correlate tra loro a causa di errori comuni nella stima della posizione del veicolo. Nei lavori successivi vennero sviluppati due aspetti essenziali per il problema dello SLAM quali:

- per mantenere la consistenza del filtro sono molto importanti le correlazioni tra gli ostacoli nella mappa;
- una soluzione completa dello SLAM richiede che il vettore dello stato, che consiste di tutti gli stati nel modello del veicolo e tutti gli stati di ogni ostacolo della mappa, ha bisogno di essere aggiornato in seguito a ogni osservazione.

Conseguenza di ciò in alcune applicazioni reali è che il filtro di Kalman ha bisogno di usare un vettore di stato molto grande (dell'ordine del numero di ostacoli presenti nella mappa) ed è in generale, computazionalmente intrattabile. Data tale complessità computazionale, sono state proposte una serie di approssimazioni alla soluzione completa dello SLAM le quali assumono che le correlazioni tra gli ostacoli possono essere minimizzate o eliminate in maniera tale da disaccoppiare gli ostacoli e progettare un filtro con costi computazionali decisamente minori (vedi Renken [21], Leonard e Durrant-Whyte [13]).

Molto importante nella trattazione del problema dello SLAM è il lavoro di Dissanayake *et al* [6] il cui fondamentale contributo è stato di dimostrare l'esistenza di una stima non divergente e di chiarire la struttura generale degli algoritmi per la navigazione. Questi contributi sono fondati su tre risultati teorici:

- l'incertezza nelle stime delle mappe relative è monotona decrescente;
- queste incertezze convergono a zero;
- l'incertezza nella posizione del veicolo e nella mappa assoluta ha un *lower bound*.

Per rendere la soluzione al problema dello SLAM *trattabile* dal punto di vista computazionale esistono sostanzialmante due approcci; il primo usa approssimazioni per la stima di correlazioni tra ostacoli, il secondo sfrutta la struttura dello SLAM problem di trasformare il processo di costruzione della mappa in un problema di stima computazionalmente più semplice.

In [11] viene fornita un'implementazione real-time del problema dello SLAM; vengono presentati algoritmi ottimi che considerano forme speciali delle matrici e un nuovo filtro *compresso* che può ridurre in modo significativo la complessità computazionale.

Il problema della complessità computazionale, tuttavia, è solo uno dei problemi relativi all'utilizzo del filtro di Kalman. In primo luogo utilizzare tale filtro presuppone la conoscenza di un modello; inoltre tale modello per poter avere una stima *ottima* dello stato deve essere lineare e infine l'errore al quale sono sottoposte le misure, perchè il filtro funzioni perfettamente, è necessario che sia di tipo gaussiano.

Proprio a causa del fatto che la dinamica del sisema che stiamo considerando è non lineare, spesso si utilizza il Filtro Esteso di Kalman, il quale però non fornisce una soluzione ottima.

L'impossibilità di trovare una soluzione *ottima* al problema dello SLAM ha indotto alcuni ricercatori a sperimentare nuove tecniche di filtraggio che tendono a non fare alcuna supposizione sul tipo di modello e sul tipo di errore. In particolare nel lavoro di Duckett [8] viene presentato un approccio allo SLAM problem completamente diverso dagli approcci classici. Il problema dello SLAM viene definito come un problema di *ottimizzazione globale* nel quale l'obiettivo è la ricerca dello *spazio delle possibili mappe del robot.* A questo scopo si implementa un algoritmo genetico che, a partire da possibili mappe generate casualmente, evolve fino a convergere alla soluzione migliore, ovvero alla mappa migliore. I dati odometrici registrati sono usati come modello dal quale viene generata una popolazione iniziale di possibili traiettorie; ognuna di queste traiettorie viene poi valutata costruendo una griglia globale di occupazione usando i dati provenienti dai sensori. Per ogni soluzione candidata viene poi calcolato un *valore fitness* basato sulla consistenza e la compattezza della mappa prodotta. Le funzioni fitness sono costruite principalmente sulla base di due euristiche; la prima euristica tiene conto della **consistenza** della soluzione trovata mediante la seguente funzione:

$$MC_1 = \sum_i min(occ_i, emp_i)$$

dove occ_i è il numero di letture laser che indica che la cella i è occupata, e emp_i è il numero di letture che indica che la cella è vuota.
La seconda ha lo scopo di rendere la mappa più **compatta** facendo in modo tale da compattare quelle aree che di fatto sono fisicamente adiacenti. Ciò è reso possibile dalla seguente funzione:

$$MC_2 = (x_{max} - x_{min}) \times (y_{max} - y_{min})$$

dove x_{max} e x_{min} sono le coordinate massime e minime del rettangolo misurato in numero di celle.

La funzione fitness infine viene valutata come combinazione delle due euristiche appena descritte:

$$F = MC_1 + wMC_2$$

dove w determina l'importanza relativa di una euristica rispetto all'altra.
Il vantaggio maggiore di tale approccio è che non vengono fatte assunzioni sul modello e sul tipo di errore e il solo parametro critico è il peso w di una euristica rispetto all'altra. Lo svantaggio principale d'altro canto è dato dal fatto che è richiesto un grande costo computazionale. Ciò implica che di questo algoritmo viene fornita una versione *off-line*.

L'obiettivo di questo lavoro di tesi è proprio quello di progettare e implementare una versione *on-line* di un algoritmo genetico per la soluzione al problema dello SLAM.

Capitolo 4

Algoritmi genetici

Una delle più grandi rivoluzioni in ambito scentifico e tecnologico del nostro secolo è certamente rappresentata dall'avvento degli elaboratori elettronici. Tale rivoluzione, per molti, ha come obiettivo finale la realizzazione di programmi per la creazione di modelli del cervello, imitare l'apprendimento umano e simulare l'evoluzione biologica. La prima ha dato origine al settore delle *reti neurali*, la seconda all'*apprendimento automatico* e infine la terza alla *computazione evolutiva*.

4.1 Cenni sulla computazione evolutiva

I primi studi sui sistemi evolutivi sono stati condotti negli anni cinquanta e sessanta con lo scopo di trovare un metodo di ottimizzazione da applicare in problemi di carattere ingegneristico. L'idea di base era quella di fare evolvere una popolazione di soluzioni candidate usando alcuni operatori. Negli anni sessanta Rechenberg (1965, 1973) introdusse le *strategie evolutive* le quali vennero poi sviluppate ulteriormente da Scwefel (1975, 1977). Glover (1977) mise a punto una *ricerca scatter*, la quale considera una popolazione di punti di riferimento e genera successori come combinazioni lineari pesate. Intorno al 1966 poi Fogel, Owens e Walsh hanno sviluppato la *programmazione evolutiva* che consiste nel rappresentare le soluzioni candidate come macchina a stati finiti, fatti evolvere mutando casualmente il diagramma di transizione. Un programma evolutivo è in sostanza un algoritmo probabilistico che prende in considerazione una popolazione di n individui $P(t) = x_1^t, x_2^t, \ldots, x_n^t$ per t iterazioni; ogni soluzione x_i^t viene valutata poi mediante una funzione idoneità (*fitness*).
Gli ***algoritmi genetici*** furono inventati da John Holland negli anni '60 con lo scopo di studiare formalmente il fenomeno dell'adattamento in natura e di trasferire tale fenomeno in sistemi informatici. In *Adaptation in Natu-*

ral and Artificial System vede gli algoritmi genetici come *un'astrazione dell'evoluzione biologica* e fornisce *una struttura teorica per il concetto di adattamento*. Secondo la formulazione fatta da Holland un algoritmo genetico è costituito da una *popolazione iniziale* di *cromosomi* (per esempio stringhe di uno e zero) e, con una sorta di *selezione naturale*, *incrocio* e *mutazione*, si giunge ad una nuova popolazione alla quale a sua volta viene data un'oppurtunità di riproduzione (in media i cromosomi più adatti avranno maggiore probabilità di riprodursi).
Un cromosoma è costituito da *geni* (per esempio bit) ognuno dei quali rappresenta una istanza di un particolare *allele* (per esempio zero o uno). La selezione sceglie i cromosomi più adatti e l'incrocio scambia parti di cromosomi; la mutazione infine modifica casualmente i valori degli alleli in alcune parti.

Il metodo introdotto da Holland ha notevoli differenze rispetto alle strategie evolutive e alla programmazione evolutiva. Le prime infatti partono da una popolazione iniziale di soli due elementi (genitore e figlio) mentre la seconda usa solo la mutazione per garantire una variazione nell'evoluzione del programma. Inoltre Holland, partendo proprio dallo studio teorico del concetto di adattamento in natura, fu il primo ad introdurre una forte e solida base teorica agli algoritmi genetici basata sulla nozione chiave di *schema*.

4.2 Introduzione agli algoritmi genetici

Gli *algoritmi evolutivi* emulano processi di evoluzione naturale per risolvere problemi di ricerca globale, anche con scarsa conoscenza del dominio; in particolare, lì dove lo spazio di ricerca ha dimensioni notevoli ed eseguire una ricerca esaustiva sarebbe computazionalmente impensabile.

Alla base del modello evolutivo teorizzato da Darwin ci sono delle caratteristiche che sono facilmente traslabili nel campo della *computer science*, quali ad esempio la *varietà* e la *selezione naturale*. La prima assicura che lo spazio di ricerca non sia statico, ma contenga sempre nuovi candidati, in modo da adattarsi sempre all'ambiente. Ovviamente, questa peculiarità è un fattore determinante per gli algoritmi *adattivi*, quelli cioè dove non è possibile perseguire una ricerca tramite la consocenza del dominio di partenza, in quanto sempre diverso. La seconda caratteristica, invece, garantisce la sopravvivenza dei candidati più idonei all'ambiente, facendo evolvere individui con caratteristiche diverse in una sorta di competizione. Un grande ruolo svolge la casualità di tutto il processo; essa assicura in la varietà suddetta, impedendo di convergere solo verso un certo tipo di essere vivente, inoltre conferisce al processo evolutivo un andamento ad *equilibri punteggiati*, ovvero

che procede per salti, alternando a momenti di ristagno momenti di grandi cambiamenti. Il complesso di tali fattori, applicato ad una popolazione enorme di invidui che evolvono parallelamente, permette di perseguire scopi complessi senza la necessità di una codifica manuale come nella classica programmazione *top-down*. Usando invece il paradigma *bottom-up*, è semplice replicare tale schema con la computazione parallela; per questo molti ricercatori dell'AI hanno visto nell'evoluzione biologica una fonte di ispirazione per affrontare vari problemi.

Proprio per il loro scarso legame col dominio di applicazione e la loro versatilità, gli *algoritmi evolutivi* vengono considerati metodi di ricerca deboli, a differenza di quelli forti che sono strettamente connessi al contesto di utilizzo. In realtà oggi si parla anche di metodi *deboli evolutivi*, dove cioè al passare del tempo si apprende sempre di più sullo spazio di ricerca, sfruttando quindi un numero maggiore di assunzioni.

Dal punto di vista formale, data una funzione $f : D \rightarrow \Re$, dove D è uno spazio cartesiano, un algoritmo evolutivo mira a trovare un $\tilde{X} \in D$, per cui valga la seguente relazione:

$$\forall X \in D : f(X) \leq f(\tilde{X})$$

Solitamente f prende il nome di *funzione fitness*, che valorizza la qualità degli individui; l'obiettivo è convergere ad un massimo globale, evitando punti di singolarità locali, e senza dover eseguire una ricerca esaustiva.

Le tecniche evolutive trovano grosso utilizzo in molti campi della ricerca: per la simulazione di sintesi delle proteine, per modellare sistemi immunitari naturali, nell'ambito del *machine learning*, per risolvere problemi di ottimizzazione come la disposizione del maggior numero di porte logiche all'interno di una piastrina di silicio. Il loro vantaggio é di essere *general purpose*, di essere adattivi, di non richiedere alcuna conoscenza sul dominio di applicazione. Tra gli svantaggi, tali metodi hanno una grande quantità di parametri da settare che richiedono un *tuning* abbastanza lungo. Inoltre, a causa delle dimensioni notevoli di candidati su cui lavorano, essi possono risultare onerosi a livello computazionale (sia in termini di memoria sia di tempo); grazie al loro alto grado di parallelizzazione sono stati però sviluppati versioni parallele degli algoritmi:

- **modello globale:** si calcolano i fitness in parallelo sui vari processori disponibili (nel caso migliore un processore per ogni individuo), la popolazione resta sempre una globale e tutto il resto dell'algoritmo viene elaborato in modo sequenziale ;

- **modello ad isole:** la popolazione viene divisa in gruppi detti *isole*, ognuna localizzata in un processore, e tutte le operazioni sono eseguite localmente solo per un'isola ;
- **modello diffusivo:** ogni elemento k viene sostituito da un nuovo elemento nato dall'incorcio dei vicini di k, quindi ancor più locale del precedente modello.

