UNIVERSITA' DEGLI STUDI DI ROMA
"TOR VERGATA"

FACOLTA' DI INGEGNERIA

Tesi di Laurea in Ingegneria Informatica

"Approccio genetico al problema dello SLAM"

Relatore
Ing. F. Martinelli

Laureando
Gabriele Randelli

Anno Accademico 2003/2004

Indice

1 Introduzione **2**

2 Nomad 150 **5**
2.1 Hardware 5
2.2 Software 8

3 Letteratura precedente inerente allo SLAM **10**

4 Algoritmi genetici **16**
4.1 Cenni sulla computazione evolutiva 16
4.2 Introduzione agli algoritmi genetici 17
4.3 Fondamenti teorici degli algoritmi genetici 22
4.4 Implementazione 26
4.4.1 Codifica 26
4.4.2 Selezione 29
4.4.3 Incrocio 31
4.4.4 Mutazione 32

5 SLAM Problem: analisi e soluzione genetica **33**
5.1 Il problema della localizzazione e dell'acquisizione della mappa 33
5.2 Un algoritmo genetico on-line per il problema dello SLAM . . 36

6 Risultati sperimentali **43**

7 Miglioramenti futuri e conclusioni **50**

Bibliografia **52**

Capitolo 1

Introduzione

Nel presente lavoro si vuole proporre un metodo risolutivo al problema dello *SLAM*[1], basato su un algoritmo genetico *online*. Lo *SLAM* nasce nella robotica mobile, e riguarda l'acquisizione e la localizzazione di tali sistemi in ambienti non noti a priori, al fine di avere a disposizione una conoscenza dell'ambiente di lavoro, fruibile in seguito per altre operazioni. In molte applicazioni di robotica infatti, è necessario avere una conoscenza riguardo il mondo esterno, che di solito viene fornita dai progettisti dei sistemi stessi. Si pensi, ad esempio, ai macchinari di catene di montaggio che devono lavorare su un prodotto particolare, alla pianificazione del moto di alcuni robot, sono tutte condizioni per le quali bisogna sapere dove si agisce. Nell'ambito dell'intelligenza artificiale, o AI, un sistema robotico, meglio un agente, che ha una conoscenza fornita a priori e a cui non viene fornito alcuno strumento di apprendimento, è un agente non autonomo, non in grado di adattarsi a circostanze diverse da quelle per cui ne è preposto il suo funzionamento. Conoscere il proprio ambiente però non è il fine ultimo di tali agenti (detti *basati su obiettivo*), essi se ne servono solo per svolgere i loro compiti, ed è assolutamente limitante sapere che possano lavorare solo nello spazio che conoscono. Lo *SLAM* si propone di fornire un metodo di apprendimento generalizzato, in modo da sollevare il progettista da ogni onere di tal genere, e rendendo il sistema indipendente da ogni legame con l'ambiente, in quanto sempre in grado di acquisire da solo la conoscenza necessaria. Numerosi sono però i problemi: in prima analisi, non esiste una vera e propria tecnica generalizzata, ma è necessario porre alcune assunzioni sul tipo di ambiente su cui si lavora, e sull'equipaggiamento robotico a disposizione. Inoltre, ogni robot è fornito di sensori con cui acqisisce dati dall'ambiente, e di attuatori con cui agisce ed opera in questo; purtroppo né i primi né i secondi sono privi di errori. È dunque necessario un filtraggio per eliminare tali errori, in

[1] Acronimo di *Simultaneous Localization and Map Building.*

particolare quelli odometrici, e permettere al robot di localizzarsi con precisione, al fine di avere una mappa precisa. Una tecnica particolarmente usata è il Filtro Esteso di Kalman, che richiede però una conoscenza dell'ambiente, pone forti vincoli su di esso, e lavora con un modello del sistema alquanto complesso a causa delle molteplici variabili da considerare.

Questa tesi vuole invece fornire un modello diverso, basato sugli *algoritmi evolutivi*, una tecnica che, emulando l'evoluzione biologica teorizzata da Darwin tramite la selezione naturale, permette di risolvere problemi di varia natura ricercando la soluzione migliore in uno spazio di ricerca molto ampio. Un tale strumento è slegato dal dominio di lavoro, e si adatta quindi a varie situazioni, rendendosi uno strumento ideale in un campo aperto come quello dell'AI e della robotica mobile, sebbene abbia lo svantaggio di richiedere un notevole impegno computazionale. L'algoritmo sviluppato è inoltre *online*, vantaggio che permette di avere real-time una mappa, seppur parziale, dell'ambiente. Occorre comunque specificare alcune assunzioni che vengono fatte: in primo luogo, tale algoritmo si applica esclusivamente ad ambienti di media scala e chiusi, ovvero tutti quegli ambienti come corridoi, piani di palazzi, case, etc. Inoltre, tali ambienti devono essere statici, dunque il loro stato non è funzione del tempo, e le loro condizioni restano sempre le medesime. Si richiede infine una larghezza almeno pari a 120cm, al fine di permettere al robot di potersi muovere per la scansione; questo implica che il sistema non è in grado di attraversare le normali porte, aventi larghezza di 1m, ma solo quelle più grandi.

Il robot utilizzato è il *Nomad 150*, prodotto dalla *Nomadic Software Inc.*, un robot costituito da tre sistemi sensoriali, sonars, tattile e odometrico, e da un sistema di tre motori che controllano la rotazione della torretta e gli spostamenti rotazionali e traslazionali di tre ruote. Queste sono simmetriche tra di loro, sono in grado di ruotare su se stesse, e costituiscono un sistema anolonomo con *zero gyro-radius*. Non è invece dotato di alcun computer di bordo, ma solo di un interfaccia seriale o wireless per collegarne uno.

Il resto della tesi è suddiviso nel seguente modo:

Capitolo 2 descrive il *Nomad 150*, il suo equipaggiamento, ed il simulatore *Nserver* con esso fornito;

Capitolo 3 excursus sulle tecniche utilizzate nello *SLAM*, tra cui il Filtro Esteso di Kalman e gli *algoritmi evolutivi*;

Capitolo 4 cenni teorici sugli algoritmi evoluti, con gli *algoritmi genetici* in

dettaglio, il loro fondamento teorico basato sulla teoria degli schemi di Holland, e le fasi principali nella loro implementazione;

Capitolo 5 analisi dettagliata dello *SLAM*, del problema della localizzazione e dell'acquisizione della mappa e descrizione ad alto livello dell'algoritmo utilizzato nel presente lavoro;

Capitolo 6 risultati del lavoro simulativo e di testing dell'algoritmo con il *Nomad 150*, nonché spiegazione dei dettagli implmentativi dell'algoritmo;

Capitolo 7 riassunto del lavoro svolto, conclusioni sugli obiettici perseguiti, sui limiti del sistema e sui possibili miglioramenti futuri.

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 ($1inch \approx 2.5cm$).