Dagli algoritmi evolutivi si dipartono tecniche come:

- programmazione evolutiva;
- strategie evolutive;
- sistemi classificatori;
- algoritmi genetici;
- programmazione genetica.

Le prime tre categorie hanno suscitato scarso interesse nella comunità; la programmazione evolutiva sfrutta esclusivamente l'operazione di mutazione e assume come candidati dei veri e propri programmi (rappresentati come macchine a stati finiti), mentre le strategie evolutive non utilizzano una popolazione di individui ma si incentrano particolarmente sul rapporto genitore-figlio. La programmazione genetica nasce dalla programmazione evolutiva, ma utilizza prevalentemente l'operatore di incrocio; l'idea alla base è di far evolvere vari tipi di programmi fino a convergere a quello maggiormente in grado di risolvere un particolare problema.

Al fine di fornire strumenti adeguati per il seguito del presente lavoro, questa digressione sulle tecniche evolutive si incentra particolarmente sull'ultima categoria, quella degli *algoritmi genetici*. Tralasciando i cenni storici, trattati nel paragrafo 4.1, nonché la dimostrazione formale di Holland, che sarà oggetto del seguente, risulta interessante chiarire l'idea che c'è dietro gli algoritmi genetici, ed introdurre un pò di terminologia, ereditata dalla biologia. Un *algoritmo genetico* è un un algoritmo che ricerca la soluzione di un problema facendo evolvere geneticamente un insieme di soluzioni, scelte all'inizio casualmente. Gli individui più forti avranno maggiori possibilità di sopravvivere e di riprodursi, e genereranno un numero maggiore di discendenti, tramandando quindi i loro geni alle generazioni successive. Ovviamente, a seguito della riproduzione, un figlio può essere in modo casuale tanto più forte del padre, quanto più debole, ma parallelizzando il processo per un numero enorme di individui si vengono comunque a creare vari corredi genetici

forti, seppur diversi tra di loro. In generale questo implica che, seppure l'incertezza indotta dalla riproduzione tra individui può dar luogo a discendenti peggiori, si assiste ad una convergenza verso entità via via più forti.

Vista l'evidente affinità di questi algoritmi con l'evoluzione naturale, molti dei termini presenti in letteratura sono stati presi in prestito dalla biologia. Tutti gli organismi viventi sono composti da cellule, che nel loro nucleo contengono lo stesso patrimonio genetico, detto ***genoma***; questi è l'insieme di uno o più ***cromosomi***, composti a sua volta da ***geni***. Un gene individua una particolare caratteristica dell'individuo, ed i modi possibili di tale caratteristica vengono detti ***alleli***. L'insieme delle caratteristiche presenti in un cromosoma, ovvero l'insieme di geni presenti, e la loro posizione al suo interno, costituiscono il ***genotipo***, mentre l'instanza di tali caratteristiche con opportuni valori, ovvero la fisionomia che risulta nell'individuo, prende il nome di ***fenotipo***. Se i cromosomi si presentano accoppiati, si parla di organismi ***diploidi***, se sono singoli ***aploidi***; nel primo caso la riproduzione prevede uno scambio di geni tra i due elementi della coppia di ciascun genitore, accoppiando poi i due cromosomi prodotti. Nel secondo caso invece i geni vengono scambiati direttamentre tra i due genitori, dando vita a due discendenti.

Negli *algoritmi genetici* un cromosoma è una soluzione candidata, di solito rappresentata come una stringa binaria, mentre un gene può essere uno o più bit, i cui alleli sono i valori 0 o 1. In realtà il genoma è composto da un unico cromosma, quindi i due termini sono interscambiabili in questo contesto; inoltre parliamo di organismi aploidi e quindi l'incrocio consiste nello scambio di alcuni bit tra due stringhe binarie.

Le fasi presenti in una singola iterazione di un algoritmo genetico sono le seguenti (si veda anche il diagramma in figura 4.1):

1. **codifica -** le soluzioni candidate vengono codificate come stringhe binarie, ovvero si identificano i genotipi dei corrispondenti fenotipi;

2. **fitness -** si valuta la qualità di ciascun individuo, sulla base di una funzione euristica;

3. **selezione -** si sceglie quali cromosomi possono riprodursi, quante volte e quanti figli genereranno;

4. **incrocio -** si scelgono le coppie e si fanno mischiare i loro geni, generando due figli per coppia;

5. **mutazione -** si inverte un singolo bit, al fine di esplorare genomi sempre nuovi e diversi;

6. **decodifica -** dai nuovi genotipi nati, si decodificano i corrispondenti fenotipi.

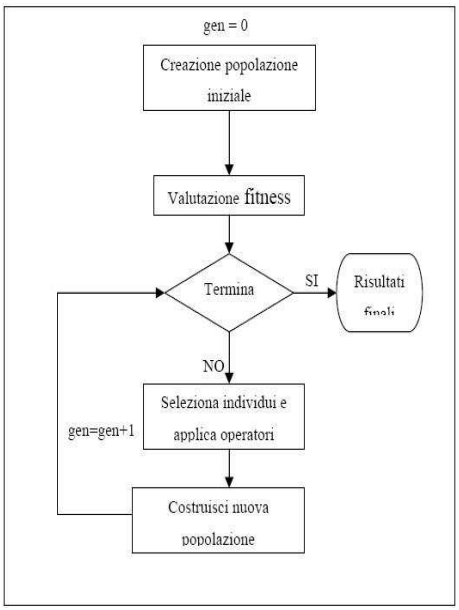

Figura 4.1: Schema di un *algoritmo genetico*

Al fine di collocare gli *algoritmi genetici* in un contesto più ampio, vale la pena soffermarsi sui vari metodi usati per risolvere problemi di ricerca di una soluzione ideale. In generale il termine *ricerca* può esser visto da almeno tre punti di vista diversi:

- **ricerca di dati memorizzati -** il problema è di ritrovare una particolare informazione da una base di dati, dove sono memorizzate e rappresentate secondo un certo paradigma logico (ad esempio secondo un modello relazionale);

- **ricerca di percorsi verso obiettivi -** bisogna individuare una serie di azioni che portano dallo stato iniziale di un agente al suo stato obiettivo. È un tipo di ricerca molto usata nell'AI, dove al fine di risolvere un problema è appunto necessario compiere una serie di azioni. Alcuni degli algoritmi più usati in tale contesto sono la *visita in profondità* (*depth-first search*), il metodo *separa e limita* (*branch and bound*) e l'algoritmo $A*$;

- **ricerca di soluzioni -** È il tipo di ricerca più generale, che ha alla base l'obiettivo di trovare la soluzione migliore da una base di soluzioni candidate.

Il primo tipo di ricerca è possibile solo se abbiamo una serie di dati esplicitamente organizzati e memorizzati, che permettono di giungere solo a certi tipi di informazioni[1]. Ovviamente, non sempre si hanno dei dati in possesso, che rendano la ricerca un semplice reperimento di informazioni, ma è necessario acquisire tali informazioni, valutarle, e confrontarle tra di loro; inoltre quasi sempre svolgere tale ricerca in modo esaustivo richiederebbe un tempo eterno. Il secondo tipo di ricerca, quello di percorsi verso obiettivi, si basa sulla visita di un albero, dove ogni nodo rappresenta uno stato ed i suoi figli i possibili stati successivi. Se si andasse ad analizzare tutto l'albero, ovviamente si tornerebbe ad un caso di ricerca esaustiva; è però possibile, sulla base di certe funzioni euristiche, limitarne la visita solo a certi rami. Non sempre ricercare richiede però di dover trovare degli stati intermedi per giungere ad un obiettivo, perché non sempre sono richieste delle azioni per perseguire un risultato. Quando l'unica cosa importante è il risultato finale di una ricerca, entrano in gioco i metodi di ricerca generali, come gli *algoritmi genetici*[2]. Alcuni dei metodi più usati sono il metodo della *salita lungo il gradiente* (*hill climbing*), la *ricottura simulata*, la ricerca tabù.

4.3 Fondamenti teorici degli algoritmi genetici

Gli algoritmi genetici, secondo la formulazione tradizionale introdotta da Holland (1975), funzionano scoprendo, favorendo e ricombinando le soluzioni "buone". Esse tendono a essere formate da buoni "*blocchi costitutivi*" (combinazioni di valori di bit che conferiscono al cromosoma al quale appartengono maggiore idoneità).
Per formalizzare la nozione di blocco costitutivo, Holland introdusse il concetto chiave di ***schema***. Uno schema è un'insieme di stringhe di bit che possono essere descritte da un modello costituito da *uno*, *zero* e da *caratteri jolly* (possono essere uno o zero). Per esempio:

$$H = \underbrace{1 * * * \cdots * 1}_{\iota \text{ volte}}$$

[1] Un DBMS permette all'utente di chiedere solo ciò per cui è stato progettato, ma non di rispondere a qualsiasi richiesta inerente ai dati che possiede.

[2] È ovvio che gli algoritmi di ricerca di percorsi verso obiettivi possono esser visti come un sottoinsieme di quelli di ricerca di soluzioni generici.

rappresenta l'insieme di tutte le stringhe di lunghezza l che iniziano e finiscono con uno.
Gli schemi vengono inoltre detti ***iperpiani*** perchè essi definiscono piani di varie dimensioni nello spazio l-dimensionale delle stringhe di bit di lunghezza l; mentre le stringhe appartenenti ad H sono anche dette ***istanze*** di H. In generale schemi diversi hanno diverse caratteristiche ma possiedono comunque alcune proprietà comuni come *l'ordine* e la *lunghezza di definizione*. *L'ordine* $o(H)$ è rappresentato dal numero di bit definiti (ovvero non caratteri jolly)

$$o(s) = \text{lunghezza totale dello schema - numero di caratteri jolly}$$

La lunghezza di definizione $d(H)$ è la distanza tra la prima e l'ultima posizione fissata nella stringa; essa è molto importante perchè definisce in un certo senso la *compattezza* della stringa. Si può osservare inoltre che ci sono 2^l possibili stringhe con 2^{2^l} possibili sottoinsiemi ma solo 3^l possibili schemi. Ciò implica che non tutti i possibili sottoinsiemi dell'insieme delle stringhe di lunghezza l possono essere descritti da uno schema. Così data una popolazione di n stringhe, il numero di schemi diversi di cui le stringhe sono istanze è compreso tra 2^l e $n2^l$. Ciò implica che se da una parte l'algoritmo genetico valuta esplicitamente l'idoneità di n individui, dall'altra in realtà stima implicitamente l'idoneità media di un numero molto maggiore di schemi dove l'idoneità media di uno schema è definita come l'idoneità media di tutte le possibile istanze di quello schema. Così come gli schemi non sono rappresentati o valutati esplicitamente dall'algoritmo genetico, anche le stime dell'idoneità media degli schemi non sono calcolate o memorizzate in modo esplicito. Tuttavia, il comportamento dell'algoritmo genetico in termini di crescita di uno schema da una generazione alla successiva, può essere descritto come se queste medie fossero effettivamente calcolate e memorizzate. Esso può essere descritto come segue.
Sia H uno schema con almeno una istanza appartenente alla popolazione, $\hat{u}(H,t)$ l'idoneità di H all'istante t tale che:

$$\hat{u}(H,t) = \sum_{x \in H} \frac{f(x)}{m(H,t)}$$

con $f(x)$ idoneità di x e $m(H,t)$ numero di istanze appartenenti ad H all'istante t; e tenendo conto inoltre che per ogni istanza di H la probabilità di selezione per la stringa x_i è data da:

$$p_{sel} = \frac{f(x_i)}{\sum_{x \in pop} f(x_i)}$$

è chiaro che il numero medio di istanze appartenenti allo schema H all'istante $t+1$ è dato dalla seguente relazione:

$$E[m(H,t+1)] = m(H,t) \cdot \text{PopSize} \cdot \frac{\hat{u}(H,t)}{\sum_{x \in pop} f(x_i)} \tag{4.1}$$

Considerando poi che l'doneità media si scrive come:

$$\bar{f}(t) = \frac{\sum_{x \in pop} f(x_i)}{\text{PopSize}}$$

allora la (4.1) può essere riscritta nel modo seguente:

$$E[m(H,t+1)] = m(H,t)\frac{\hat{u}(H,t)}{\bar{f}(t)} \tag{4.2}$$

La (4.2) pone in evidenza la dipendenza implicita del numero medio di istanze al tempo $t+1$ con l'idoneità media di H. D'altra parte però nella descrizione appena effettuata della dinamica dell'algoritmo genetico, non si è ancora tenuto conto degli effetti dell'incrocio e della mutazione. Per capire come l'incrocio agisce nell'andamento del numero di istanze apparteneti ad un dato schema si può prendere in esame l'esempio seguente:

Sia $s_1 = (11101111101000100011000000 1000110)$ una generica stringa; essa può essere considerata come istanza di due schemi $H_0 = (****111* \ldots *)$ e $H_1 = (111 * \ldots * 10)$. Si può a questo punto supporre di fare incrociare s_1 con la stringa $s_2 = (00010100001001010100101011111 1011)$ nella posizione 20. Si produrranno così facendo due discendenti:

$$\begin{aligned} s_1' &= (11101111101000100011101011111 1011) \\ s_2' &= (00010100001001010100000000 1000110) \end{aligned}$$

Si può facilmente notare che lo schema H_0 *sopravvive* a tale incrocio, cioè uno dei discendenti sarà istanza di H_0, mentre lo schema H_1 verrà distrutto, ovvero nessun discendente apparterrà a tale schema. La ragione di questo fatto risiede nel concetto di "lunghezza di definizione". Essa come risulta abbastanza evidente da questo esempio gioca un ruolo molto rilevante ai fini della distruzione o della sopravvivenza di un determinato schema. Infatti La lunghezza di definizione dello schema H_0 è $d(H_0) = 2$ mentre la lunghezza di definizione dello schema H_1 è $d(H_1) = 32$.

Da questi semplici ragionamenti si può dedurre la probabilità di distruzione di un generico schema H:

$$p_d = \frac{d(H)}{l-1}$$

perchè la posizione di incrocio è selezionata uniformemente tra le $m-1$ possibili posizioni.
Da cui la probabilità di sopravvivenza:

$$s_c = 1 - p_d = 1 - \frac{d(H)}{l-1} \tag{4.3}$$

Ciò è quanto avviene se i cromosomi ad una data popolazione vengono fatti incrociare sempre, con probabilità uguale ad uno. Nel caso in cui, invece, l'incrocio viene effettuato con una certa probabilità p_c allora la (4.3) diventa:

$$s_c = 1 - p_c \frac{d(H)}{l-1} \tag{4.4}$$

In generale però esistono incroci che pur avvenendo nelle posizioni definite non distruggono lo schema. In virtù di tale ragione la (4.4) viene modificata con un limite superiore:

$$s_c \geq 1 - p_c \frac{d(H)}{l-1} \tag{4.5}$$

In modo molto simile la probabilità s_m che H sopravviva alla mutazione:

$$s_m = (1-p_m)^{o(H)} \tag{4.6}$$

dove p_m è la probabilità che venga mutato un bit di una stringa. La relazione (4.6) ha una sua dimostrazione immediata considerando che per ogni bit la probabilità che un bit non sia mutato è $(1-p_m)$ da cui la probabilità che nessun bit sia mutato è $(1-p_m)$ moltiplicato per se stesso $o(H)$ volte.
Tenendo perciò conto degli effetti "distruttivi" dell'incrocio e della mutazione l'equazione (4.2) diventa:

$$E[m(H,t+1)] \geq \frac{\hat{u}(H,t)}{\bar{f}(t)} m(H,t)(1 - p_c \frac{d(H)}{l-1})(1-p_m)^{o(H)} \tag{4.7}$$

Questa disuguaglianza è nota come "***Teorema degli schemi***". Secondo tale teorema, tenendo inoltre presente la (4.5) e la (4.6), si può quindi osservare che schemi brevi e di basso ordine presentano un numero di istanze crescente. Inoltre il numero di istanze cresce in modo esponenziale. Infatti, se per esempio consideriamo che lo schema H sia superiore alla media di un fattore percentuale ε, cioè:

$$\hat{u}(H,t) = \bar{f}(t) + \varepsilon \bar{f}(t)$$

allora, facendo opportune sostituzioni ritroviamo:

$$E[m(H,t)] = E[m(H,0)](1+\varepsilon)^t \tag{4.8}$$

La (4.8) rappresenta una progressione geometrica, per cui lo schema con idoneità sopra la media riceve un incremento di tipo esponenziale.

La teoria appena descritta soprattutto negli ultimi anni è stata oggetto di numerose critiche. Innanzitutto tale teoria è tutt'altro che esaustiva; infatti essa tiene conto solo degli effetti "distruttivi" dell'incrocio e della mutazione tralasciando completamente quelli che sono gli effetti "costruttivi", ovvero la capacità di dare luogo, dati due individui appartenenti a schemi non molto buoni, ad individui che sono istanze di schemi migliori. Questa capacità "costruttiva" degli algoritmi genetici è nota come "***Ipotesi dei Blocchi Costitutivi***" (*Building Block Hypothesis* in Goldberg 1989). Questa ipotesi in realtà non trova un vero e proprio riscontro in una formulazione matematica.
Il Teorema degli schemi e l'ipotesi dei blocchi costitutivi si interessano principalmente del ruolo fondamentale dell'incrocio tralasciando invece la capacità insita nella mutazione di garantire una certa diversità tra i cromosomi. Quasta è una proprietà molto forte che in un certo senso permette di evitare la convergenza dell'algoritmo verso minimi locali. Inoltre la formulazione appena effettuata ha il forte limite che consiste nell'analisi di blocchi costitutivi derivanti dall'applicazione di un certo tipo di operatore di incrocio. Di recente alcuni ricercatori infatti hanno analizzato e messo a punto nuovi metodi di incrocio che manipolano tipi diversi di blocchi costitutivi. Questi ovviamente non sono gli unici limiti alla formulazione teorica sugli algoritmi genetici, ma esistono tutt'ora molti problemi aperti. Resta ancora infatti da determinare le leggi generali che descrivono il comportamento macroscopico degli algoritmi genetici; non si conoscono ancora bene, inolre, quali previsioni si possono fare sulla variazione dell'idoneità in funzione del tempo e sulla dinamica della struttura della popolazione. Non si conosce bene l'influenza degli operatori a basso livello su tale dinamica. Infine non si sa con chiarezza in quali tipologie di problemi gli algoritmi genetici producono risultati utili e migliori di altre tecniche. Tutte queste problematiche sono tutt'ora oggetto di numerose ricerche e discussioni critiche che stanno dando vita a nuovi approcci alla formulazione classica di Holland.

4.4 Implementazione

4.4.1 Codifica

Sulla base di quanto visto nella sezione 4.3 , la codifica assume un ruolo tutt'altro che di rilievo negli *algoritmi evolutivi* , proprio al fine di assegnare

a individui forti degli schemi di ordine basso e brevi.

Dopo Holland, che ha basato i suoi studi su codifiche di tipo binario, c'è stato un proseguio nell'uso di tale metodo, mentre solamente negli ultimi anni ci si è spostati su altri tipi di codifiche. In realtà, a tutt'oggi si discute sulla bontà di una codifica binaria rispetto ad una con un alfabeto a cardinalità molto più grande (come quello decimale ad esempio), giungendo però a risultanti discordanti. Holland ha basato la sua teoria degli schemi sul primo tipo di codifica, confrontando una codifica binaria con stringhe lunghe 100 con una con molti più alleli ma 30 di lunghezza. Argomentando come fattore dominante degli *algoritmi evolutivi* la nozione di schema, è ovvio che il primo tipo permette di avere per ogni istanza 2^{100} schemi contro 2^{30} del secondo[3].

Uno degli svantaggi delle codifiche binarie è di risultare alquanto innaturali rispetto a molti dei problemi che si affrontano, per questo ultimamente si sono studiate delle codifiche alternative. Alcune rappresentazioni usate sono i *caratteri di Kitano* per le grammatiche di generazione dei grafi, la rappresentazione a valori reali di Meyer e Packard per gli insiemi di condizioni, quelle di Montana e Davis per i pesi delle reti neurali, quella di Schultz-Kremer per gli angoli di torsione delle proteine. In realtà, nonostante la critica di Holland, molti studi empirici hanno dimostrato risultati migliori con codifiche diverse da quella binaria. Un tipo di codifica innovativa su cui si sta ancora lavorando è quella ad albero, che permette di lavorare su spazi di ricerca infiniti, perché dall'incrocio possono uscire alberi a qualsiasi dimensione. Nel futuro di questo campo c'è la ricerca di codifiche dinamiche, in grado di modificarsi in funzione del problema, ma con una struttura che resti sempre uguale.

Si è discusso precedentemente di come, in biologia, un gene esprima una particolare caratteristica indipendentemente dalla posizione che ha nel cromosoma, anche se spesso può accadere che i geni lavorino insieme in una sorta di rete di compensazione. Negli *algoritmi genetici* invece, a seguito dell'incrocio, tale gene si può spezzare, o può perdere la sua posizione, comportando una perdita di idoneità; tale problema va sotto il nome di *problema dell'interconnessione*. Quello che si vuole fare è cercare di adattare la codifica al problema, ovvero, una volta identificati per via empirica quali sono gli schemi più forti, se ne studia la loro codifica, e si cerca di riordinare gli alleli in quel modo.

[3] È comunque opportuno ricordare che la teoria di Holland è stata criticata proprio per questo ruolo predominante dello schema rispetto al rapporto genitore-figlio.

La prima tecnica finalizzata a tale scopo è l'*inversione*, un operatore di riordinamento che permette di spostare, come nella biologia, la posizione di un allele, senza modificare il suo ruolo all'interno del cromosoma. In pratica, si associa ad ogni allele la sua posizione originaria, in modo da poterlo poi spostare per rendere la codifica più idonea, garantendo dopo l'incrocio che possa esser riportato in quella. Lo spostamento avviene scegliendo due alleli, e cambiandoli di posto tra di loro, come mostrato nell'esempio, dove la coppia (x, y) rappresenta la posizione x dell'allele y:

$$(1,0)(2,0)(3,0)(4,1)(5,0)(6,1)(7,0)(8,1)$$
$$(1,0)(2,0)(6,1)(4,1)(5,0)(3,0)(7,0)(8,1)$$

L'inversione, unita all'incrocio, genera però una serie di problemi, come mostrato nel seguente esempio. Supponiamo di avere i seguenti cromosomi da incrociare:

$$(1,0)(2,0)(6,1)(5,0)(4,1)(3,0)(7,0)(8,1)$$
$$(5,1)(2,0)(3,1)(4,1)(1,1)(8,1)(6,0)(7,0)$$

Se il punto di incrocio cade dopo il terzo bit i due discendenti saranno

$$(1,0)(2,0)(6,1)(4,1)(1,1)(8,1)(6,0)(7,0)$$
$$(5,1)(2,0)(3,1)(5,0)(4,1)(3,0)(7,0)(8,1)$$

Il primo figlio presenta due copie di bit 1 e 6, mentre il secondo di bit 3 e 5; in che modo si può risolvere? Vi sono sostanzialmente due tecniche possibili:

- permettere l'incrocio esclusivamente tra cromosomi con la stessa permutazione di alleli;
- si sceglie uno dei due genitori come *padrone* e l'altro come *schiavo*: lo schiavo viene riordinato come il padrone e dopo l'incrocio il discendente riacquisterà l'ordine originale.

Un'altra, ideata nel 1987 da Schaffer e Morishima, è detta *evoluzione di punti caldi*. Questa consiste nell'associare al cromosoma una maschera di pari lunghezza, dove se il bit i è posto a 1 si può incrociare tra l'allele i e quello *i+1*; in pratica si decide dove è possibile eseguire l'incrocio per ogni cromosoma. Un esempio è il seguente:

1 0 0 1!1 1 1!1

0 0 0 0 0 0!0 0

I punti esclamativi rappresentano i punti di incrocio possibili, ovvero la maschera di incrocio[4]. Tale tecnica si presta poi per l'incrocio a più punti, che darà luogo, nel precedente esempio, ai seguenti discendenti:

1 0 0 1!0 0!1!0

0 0 0 0 1 1 0 1

Da notare che anche i punti esclamativi vengono ereditati.

4.4.2 Selezione

La selezione è la fase degli *algoritmi genetici* dove si scelgono gli individui che genereranno i discendenti della prossima generazione, nonché il numero di figli che possono avere. Vi sono due possibili tipi di popolazioni:

- *generazionale*, ovvero che viene interamente sostituita dai discendenti ad ogni generazione;
- *steady-state*, dove solo una porzione di essa viene sostituita.

Se la selezione predilige troppo gli individui forti, si rischia di avere un'esplorazione minima all'interno dello spazio di ricerca, convergendo troppo rapidamente e rischiando di incorrere in un massimo locale; al contrario una selezione casuale comporterebbe una convergenza lenta. La prima parte della selezione si occupa di scegliere gli individui destinati a riprodursi e spostarli nella cosiddetta *mating pool*, dove verrano poi accoppiati.

La tecnica più usata per tale cosa è quella *proporzionale all'idoneità*. Il numero *atteso* di volte in cui l'individuo viene scelto per riprodursi prende il nome di *valore atteso*, ed è pari al rapporto tra l'idoneità (fitness) dell'individuo e quella media della popolazione. In questo modo più un individuo è idoneo più volte ci si aspetta che venga scelto. Il *metodo della roulette*, proposto da Holland, prevede di assegnare ad ogni cromosoma una fetta di roulette in funzione di tale valore atteso, di farla girare N volte lanciando la pallina, e di selezionare l'individuo su cui la pallina si ferma. Un limite di tale tecnica è che potrebbe anche accadere di avere N lanci pessimi, ovvero dove la pallina cada sempre sugli individui peggiori, perdendo quindi i fitness maggiori.

[4]Una notazione più esplicita è 10011111 : 00010010.

A tale scopo James Baker ha proposto un tipo di campionamento diverso, sempre basato su un modello proporzionale all'idoneità: il *Campionamento Stocastico Universale (CSU)*. Per evitare di perdere fitness buoni, si esegue un solo giro di roulette e si usano N indici equidistanti per selezionare gli individui; in tal modo un cromosoma forte viene selezionato quasi certamente, in quanto è difficile che nessun indice compaia nella sua fetta. Entrambe le tecniche di campionamento viste, essendo basate su proporzionalità al fitness, soffrono della possibilità di portare ad una convergenza prematura, poichè favorendo i cromosomi forti portano un aumento esponenziale delle istanze dei loro schemi (come visto nella teoria di Holland al par. 4.3).

Alcune tecniche cercano di garantire un maggior movimento orizzontale (esplorazione) in favore di quello verticale (selezione), arrivando al numero di selezioni di un individuo direttamente dal fitness, tramite un cambiamento di scala (tecniche di *rimappamento implicito del fitness*). Il *cambiamento di scala sigma* porta un numero di discendenti agli individui più idonei costante, indipendentemente dalla varianza dell'idoneità della popolazione:

$$expval(i,t) = \begin{cases} 1 + \frac{f(i)-\bar{f}(t)}{2\sigma(t)} & \text{se } \sigma(t) \neq 0 \\ 1.0 & \text{se } \sigma(t) = 0 \end{cases}$$

dove *expval(i,t)* è il valore atteso dell'individuo i al tempo t, *f(i)* l'idoneità di i, $\bar{f}(t)$ l'idoneità media della popolazione al tempo t e $\sigma(t)$ la deviazione standard della popolazione.

Rispetto al cambiamento di scala sigma, la *selezione di Boltzmann* garantisce pressioni evolutive diverse nel tempo. In particolare, sarebbe necessario mantenerla bassa all'inizio dell'algoritmo, al fine di esplorare ampiamente nello spazio di ricerca, e accelerarla quando invece è il momento di convergere verso una soluzione. L'approccio si basa su una *temperatura* che influenza il valore atteso degli individui, partendo alta e scendendo mano mano, raffreddando sempre di più il movimento orizzontale. Un esempio è:

$$expval(i,t) = \frac{e^{f(i)/T}}{\langle e^{f(i)/T} \rangle_t}$$

T rappresenta la temperatura e $\langle\rangle_t$ la media sulla popolazione al tempo t.

Un ultimo tipo di tecniche che vale la pena vedere è quella che rimappa esplicitamente il fitness, al fine di ottenere il valore atteso. La *selezione a torneo* prevede delle competizioni tra individui, basate sul fitness di ciascuno. Il caso più semplice, la *selezione a torneo binaria*, prende coppie di cromosomi casualmente e mette nella *mating pool* quello più idoneo. Si possono

anche fare tornei tra N cromosomi, avvantaggiando in questo modo i fitness migliori, dato che un individuo debole avrà ancor meno possibilità di vincere. Esistono infine tornei *probabilistici*, dove cioè il cromosoma più forte vince solo entro una certa probabilità $r < k$, dove k è una soglia prestabilità.

4.4.3 Incrocio

L'incrocio è un passo fondamentale in un algoritmo genetico. Nel paragrafo (4.3), infatti, si vede chiaramente come questo operatore sia alla base del *Teorema degli schemi* formulato da Holland negli anni '70. Nella formulazione teorica originale l'incrocio veniva effettuato in un punto: si sceglie a caso una singola posizione e si scambiano le parti dei due genitori che seguono e precedono la posizione di incrocio per formare i due discendenti, ricombinando così blocchi costitutivi appartenenti a stringhe diverse. Questo modo di effettuare l'incrocio tra due individui ha però alcuni limiti. Per esempio non è possibile ricombinare tutti gli schemi possibili. Se consideriamo infatti istanze di $11*****1$ e di $****11**$ non è possibile ricombinarle per formare un'istanza di $11**11*1$. Inoltre effettuando l'incrocio in un punto, come si può facilmente osservare in (4.5), la probabilità che uno schema sopravviva è maggiore se lo schema ha una lunghezza di definizione minore. Questo fatto è noto come "**dipendenza posizionale**". Per evitare questi problemi legati all'impossibilità di ricombinare tutti gli schemi possibili e per abbassare la dipendenza posizionale si può introdurre un ulteriore operatore descritto nella sezione 4.4.1: l'inversione. Tuttavia alcuni ricercatori sostengono che l'inversione non riesce comunque a mettere vicini tutti i bit legati dal punto di vista funzionale. Hanno inoltre osservato che l'incrocio in un punto tratta alcune posizioni in modo preferenziale nel senso che i segmenti scambiati tra due genitori contengono sempre gli estremi della stringa.

Allo scopo di superare questi problemi si utilizzano altre tecniche di incrocio. Ad esempio si può utilizzare l'incrocio a due punti: l'idea qui è quella di scegliere a caso due posizioni e scambiare i segmenti compresi. Questa tecnica ha il grande vantaggio di ridurre la *dipendenza posizionale* e ha meno probabilità di distruggere schemi con elevata lunghezza di definizione; inoltre i segmenti scambiati non contengono necessariamente gli estremi delle stringhe. Negli ultimi dieci anni poi sono state messe a punto numerose altre tecniche di incrocio come per esempio l'incrocio paramtrico uniforme, in cui gli scambi si possono effettuare in ogni posizione con probabilità p.

In generale è abbastanza complicato scegliere quale tecnica di incrocio sia più conveniente utilizzare. La difficoltà nasce dal fatto che gli operatori a basso livello di un algoritmo genetico hanno un forte legame tra loro. In particolare l'incrocio dipende dal tipo di funzione fitness utilizzata, dalla cod-

ifica scelta e da altre caratteristiche. Inoltre tale legame è uno dei problemi aperti nell'ambito degli algoritmi genetici. Ad esempio se si impiega una codifica ad albero (si veda a riguardo la sezione 4.4.1) è necessario utilizzare una tecnica di incrocio e mutazione appositamente definita. Nel caso, invece, di codifica binaria le tecniche di incrocio sopra descritte si possono adattare abbastanza bene. Come già visto nel paragrafo 4.3 e accennato all'inizio di questa sezione, l'incrocio è un operatore fondamentale nell' evoluzione di un algoritmo genetico. Esso infatti nella formulazione classica di Holland è uno dei principale artefici della sopravvivenza o della distruzione di uno schema. Nell'*Ipotesi dei Blocchi Costitutivi*, invece, si congettura la possibilità che tale operatore ha nel ricombinare a partire da schemi non molto buoni o buoni, schemi nettamente migliori. Inoltre si potrebbe prendere in considerazione l'idea che questo operatore invece di essere utile ai fini di preservare o distruggere blocchi costitutivi, potrebbe essere utile come operatore di "***macro-mutazione***" che, esattamente come la mutazione, consente ampi spostamenti nel campo di ricerca, evitando così di incorrere in massimi locali.

4.4.4 Mutazione

Come visto nella sezione precedente, l'incrocio, secondo alcuni ricercatori, può essere visto come un operatore di "***macro-mutazione***" consentendo così una ampia e il più possibile esaustiva ricerca nello spazio delle soluzioni. Questo, in generale, è l'obiettivo della ***mutazione***. Nella formulazione classica di Holland, tuttavia, essa provvede semplicemente ad assicurare che la popolazione non prenda valori fissati in certe posizioni, ricoprendo quindi un ruolo marginale rispetto all'incrocio. Nella programmazione evolutiva e nelle prime versioni di strategie evolutive, la mutazione è l'unico operatore ad avere il ruolo di garantire una certa varietà nell'evoluzione dei programmi. Negli ultimi dieci anni sono stati condotti numerosi studi da ricercatori per chiarire al meglio il ruolo della mutazione nell'ambito degli algoritmi genetici. Questi studi hanno rivolto l'attenzione sul fatto che il punto fondamentale nell'algoritmo non è tanto nel dare maggiore importanza all'incrocio o la mutazione quanto trovare il giusto equilibrio tra incrocio, mutazione, selezione, dai particolari della funzione fitness e infine dalla codifica.

Secondo Mitchell [16] infine la direzione giusta da seguire in questo genere di studi è la ricerca di un modo per fare adattare dall'algoritmo genetico stesso le probabilità di incrocio e di mutazione nel corso della ricerca.

4.5 Auto-localizzazione: Algoritmi genetici e osservatori

Il sistema che si sta considerando può essere desritto dal seguente modello a tempo discreto:

$$\begin{cases} x_{k+1} &= f(x_k, u_k, \delta_k) \\ a_{k+1} &= a_k \\ y_k &= h(x_k, a_k, w_k) \end{cases} \tag{4.9}$$

La posizione del robot x al tempo $k+1$ è legata alla x_k, all' ingresso u_k e al rumore sullo stato δ_k da una funzione non lineare. In tale rappresentazione del modello gli a_i rappresentano delle caratteristiche geometriche dell'ambiente le quali non si modificano nel tempo e non sono note. Infine l'uscita y_k rappresenta le misure effettuate dai sensori al passo k, che si assume siano affette da un certo errore di misura w_k. Lo scopo fondamentale dell' *auto-localizzazione* è quello di fornire una stima dello stato $\hat{x}_k$ e dei parametri $\hat{a}_i$ minimizzando gli effetti dei disturbi sullo stato e sull'uscita. Queste stime possono essere effettuate mediante *osservatori*, dispositivi in grado, dati in ingresso l'ingresso u_k del sistema e l'uscita y_k, di restituire la stima $\hat{x}_k$ e $\hat{a}_k$. Il modello dell'osservatore può essere descritto come segue:

$$\begin{cases} \hat{x}_{k+1} &= f(\hat{x}_k, u_k) + v_1(y_k, \hat{y}_k) \\ \hat{a}_{k+1} &= \hat{a}_k + v_2(y_k, \hat{y}_k) \end{cases} \tag{4.10}$$

La (4.10) rappresenta la struttura tipica di un osservatore. La prima parte è proprio la copia esatta del sistema (4.9), e la seconda è un termine correttivo che dipende dall' *innovazione*. In figura 4.2 si riporta lo schema a blocchi di questo osservatore.

Per la progettazione di tale osservatore spesso si utilizza la tecnica del *filtro di Kalman* (per un'analisi più dettagliata si veda il capitolo 3). In questo lavoro di tesi si utilizza invece un approccio totalmente diverso basato sugli algoritmi genetici.

Nella maggior parte dei lavori sugli algoritmi genetici, questi vengono visti come una tecnica di ottimizzazione a problemi di diversa natura, dalla biologia, l'ingegneria fino alla matematica. In questo lavoro di tesi si è voluta

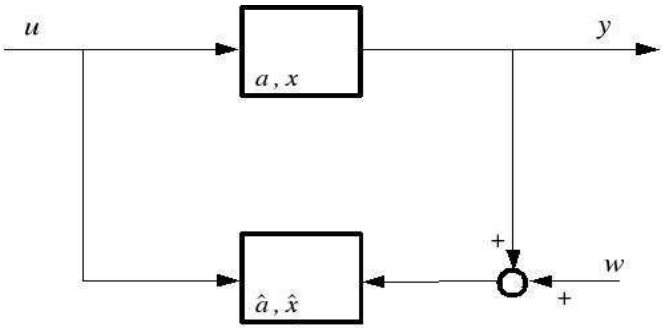

Figura 4.2: Diagramma a blocchi di un osservatore

dare un'interpretazione leggermente diversa: gli algoritmi genetici vengono visti come osservatori per la stima dello stato che minimizzino gli errori di misura. Nel diagramma seguente viene descritta l'analogia tra gli algoritmi genetici e gli osservatori.

A tale fine si suddivide il percorso del robot in step, e si suppone che al primo

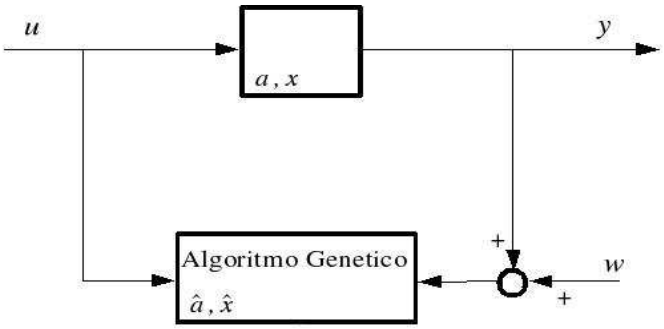

Figura 4.3: Analogia tra osservatori e algoritmi genetici

step il robot conosca perfettamente lo stato iniziale. Tale assunzione è lecita in quanto nel momento dell'inizializzazione del robot viene anche fissato il sistema di riferimento centrato proprio nel punto in cui si trova il robot. Si consideri ora un generico step $k+1$. In questo intervallo il robot si sposta da

una posizione iniziale x_k supposta corretta dal momento che rappresenta la stima migliore al passo k verso uno stato finale x_{k+1} necessariamente corrotto da disturbi ed errori di misura. La procedura si può riassumere in passi:

- Acquisizione nello stato iniziale dei dati sensoriali relativi all'ambiente.
- Generazione di un certo numero di correzioni casuali Δx Δy da applicare alle coordinate del robot alla fine dello step.
- A partire da tutte le nuove coordinate calcolate al passo precedente vengono individuate le possibili coordinate del muro.
- In questa fase parte il vero e proprio algoritmo genetico. L'insieme delle correzioni Δx e Δy rappresenta la *popolazione iniziale*. Essa viene valutata mediante una *funzione fitness* e fatta evolvere per più generazioni finchè non si arriva alla *correzione migliore* da applicare alle coordinate finali del robot.

Prima di iniziare con l'analisi dei passi dell'algoritmo genetico e della funzione fitness è utile tenere presente che il tipo di mappa utilizzata per la rappresentazione dell'ambiente è una ***griglia di occupazione*** costituita da celle di dimensioni 5×5 pollici.

Il primo passo della procedura consiste nell'individuazione delle celle che appartengono agli ostacoli dell'ambiente mediante le rilevazione dei sonar. Poichè il robot è provvisto di 16 sonars disposti sulla torretta e quindi un sonar e il suo successivo formano un angolo di $22,5°$, si è scelto di effettuare 10 rotazioni della torretta di $2,5°$ ognuna così da rendere più continua e robusta l'informazione sensoriale. Infatti anche se una lettura sonar dovesse essere corrotta da un disturbo elevato o anche se l'onda non dovesse ritornare indietro a causa di fattori geometrici e di scarsa precisione degli strumenti, allora si hanno comunque buone probabilità che un altro sonar riesca a individuare una determinata cella. Inoltre si hanno un numero elevato di informazioni e questo, come verrà discusso nel capitolo seguente, è molto utile sia per la localizzazione che per l'esplorazione. Per ogni cella i viene calcolato un valore $occ_i \in [0, 1]$ che valuta se quella cella è occupata o libera. Una volta arrivato alla fine dello step, viene generato casualmente un vettore di correzioni $\Delta x, \Delta y$. Le coordinate del robot alla fine dello step vengono poi modificate sulla base delle singole correzioni e a partire dalle nuove coordinate vengono di nuovo individuate le celle in cui ipoteticamente si potrebbero trovare gli ostacoli. A questo punto si effettua una sorta di *matching* tra le singole mappe locali calcolate alla fine dello step e quella "corretta" individuata all'inizio dello step. Questo *matching* viene effettuato mediante la ***funzione***

fitness. L'idea che sta alla base di questa euristica è molto semplice e intuitiva: viene fatta una semplice somma delle celle che coincidono tra le singole mappe locali finali e la mappa locale iniziale. La funzione fitness risulta così definita:

$$F = \sum_i occ_i \quad \forall i \tag{4.11}$$

Questa euristica dà in qualche modo la "*bontà*" di una soluzione,una particolare coppia (Δx, Δy), rispetto ad un'altra. Una volta selezionati gli *individui* migliori, si fanno *incrociare* dando luogo così a nuovi *individui* che appartengono ad una nuova generazione. Questa procedura si itera per un certo numero di generazione, finchè poi l'algoritmo convergerà ad una unica soluzione che sarà quella che rende massima la funzione fitness.

Questa tecnica può essere interpretata da un punto di vista leggermente diverso. Infatti trovare la ***soluzione migliore*** che massimizza la (4.11) vuol dire stimare lo stato x minimizzando eventuali errori. In particolare dalla (4.11) si evince che ciò che implicitamente fa l'algoritmo genetico e quello di individuare ad ogni passo k la stima $\hat{x}_k$ che, in un intorno delle letture odometriche $\hat{x}_{o,k}$, *minimizzi* lo scarto tra le letture sonar attese e reali. Queste in media sono minime quando $\hat{x}_k = x_k$ Appare subito evidente la differenza sostianzale tra una tecnica basata sull'utilizzo di un algoritmo genetico così come viene formulato in questa sede, e una tecnica basata sul filtro di Kalman. Quest'ultima è una tecnica di ***filtraggio ottimo ai minimi quadrati*** che in sostanza ha come scopo quello di minimizzare ***l'errore quadratico medio***. Si trova, in altre parole il $\min E[\sum_k ||x_k - \hat{x}_k||^2]$. Una ulteriore importante differenza tra queste due tecniche consiste nel fatto che mentre nel caso di filtri ottimi è necessaria una descrizione esatta del modello nel caso di utilizzo di algoritmi genetici le relazioni dinamiche del sistema possono anche non essere note. Infatti, mentre nel primo caso l'osservatore viene costruito facendo una copia esatta del modello con l'aggiunta di un termine correttivo, nel secondo caso questa copia non può essere fatta in quanto non si conosce il modello. Tuttavia il comportamento evolutivo e adattativo dell'algoritmo ha l'effetto di simulare la dinamica del modello e di fare in modo che l'*uscita attesa* sia il più possibile vicina all'*uscita reale*. Se da un lato il filtro di Kalman è ottimo solo per modelli lineari e con errore gaussiano (e quindi non è ottimo in questo caso in cui la dinamica è non lineare), il funzionamento degli algoritmi genetici presenta ancora parecchi problemi aperti. Per questo motivo il confronto dei due metodi su base teorica richiede notevoli sviluppi mentre è senza dubbio interessante un confronto sperimentale.

Capitolo 5

Tecniche di esplorazione e pianificazione del moto

La ***navigazione*** è un processo attraverso il quale un robot determina un percorso o una traiettoria da un posto a un altro; in questo processo sono fondamentali le tecniche di stima di una certa posizione rispetto al *mondo conosciuto*. Esso è composto da *superfici* le cui posizioni relative una rispetto all'altra vengono rappresentate su una *mappa*.

Tale definizione pone subito in rielievo una questione molto importante: per una corretta navigazione autonoma di un robot è necessaria la conoscenza dell'ambiente in cui il robot opera, ad esempio una mappa, e inoltre è necessaria una abilità di autolocalizzarsi allo scopo di navigare tra posizioni arbitrarie. Proprio per questa ragione nella maggior parte dei lavori sulla robotica mobile i problemi di localizzazione e ricostruzione della mappa vengono affrontati in modo separato. Se infatti si opera in un ambiente sconosciuto senza l'intervento umano, il robot al fine di esplorare e ricostruire un ambiente sconosciuto ha bisogno di conoscere la sua esatta posizione e viceversa al fine di conoscere la sua posizione, è necessaria una mappa dell'ambiente.
In definitiva, per risolvere il problema dello SLAM, sono necessari tre aspetti importanti:

- *Auto-localizzazione*
- *Acquisizione della mappa*
- *Esplorazione*

Per quanto riguarda il primo punto, esso stabilisce le tecniche utilizzate per fare in modo che il robot, sulla base dei valori provenienti dai sensori, si riesca

a localizzare correttamente correggendo perciò gli errori dei sensori presenti nelle misure effettuate. Questo aspetto è stato oggetto di analisi nel paragrafo 4.5. Per quanto, invece, concerne il secondo aspetto, cioè l'acquisizione della mappa, viene presentata una trattazione dettagliata nel lavoro di Randelli [20]. L'analisi delle metodologie per l'esplorazione, infine, sarà oggetto dei paragrafi successivi.

5.1 L'esplorazione nel problema dello SLAM

Nel problema dello SLAM, come già visto in precedenza, un robot viene fatto navigare in un ambiente ad esso non noto, al fine di ricostruire una mappa. Si pone perciò in evidenza un fatto cruciale: il robot deve essere in grado non solo di muoversi correttamente tra i vari ostacoli, ma anche di riconoscere le zone che ha già esplorato e capire se esistono nell'ambiente in cui opera zone non ancora visitate. Tale ricerca deve ovviamente essere esaustiva, ovvero non deve tralasciare nulla dell'ambiente; inoltre a tale scopo il robot ha a disposizione solo i dati provenienti dai sensori. Il problema perciò diventa quello di riuscire ad acquisire un numero maggiore di informazioni dall'ambiente e interpretarli nel modo migliore possibile.
D'altra parte per riconoscere correttamente se una zona è stata già esplorata è necessario che il robot si localizzi correttamente. Risulta chiaro perciò il forte legame che c'è tra l'*auto-localizzazione* e l'*esplorazione*. Questo legame può essere visto in ambo i sensi. Infatti per scoprire nuove zone dell'ambiente è necessario che il robot sappia con precisione dove si trova, viceversa al fine di una buona localizzazione, è necessaria una buona tecnica di esplorazione finalizzata all'acquisizione di nuove informazioni provenienti dai sensori.

Esistono diverse strategie di esplorazione in letteratura. Le principali sono:

- *Controllo umano*
- *Controllo reattivo*
- *Approaching the unknown*
- *Startegie di ricerca ottima*

Nel primo caso il robot è guidato nell'ambiente da un operatore umano; questo approccio richiede inevitabilmente l'intervento umano nel processo di costruzione della mappa, ed è perciò inapplicabile nel caso in cui si richiede

una totale auotonomia al robot. Una valida alternativa a questo approccio è data dall'utilizzo di strategie basate sul comportamento reattivo o su una esplorazione random dell'ambiente. Questo tipo di approccio è di gran lunga il più robusto rispetto alle altre metodologie ma al contempo non può garantire una completa esplorazione di un ambiente di medie dimensioni. Un classico esempio di comportamento reattivo è dato da strategie del tipo ***wall-following***, ma non risulta efficace in ambienti aperti; un'esplorazione di tipo random invece non garantisce buoni risultati se l'ambiente è abbastanza complesso.
La strategia conosciuta come *approaching the unknown*, consiste nel scegliere tra tutte le direzioni possibili che può percorrere il robot, quella verso l'ambiente sul quale si hanno meno informazioni possibili. L'acquisizione della mappa guida questo processo di esplorazione mentre contemporaneamente nuove informazioni sensoriali sono utilizzate per aggiornare la mappa. L'idea che sta alla base di questa tecnica è che il robot si muove verso l'estremo della mappa esistente e utilizza le informazioni provenienti dai sensori per individuare territori inesplorati. La principale differenza con i metodi descritti in precedenza è che non richiede nè una alta precisione sensoriale nè assunzioni o semplificazioni dell' ambiente.
Un ulteriore metodo di esplorazione è rappresentato dalle *strategie di ricerca ottima*. Questa tecnica alle volte potrebbe risultare poco vantaggiosa qualora non si hanno a disposizione una quantità sufficiente di informazioni o se il costo dell'ottimizzazione a livello computazionale risulta troppo elevato. In questo lavoro di tesi si cerca di trovare una sintesi di alcune tecniche di esplorazione. Si divide l'esplorazione in due livelli gerarchici:

- Basso livello
- Alto livello

La prima riguarda tutte le metodologie utilizzate nell'esplorare un singolo ostacolo definito da una spezzata chiusa (può essere per esempio un muro esterno di un ambiente qualsiasi o anche una colonna).
Per quanto riguarda il secondo punto, esso riguarda le tecniche usate per l'individuazione di nuovi ostacoli e il metodo per raggiungerli evitando eventuali ostacoli già esplorati.

5.2 Esplorazione a basso livello

A basso livello, una volta individuata una zona da esplorare, si applica una strategia di tipo ***wall-following***, ovvero il robot segue una traiettoria parallela al muro nella direzione tale che l'ostacolo si trova sempre alla destra

del robot. In questa fase avviene la vera e propria acquisizione della mappa e vengono aggiornati i sensori.
Il comportamento iniziale del robot è quello di individuare la direzione nella quale la distanza tra un eventuale ostacolo e il robot stesso è minima. Appena i sonar rilevano un valore inferiore ad una certa soglia (9 pollici), si attiva una certa procedura che fa sì che il robot si mantiene ad una certa distanza dall'ostacolo. Appena, per qualsiasi motivo per esempio una curva, il robot si allontana o si avvicina troppo all'ostacolo vengono attivati dei controlli sulle ruote che lo portano a rientrare nel range definito. Quanto detto può essere riassunto nei seguenti passi:

- Vai avanti
- Allontanati dall'ostacolo se le letture sonar sono inferiori a 9
- Avvicinati all'ostacolo se le letture sonar sono maggiori di 13

In letteratura si trovano approcci simili al problema dell'esplorazione e in particolare al comportamento reattivo. Per esempio in Nehmzow (1992), al fine di ottenere un controllo per evitare gli ostacoli, vengono implementate nel robot solo due regole istintive, una per insegnargli a muoversi velocemente se i i valori provenienti dai sensori rientrano in un certo range, e un'altra per insegnargli ad evitare l'ostacolo entro un certo range di sensori. Alcune volte tuttavia questo comportamento potrebbe non essere molto utile ai fini di una buona ricerca di zone nuove. Se infatti due zone disgiunte fossero però molto vicine, il robot, nell'analizzare una di esse rischierebbe di "perdersi" e di passare subito alla seconda senza prima terminare la prima. Con l'inserimento della terza regola oltre a evitare collisioni con l'ostacolo riuscirebbe anche a rimanere il più possibile nell'intorno della traiettoria ideale, ovvero quella parallela all'ostacolo.
Questo tipo di comportamento viene eseguito in real-time e ovviamente deve essere in grado di gestire situazioni di curva le più generali possibili. Non si fanno perciò assunzioni sulla topologia degli ostacoli e il controllo viene fatto in modo adattativo. A questo scopo si è messo a punto un sistema che riesce a riconoscere se il robot si trova in curva, se essa è verso destra o verso sinistra e se invece la zona che sta esplorando è rettilinea. In questo modo pur non avendo nessuna conoscenza su che tipo di curva si sta affrontando si riesce a gestire l'intera curva in maniere del tutto autonoma e indipendente. É stato possibile realizzare ciò grazie a un controllo che, una volta che il robot si trova all'interno del range $[9, 13]$, fa in modo che rimanga all'interno senza uscire fuori. Questo vuol dire che se i sonar dovessero per qualche motivo rilevare valori minori o maggiori di determinate soglie allora il robot si troverebbe necessariamente in prossimità di una curva. A questo

punto una volta entrato nella sotto-procedura relativa alla curva (destra o sinistra a seconda se i sonar siano maggiori o minori) se ne esce solo a curva terminata. Si sceglie, ad ogni step di scansione e aggiornamento dei sensori, la direzione del sonar che restituisce un valore minimo modificata di un certo angolo prefissato, finchè il valore minimo dei sonar non diventa minore di una certa soglia piuttosto bassa. Successivamente si sceglie la direzione opposta al sonar con valore minimo modificata ancora una volta di un certo angolo fissato. Una volta entrato nel range "buono" di valori, il robot si calcola la direzione perpendicolare alla direzione del sonar minimo, dove per sonar minimo si intende il sonar che restituisce il valore minimo tra tutti i 16 sonar (per quanto riguarda la curva a sinistra ovviamente le fasi sono invertite). A questo punto la sotto-procedura relativa alla curva è terminata e il robot attiva la procedura per il controllo sulla traiettoria ideale. Questo controllo è dinamico, real-time e ancora una volta non utilizza alcun tipo di assunzione. Si immagina di considerare un singolo step del giro di scansione e aggiornamento. Sulla base dei valori del sonar minimo all'inizio dello step e alla fine si calcola la differenza in valore assoluto e che rappresenta lo scostamento rispetto alla normale all'ostacolo. Si ricava poi l'angolo α formato da tale scostamento e dalla traiettoria del robot seguita allo step k. Sia b la traiettoria del robot definita come $b = \sqrt{(x_i - x_f)^2 + (y_i - y_f)^2}$, con (x_i, y_i) le coordinate iniziali del robot e (x_f, y_f) quelle finali. Sia inoltre a lo scostamento rispetto alla traiettoria ideale, allora:

$$\alpha = 90 - \arccos\left(\frac{a}{b}\right)$$

Questo angolo rappresenta proprio l'angolo da dare alle ruote per minimizzare lo spostamento effettivo tra la traiettoria reale e quella ideale. Questo controllo risulta necessario a causa del fatto che i sonar sono 16 posizionati lungo la circonferenza del robot. Perciò quando consideriamo per esempio la direzione del sonar minimo, anche se in linea teorica, se cioè avessimo a disposizione infiniti sonar sulla circonferenza, tale direzione è proprio la perpendicolare all'ostacolo, effettivamente poi la direzione non sarà esattamente perpendicolare. Quindi poichè la traiettoria da seguire viene calcolata proprio a partire dalla direzione del sonar minimo aggiungendo 90°, si deduce chiaramente che essa non potrà mai essere perfettamente parallela all'ostacolo e quindi esisterà sicuramente un certo istante in cui il robot uscirà dal range buono nonostante si trovi ad analizzare un tratto rettilineo dell'ostacolo. Da qui la necessità di effettuare questo controllo. A regime infatti $a \ll b$ e perciò $\arccos\left(\frac{a}{b}\right) \to 90°$ e $\alpha \to 0$.

Terminato il giro di scansione dell'ostacolo si avvia una procedura che consente al robot di riconoscere l'ostacolo come zona esplorata e di individuare le eventuali zone inesplorate. Anche in questo caso appare molto evidente la

necessità di una buona auto-localizzazione. Se il robot si localizzasse male e quindi si "perdesse" risulta chiaro che non riuscirebbe a riconoscere le zone sulle quali è già passato nonostante ci passi realmente. Quindi non riuscirebbe a chiudere il giro correttamente e attivare la procedura di esplorazione ad alto livello.

5.3 Esplorazione ad alto livello

Le procedure descritte nel paragrafo precedente riguardano l'esplorazione di un singolo ostacolo all'interno della mappa. Queste procedure sono ovviamente incomplete; anche ammesso che il robot riesca a terminare correttamente tutte le procedure a basso livello, esse darebbero solo un'informazione parziale rispetto all'ambiente. Infatti esse fornirebbero solo la mappa relativa a quel singolo ostacolo, tralasciando magari una parte cospicua dell'ambiente. Risulta evidente da questa analisi preliminare la necessità di implementare strategie di ricerca di ostacoli nuovi ancora da esplorare. Questo implica sostanzialmente due fattori:

- La necessità di lavorare in ambienti di piccole o medie dimensioni
- La necessità di riconoscere se un ostacolo sia effettivamente da esplorare oppure sia già stato esplorato in precedenza.

La prima condizione è necessaria perchè i sonar, che sono gli unici sensori "esterni" che possiede il robot (eccetto i bumper), lavorano bene solo entro un certo limite, superato il quale l'onda non ritorna al sonar. La seconda è necessaria perchè si potrebbero verificare situazioni in cui mentre il robot sta scandendo un certo ostacolo potrebbe rilevarne un altro e non riconoscere se è stato già esplorato. Tuttavia non basterebbe solo questo tipo di analisi. Per spiegare le ragioni di questo fatto si può prendere un esempio. Se il robot deve esplorare e ricostruire la mappa di un ambiente costituito da una parete esterna e da due colonne al centro e se ha già esplorato la parete esterna ovviamente il robot si dirigerà nella direzione di una delle due colonne. Si denoti ora con la lettera A la prima colonna che incontra e con la lettera B la seconda. Supponiamo anche che la colonna A e la colonna B si trovano allineate rispetto alla direzione in cui procede il robot. Allora succederà che il robot inizierà a esplorare la colonna A e terminerà esattamente nella stessa posizione in cui ha iniziato. Ciò implica che nel momento in cui si attivano le procedure ad alto livello il robot individua la colonna B come zona da esplorare e si muove nella direzione di B senza riconoscere che il percorso in realtà è ostruito dalla colonna A. Il problema inoltre nasce anche dal fatto che, mentre nelle procedure a basso livello avviene una scansione dell'ambiente

e l'aggiornamento dei sensori al fine della ricostruzione dell'ambiente, nelle procedure ad alto livello tutto ciò non avviene perchè fondamentalmente non vi è alcuna necessità; inoltre inserire anche l'aggiornamento e la scansione potrebbe risultare troppo costoso a livello computazionale e appesantirebbe il movimento del robot.
Da qui la necessità di mettere a punto una pianificazione del moto finalizzata a evitare gli ostacoli presenti nella traiettoria che unisce il punto iniziale da cui parte il robot e l'ostacolo finale (verrà descrittà nel paragrafo successivo).

Tornando all'esplorazione, si descrive ora più nel dettaglio la tecnica utilizzata per riconoscere ostacoli ancora non esplorati e quelli già esplorati. A titolo preliminare bisogna precisare che si utilizza allo scopo della rappresentazione grafica dell'ambiente una griglia di occupazione, una mappa costituita da celle (pixel) di dimensioni 5×5 pollici ($1inch \approx 2.5cm$). Dal punto di vista implementativo si è creata una struttura dinamica denominata cella nella quale sono presenti alcuni campi. Ai fini dell'esplorazione è utile considerare tre campi della struttura cella che sono il campo "visited", il campo "new_obs" e il campo "wall". Il primo indica che la cella considerata appartiene alla traiettoria seguita dal robot nel corso della scansione; il secondo indica che le celle sono rilevate dal robot ma non appartengono all'ostacolo che si sta considerando e infine il terzo indica se la cella considerata appartiene all'ostacolo che si sta esplorando. Questa tecnica si svolge sostanzialmente in due fasi:

- Individuazione traiettoria del robot e rilevazione di celle esterne all'ostacolo
- Verifica che le celle individuate al passo precedente non appartengano ad ostacoli esplorati

La prima fase si svolge contemporaneamente al giro di scansione. Consiste nell'aggiornare il campo *visited* nelle celle dove ad ogni step si trova il robot e nel contempo aggiornare il campo new_obs delle celle individuate dal sonar opposto al minimo. Il fatto che a individuare tali celle sia il sonar opposto al sonar minimo, proprio grazie alle regole istintive descritte nel paragrafo 5.2, garantisce che nella maggior parte dei casi siano celle non appartenenti all'ostacolo che si sta esplorando in quel momento.
Durante la seconda fase, che avviene soltanto a giro ultimato, il robot analizza tutte le celle appartenenti alla mappa che abbiano il campo new_obs pari a uno ma che non appartengano alla traiettoria del robot (che non hanno perciò *visited* pari a uno) e a partire da quelle celle e per ognuna di esse, viene fatta partire una croce che scandisce le celle nelle quattro direzioni: verso l'alto, verso il basso, a destra e a sinistra.

L'idea qui è quella di considerare come zona visitata quella delimitata dalla traiettoria seguita dal robot, cioè dalle celle con il campo *visited* pari a uno. Il problema che appare subito evidente è quale porzione di spazio considerare, se la parte interna alla traiettoria o quella esterna. Nel caso di scansione della parete esterna, infatti, la zona esplorata sarà ovviamente quella esterna, nel caso di ostacolo interno, invece, la parte da considerare già esplorata è chiaramente quella interna. Per risolvere questo problema si può fare affidamento su due fattori: il fatto che il robot si muove evitando l'ostacolo tenendolo sempre sulla destra e la capacità di discriminare con buona precisione le curve a destra o a sinistra e una zona rettilinea. Infatti, poichè il robot tiene l'ostacolo sempre alla sua destra, allora il giro verrà effettuato sempre in senso *orario*, se si sta scandendo un ostacolo interno, e in senso *antiorario*, se il robot sta esplorando una parete esterna. Perciò il punto fondamentale è insegnare al robot a riconoscere il verso in cui sta effettuando il giro. A questo punto entra in gioco il secondo fattore importante per risolvere il problema. Poichè il robot è in grado non solo di riconoscere perfettamente le curve ma anche di discriminare se la curva è verso destra o verso sinistra, allora se per esempio il robot effettua un numero maggiore di curve a destra questo implica necessariamente che il verso di percorrenza è *orario* e viceversa. Ciò è garantito dal fatto puramente geometrico che gli ostacoli sono rappresentati da spezzate chiuse. A questo punto il robot possiede tutti gli elementi necessari per una corretta ed esaustiva esplorazione dell'ambiente. La tecnica utilizzata si può riassumere nei seguenti passi:

- Si costruisce una croce a partire dalle celle col campo "new_obs" uguale a uno scandendo le celle adiacenti nelle quattro direzioni finchè non si arriva alle celle con il campo "visited " pari a uno (cioè che appartengono alla traiettoria del robot) oppure finchè non si arriva alla fine della mappa.

- Se tutti gli estremi della *croce* sono celle che appartengono alla traiettoria del robot e se contemporaneamente il verso di rotazione è concorde per ognuna di quelle celle (in particolare se il verso è *orario*), allora questo implica che le celle che si stanno considerando appartengono ad una zona già esplorata.

- Se invece almeno una cella all'estremo della croce appartiene alla traiettoria del robot e contemporaneamente almeno una arriva alla fine dalla mappa, e inoltre se il robot sa riconoscere se la parete esterna è già stata esplorata, allora può dedurre chiaramente che anche queste celle appartengono ad una zona già esplorata.

- In tutti gli altri casi le celle considerate appartengono a zone dell'ambiente non ancora esplorate e tra tutte il robot sceglie quella che ha la distanza minima dalla posizione in cui si trova.

Per quanto riguarda il primo punto, esso non presenta grosse difficoltà, dal momento che il robot possiede già tutte le informazioni necessarie e non si possono verificare ambiguità di alcun genere. Per quanto concerne il secondo punto, invece, il robot si può trovare in situazioni in cui non è in grado di prendere una decisione corretta e in modo univoco. Infatti se non fosse in grado di distinguere la parete esterna da tutti gli altri ostacoli che stanno all'interno, allora considerando solo la tecnica basata sulla costruzione della croce, non riuscirebbe a capire se la cella considerata è da esplorare o è già stata esplorata. Il problema si risolve facilmente grazie al fatto che il robot riesce a capire se sta eseguendo la scansione in senso orario o antiorario. Infatti poichè la parete esterna è l'unico ostacolo che viene esplorato in senso antiorario, il robot la può subito etichettare con una variabile flag. Perciò se questa variabile è impostata ad un valore pari a uno vuol dire che la parete esterna è stata già esplorata, uguale a zero viceversa. Quindi se tra le celle estreme della croce è presente almeno una che ha il campo "visited" uguale a uno e almeno uno che "esce" dalla mappa e se la variabile flag è impostata a uno, allora vuol dire necessariamente che si tratta di una zona già esplorata; se invece la variabile flag ha un valore pari a zero, allora vuol dire che non è stata ancora esplorata la parete esterna e che quindi si tratta di una zona da esplorare.
Questo metodo è molto importante sostanzialmente per due motivi. Per prima cosa, attraverso queste tecniche si riesce a dare al robot un ***metodo di apprendimento*** dell'ambiente anche dal punto di vista più ad alto livello e più approfondito sulla semantica degli ostacoli. Non riconosce solo se una cella è occupata o è vuota, ma riesce a ricostruire anche la geometria dell'ostacolo (il numero di curve a destra o a sinistra), riesce a distinguere se l'oggetto è una parete esterna o interna, riesce insomma a classificare gli ostacoli sulla base delle loro caratteristiche. La cosa potente di questo metodo è che non solo il robot apprende il modo per classificare ma anche la strutture delle categorie stesse e tutto ciò viene effettuato attraverso la semplice elaborazione dei dati provenienti dai sensori.
In secondo luogo, con questo metodo si riesce ad effettuare una esplorazione esaustiva dell'ambiente. In altre parole il robot è in grado di acquisire tutti gli ostacoli dell'ambiente senza tralasciare nulla. Ovviamente quest'ultimo aspetto è quello veramente rilevante ai fini del problema dello SLAM.

5.4 Pianificazione del moto

Resta ancora però da chiarire bene che tecnica è utilizzata al fine di evitare gli ostacoli presenti nel percorso scelto dal robot per raggiungere le zone da esplorare. Esistono in letteratura diversi metodi per la pianificazione del moto nell'ambito della robotica mobile. Possono essere classificati in tre metodi:

- *Carta stradale*
- *Decomposizione in celle*
- *Funzioni potenziali*

Tutti i metodi appartenenti alla prima classe riconducono il problema alla ricerca di un percorso su grafo definito oppurtunamente. I *grafi di visibilità* e i *diagrammi di Voronoi* sono esempi di metodi appartenenti alla prima classe.
Nei metodi che utilzzano la *decomposizione in celle* lo spazio libero viene decomposto in regioni ciascuna delle quali viene poi associata ad un grafo, i cui nodi vengono collegati se e solo se le celle corrispondenti sono adiacenti. Inoltre la *decomposizione in celle* si può ulteriormente suddividere in *esatta* o *approssimata*. Con la prima si può ottenere una pianificazione esatta e rappresenta quindi una soluzione completa al problema; al contrario la seconda non fornisce una soluzione completa al problema della pianificazione ma in molti casi risultà di notevole utilità pratica.
Infine il metodo dei potenziali ha una caratteristica principale che lo differenzia dai metodi sopra enunciati: la "*località*". In generale, infatti, il potenziale in un dato punto dello spazio dipende solo dagli ostacoli presenti in un intorno del punto dove si trova il robot. Una ulteriore caratteristica importante di questo metodo è il fatto che è molto più veloce rispetto ad altri metodi ed è quindi facilmente applicabile *on-line*. L'idea che sta alla base del *metodo dei potenziali* è quella di associare un potenziale attrattivo al punto di arrivo (*goal*) e uno repulsivo per ogni ostacolo. La funzione potenziale totale è quindi la somma di tutte i vari potenziali. Tutte queste sono tecniche che forniscono soluzioni pressochè complete considerando ambienti del tutto generali dal punto di vista topologico. Per come si è approcciato il problema dello SLAM in questo lavoro di tesi vengono applicate tecniche di pianificazione diverse dalle precedenti.

Viene calcolato il percorso diretto che il robot dovrebbe percorrere allo scopo di raggiungere la zona da esplorare. Se tale percorso non interseca nessun ostacolo, allora la strada è libera e il percorso è ovviamente quello minimo. Viceversa se il percorso calcolato interseca un ostacolo, allora i

punti di intersezione saranno almeno due. Sia a_0 il primo punto di intersezione. Partendo da a_0 si scorre lungo la traiettoria che il robot ha seguito durante la fase di esplorazione dell'ostacolo preso in considerazione finchè non si ha la visuale libera verso la cella di destinazione. Denotiamo con a_f tale punto. A questo punto si effettua un metodo analogo al primo caso: prtendo da a_f si scorre all'indietro sulla traiettoria del robot finchè, al contrario del caso precedente, non si ha la visuale libera rispetto al punto di partenza. Si denoti con a_i questo punto. Per quanto riguarda i punti intermedi, essi vengono presi equidistanti dall'insieme dei punti appartenenti alla traiettoria del robot. Si hanno così un insieme di punti, o ***nodi***, l'unione dei quali rappresenta la traiettoria da seguire per evitare l'ostacolo. Questo metodo ovviamente risulta peggiore rispetto alle alre tecniche di pianificazione nell'ipotesi in cui gli ostacoli hanno forme molto "frastagliate", dal momento che in questo caso il robot sarebbe costretto a seguire il profilo dell'ostacolo per una porzione consistente dello stesso. Nel caso in cui l'ostacolo sia costituito da un numero di spezzate chiuse non molto elevato, se assume perciò configurazioni geometriche non troppo complesse, questo metodo risulta non solo molto rapido e veloce dal punto di vista computazionale ma fornisce anche un possibile percorso minimo, in quanto il robot evita l'ostacolo muovendosi il più possibile vicino a esso.

Capitolo 6

Risultati sperimentali

L'attività sperimentale è stata condotta sia tramite il simulatore *Nserver*, sia utilizzando il *Nomad 150* in un ambiente reale, mostrato in fig.6.1. Tale ambiente, il piano del dipartimento di Ingegneria dell'Automazione della Facoltà di Ingegneria di Tor Vergata, è stato anche ricostruito sul simulatore, in modo da poter provare l'implementazione senza dover aspettare i tempi di attesa del robot, e passarla a quest'ultimo solo successivamente.

Figura 6.1: Dipartimento di Automatica

Si tratta di un rettangolo, con al centro un ostacolo (i bagni del dipartimento) con 4 porte, e cinto esternamente dalle mura degli uffici, con porte larghe 90cm, tranne due porte dei laboratori più grandi ai vertici. A causa della perdita di pacchetti TCP riscontrata utilizzando il sistema radio-modem *Mercury EN*, si è dovuto collegare il notebook direttamente al *Nomad 150*. Quest'ultimo ha un'autonomia limitata, e quindi si è ristretta l'area da esplorare ad un corridoio circolare, troncando la parte dei due corridoi di entrata e di uscita del piano. L'area non è completamente regolare, in quanto i bagni terminano con un emiciclo, e presenta alcune rientranze che rendono la ricostruzione grafica più complessa.

Il notebook utilizzato per i tests è equipaggiato con un processore Pentium 4 a 2.0GHz e 256MB di memoria RAM; *Nserver* gira su un sistema operativo Linux con kernel 2.4.

Gli esperimenti condotti sul sistema reale hanno messo ancora più in luce i limiti della soluzione proprosta con questo lavoro di tesi rispetto agli esperimenti fatti sul simulatore. Nell'ambiente simulato infatti non sono presenti errori relativi ai sensori. Questi errori si riflettono soprattutto nelle curve. Infatti a causa del fatto che il robot in questa fase della scansione si trova molto vicino al muro (ciò deriva direttamente dall'utilizzazione di regole istintive che portano il robot a seguire una traiettoria parallela al muro e molto vicina ad esso), allora quello che potrebbe accadere in curva è che il robot si trova in una configurazione tale che lo spigolo del muro si trova prorpio a metà tra due sonar facendo in modo tale che i sonar rilevino un valore sbagliato, non riuscendo perciò ad affrontare bene la curva. A questo problema, riscontrato solo nel *mondo reale*, si era riusciti a dare una soluzione soddisfacente: si aumentava in ogni stato la risoluzione dei sonar facendo ruotare la torretta, fino a prendere una quantità certamente ridondante di informazioni ma che desse maggiore credibilità ai dati provenienti dai sonar. Non sarebbe stato un grosso problema se si fosse implementato un controllo anche sulla torretta; la mancanza di tale controllo purtroppo però determina un errore che si accumula durante la scansione sulla posizione della torretta e conseguentemente sugli angoli dei sonar causando oltre che una errata ricostruzione della mappa anche una localizzazione scorretta del robot. Il compromesso che si è raggiunto in fase implementativa consiste nel fatto che viene effettuata la rotazione della torretta solo nello stato iniziale e finale di ogni singolo step. Ciò è stato fatto per garantire una quantità di informazione sufficiente per la localizzazione contenendo il più possibile gli errori, a scapito però di una buona precisione nell'affrontare le curve. Questo problema sul controllo della torretta si sarebbe potuto risolvere se per esempio il robot fosse equipaggiato con una bussola. Si sarebbe potuto infatti progettare un controllo di *tracking* facendo in maniera tale che la torretta fosse sempre orientata verso uno dei punti cardinali.

Per quanto riguarda la localizzazione molti sono stati i problemi che si sono dovuti risolvere. Il primo problema che è nato è stato proprio a livello implementativo su come effettuare la codifica delle correzioni odometriche. A tale fine si è scelto di utilizzare una codifica binaria a 16 bit (la prima metà per Δx e la seconda metà per Δy). Proprio a causa del tipo di codifica effettuata si è scelto di utilizzare una tecnica di incrocio a due punti. Infatti

con questa tecnica si hanno due sostanziali effetti: per prima cosa si riducono le "dipendenze posizionali" (si veda capitolo 4); inoltre si abbassa la probabilità che gli estremi di un cromosoma si conservino anche nelle generazioni successive. Nel caso in esame utilizzare un incrocio ad un punto singolo implicherebbe una bassa varietà di possibili correzioni. Inoltre allo scopo di garantire non solo un' *esplorazione verticale* che porta l'algoritmo ad una convergenza piuttosto rapida, ma anche una buona *esplorazione orizzontale*, atta a garantire una discreta variazione delle soluzioni al fine di evitare di incorrere in minimi locali, si è scelto l'utilizzo di una tecnica di selezione proporzionale all'idoneità con campionamento stocastico universale (CSU), mediante la serie di Boltzmann.

Un altro problema fondamentale è stata la scelta di come acquisire i dati provenienti da sensori e utilizzati per la valutazione della funzione fitness. In una fase iniziale si era scelto di utilizzare solo i dati provenienti da un numero ristretto di sonar (in particolare quelli che individuavano le celle più vicine appartenenti all'ostacolo preso in esame). Tuttavia è risultato immediatamente evidente che questo modo di procedere non dava una corretta informazione per la localizzazione. Infatti a causa del fatto che il matching tra le varie mappe locali rappresentative di tutte le possibili correzioni, e quindi la loro valutazione sulla base della funzione fitness, veniva effettuato solo alla fine dello step, l'algoritmo genetico tendeva a prediligere soluzioni che portassero il robot "indietro" rispetto alla sua effettiva posizione. Inoltre poichè l'esplorazione costringe il robot a stare molto vicino all'ostacolo e a causa inoltre del fatto che se l'onda di un sonar incontra una superficie piuttosto inclinata non riesce a tornare indietro, allora questo modo di procedere per acquisire i dati non è risultato molto vantaggioso. Si è scelto allora di utilizzare tutto l'ambiente per avere un numero maggiore di informazioni da elaborare. Tale acquisizione viene fatta a 360° con 176 rilevazioni dei sonar così da garantire una alta probabilità di correttezza delle misure. Inoltre in una versione preliminare dell'algoritmo ad ogni rilevazione sonar della stessa cella i veniva incrementata la variabile occ_i al fine di dare maggiore validità alla cella in esame. Questo approccio è quello utilizzato da Duckett in [8]. Nel caso preso in considerazione in questo lavoro tale approccio non ha fornito risultati buoni sempre a causa del tipo di tecnica utilizzata in fase di esplorazione. Infatti veniva dato un peso maggiore, nella valutazione dell'idoneità delle correzioni, a celle che si trovavano in prossimità del robot a causa anche di considerazioni geometriche. Questo fatto, esattamente come nel caso di acquisizione mediante un numero ristretto di sonar, faceva sì che l'algoritmo convergesse verso correzioni errate che tiravano il robot "indietro" rispetto alla sua posizione reale.

La soluzione a tutti questi problemi è stata, infine, quella di dare alla variabile *occ* una interpretazione *logica* (occupata - non occupata), i cui valori quindi appartengono all'intervallo discreto $0, 1$.

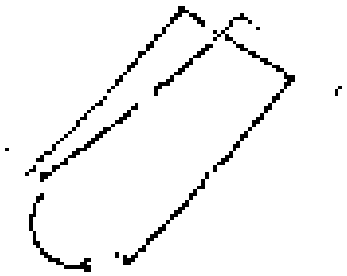

Figura 6.2: Rappresentazione della mappa senza algoritmo genetico.

In figura 6.2 viene mostrata la rappresentazione della mappa con una errata localizzazione del robot. Appare subito evidente il legame che c'è tra la localizzazione e l'acquisizione della mappa e l'esplorazione. Si può vedere chiaramente che se il robot sbaglia a stimare la sua posizione non riesce a chiudere il giro di scansione non permettendo la corretta applicazione di tutte le tecniche utilizzate per l'esplorazione nonchè la corretta aquisizione della mappa.

Per quanto riguarda l'esplorazione, a causa di alcuni problemi come per esempio l'autonomia limitata delle batterie del computer col quale sono stati condotti gli esperimenti, non è stato possibile testare l'efficienza delle tecniche descritte nel capitolo 5. Gli unici risultati sperimentali sono quelli ottenuti nell'ambiente simulativo. Si è testato l'algoritmo in diversi ambienti con configurazioni geometriche casuali e con un numero elevato di ostacoli. Nella figura 6.3 viene rappresentata una possibile configurazione di ostacoli all'interno di un ambiente chiuso.

In realtà questo è solo una delle possibili configurazioni nelle quali sono state testate le tecniche di esplorazione. La fase di scansione ed esplorazione inizia dal punto in cui si trova il robot (punto rosso). Rileva per prima cosa la parete esterna in quanto più vicina rispetto agli altri ostacoli. Una volta ultimato il giro esterno il robot individua l'ostacolo 2 e verifica che effettivamente abbia la visuale libera. Terminato il giro del secondo ostacolo il robot rileva l'ostacolo 3, ma poichè nella traiettoria è presente l'ostacolo 2

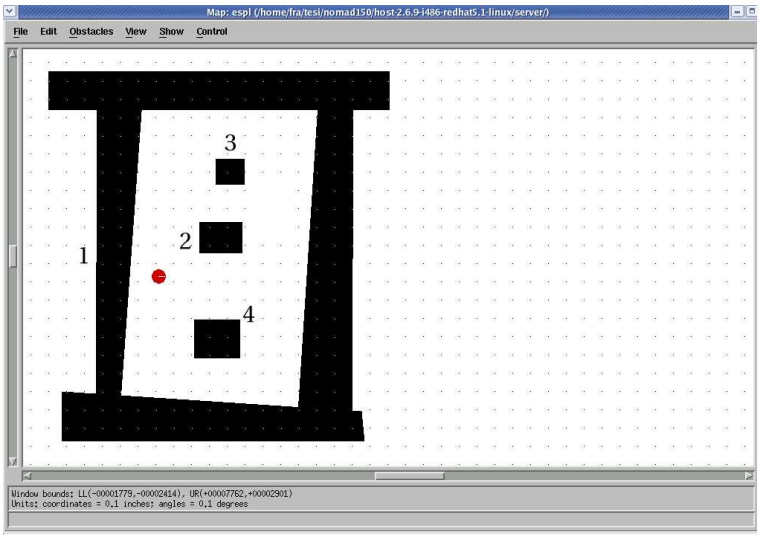

Figura 6.3: Rappresentazione dell'ambiente nel simulatore.

appena esplorato è necessario trovare un percorso alternativo per evitarlo. Terminato anche il giro intorno l'ostacolo 3 resta ancora da esplorare l'ostacolo 4 in basso a sinistra. Per raggiungere tale ostacolo il robot è costretto a evitare l'ostacolo 2. Per fare ciò vengono applicati gli algoritmi descritti al capitolo 5. Si considera il primo punto di intersezione e lo si fa scorrere lungo la traiettoria del robot finchè il percorso per raggiungere l'ostacolo 4 si libera. A questo punto si procede in modo analogo all'indietro sempre lungo la traiettoria del robot, per trovare il punto dal quale è possibile arrivare all'ostacolo 3 senza incorrere in nessun ostacolo. Per quanto riguarda il tratto intermedio si segue la traiettoria percorsa dal robot per esplorare l'ostacolo 2. In questo caso il tratto intermedio è semplicemente il segmento sinistro dell'ostacolo 2.

Infine nelle figure 6.4 e 6.5, vengono mostrate la mappe finali dell'ambiente ricostruito dall'algoritmo genetico in simulazione (fig:6.4),e dell'ambiente reale nel quale sono stati condotti gli esperimenti sempre mediante utilizzo di algoritmi genetici (fig: 6.5).

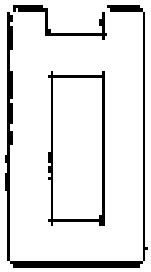

Figura 6.4: Rappresentazione grafica dell'ambiente su *Nserver* mediante algoritmi genetici.

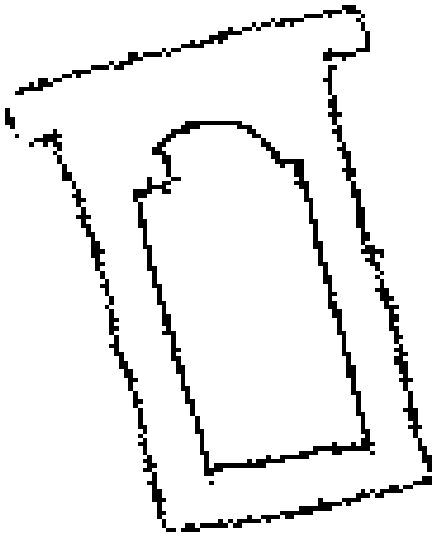

Figura 6.5: Rappresentazione grafica dell'ambiente reale (D.I.S.P.) mediante algoritmi genetici.

Capitolo 7

Conclusioni

Questo lavoro di tesi ha lo scopo di fornire una possibile soluzione al problema dello SLAM. Viene utilizzato un approccio totalmente diverso rispetto alle soluzioni classiche che fanno uso del filtro di Kalman. In particolare il problema della localizzazione, che può essere ricondotto al problema della stima dello stato del robot, viene affrontato non con una tecnica di filtraggio ottimo mediante osservatori, bensì attraverso l'implementazione di un algoritmo genetico. I punti di forza di questo approccio sono sostanzialmente due:

- L'implementazione di una versione *on-line* dell'algoritmo (in letteratura infatti è presente solo una soluzione che prevede una versione *off-line* dell'algoritmo).
- La sostanziale indipendenza nella tecnica del filtraggio da qualsiasi tipo di assunzioni e dal modello del sistema.

Tuttavia esistono dei forti limiti alla soluzione che viene proposta in questo lavoro.

Il limite fondamentale è proprio intrinseco all'utilizzo degli algoritmi genetici in quanto, nonostante siano stati fatti numerosi studi soprattutto negli ultimi anni, rimangono ancora molte questioni aperte, una delle quali è proprio la giustificazione formale e teorica del funzionamento degli algoritmi genetici. Non si riesce ancora a fare chiarezza sul meccanismo dell'evoluzione, come attraverso essa si possa convergere a soluzioni ottime.

Un ulteriore limite al lavoro svolto in questa sede risiede nel fatto che non viene effettuato alcun tipo di controllo sugli errori nel movimento della torretta. Non avere una corretta conoscenza della posizione effettiva della torretta

potrebbe implicare una pressochè errata interpretazione dei dati provenienti dai sensori, dal momento che i 16 sonar sono disposti proprio sulla torretta.

In secondo luogo, i dati sensoriali alle volte risultano del tutto insufficienti sia per una corretta localizzazione che per una efficace esplorazione. Proprio alla luce di quanto appena detto si sarebbero potute scegliere tecniche di esplorazione a basso livello diverse da quelle utilizzate in questo lavoro. Infatti a causa del fatto che, per quanto descritto nel capitolo 5, il robot si trova molto vicino all'ostacolo allora la maggior parte dei sonar puntano all'ostacolo da analizzare con un angolo troppo inclinato per far sì che le onde dei sonar riescano a ritornare correttamente. Se, per esempio, si fosse utilizzata una strategia di esplorazione diversa, non basata su un comportamento reattivo tipo *wall-following* bensì una tecnica che portava il robot a mantenere una traiettoria il più possibile equidistante dagli ostacoli presenti nell'ambiente, allora si sarebbero avuti a disposizione certamente un numero maggiore di informazioni sensoriali.

In tali direzioni si potrebbero indirizzare lavori futuri, verso un controllo sulla posizione della torretta e verso una migliore acquisizione dei dati sensoriali (eventualmente anche mediante altre tipologie di sensori come laser o infrarossi).
Inoltre al fine di migliorare l'esplorazione,possono essere fuse le metodologie utilizzate in questo lavoro con tecniche di intelligenza artificiale che si basano sul *riconoscimento di pattern*. Mediante tecniche di questo tipo, per esempio, è possibile far *apprendere* al robot la nozione di porta (aperta o chiusa), o per esempio di corridoio con lo scopo di incrementare la conoscenza semantica che il robot acquisisce durante l'esplorazione.

Bibliografia

[1] N.Ayache, Faugeras, *Maintaining a representation of the environment of a mobile robot*, IEEE Trans. Robot. Automat., vol. 5, 1989.

[2] M. Bucci, D. Circelli, *Algoritmi genetici e simulated annealing per la soluzione parallela di problemi di ottimizzazione combinatoriale*, Università degli studi di Pisa, Facoltà di Scienze Matematiche, Fisiche e Naturali, Corso di Laurea in Scienze dell'Informazione, 1996.

[3] R.Chatila, J-P Laumond, *Position referencing and consistant word modeling for mobile*, Proc. IEEE Int. Conf.Itell.Robots Syst., 1993.

[4] R.Cheeseman and P. Smith, *On the representation and estimation of spatial uncertainty*, Int. J.Robot. Res., 1986.

[5] Dartmouth College Computer Science Department, *Robo-Rats Locomotion: Car-type Drive*, http://www.cs.dartmouth.edu/%7Erobotlab/robotlab/courses/cs54-2001s/car.html.

[6] M.W.M.G.Dissanayake, P.Newman, S.Clark, H.F.Durrant-Whyte,M.Csorba, *A solution to the Simultaneous Localization and Map Building (SLAM) Problem*, IEEE Trans. Robotics Aut., vol. 17, 2001.

[7] T.Duckett, *Concurrent map building and self-localisation for mobile robot navigation*, University of Manchester, 2000.

[8] T.Duckett, *A genetic algorithm for simultaneous localization and mapping*, IEEE Proc. ICRA, 2003.

[9] H.F.Durrant-Whyte, *Uncertain geometry in robotics*, IEEE Trans. Robot. Automat., vol. 4, pp 23-31, 1988.

[10] G. Folino, *Algoritmi evolutivi e programmazione genetica: strategie di progettazione e parallelizzazione*, Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni, 2003.

[11] J.E.Guivant, E.M.Nebot, *Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation*, IEEE Trans. Robotics Aut., vol. 17, 2001.

[12] I.Inchingolo, *Algoritmi genetici*, Corso di Matematica, 2003.

[13] J.Leonard, H.F.Durrant-Whyte, *Directed Sonar Sensing for Mobile Robot Navigation*, Boston, MA: Kluwer Academic, 1992.

[14] Jean-Claude Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, second printing, 1991

[15] Martinelli Francesco *Pianificazione del moto dei robot*, Dispense del corso di Robotica Industriale, 1999.

[16] M.Mitchell, *Introduzione agli algoritmi genetici*, Apogeo srl, 1998.

[17] Nomadic Technologies Inc., *Nomad 200 Hardware Manual*, 1997.

[18] C. Pisani, *Gli algoritmi genetici ed alcune applicazioni*, Università degli Studi di Torino, 2001.

[19] Polaroid OEM Components Group, *Technical Specifications for 6500 Series Sonar Ranging Module*, 1999.

[20] Randelli Gabriele, *Approccio genetico allo SLAM Problem*, Tesi di Laurea in Ing.Informatica, Università degli studi di Roma, *Tor Vergata*,2004.

[21] W.D.Renken, *Concurrent localization and map building for mobile robots using ultrasonic sensors*, Proc. IEEE Int. Conf.Itell.Robots Syst., 1993.

[22] Rijswijk Institute of Technology, *Motorola 68HC11 Microcontroller*, http://www.hc11.demon.nl/thrsim11/index.htm.

[23] F.Romanelli, *Laboratorio di Automatica: Robot Nomad 150*, Tesi di Laurea in Ing.Informatica, Università di Tor Vergata, 2002.

[24] R.Smith, M.Self, P.Cheeseman, *Estimating uncertain spatial relationships in robotics*, in Autonomous Robot Vehicles, I.J.Cox e G.T.Wilfon, Eds,New York: Springer Verlag, 1990, pp.167-193.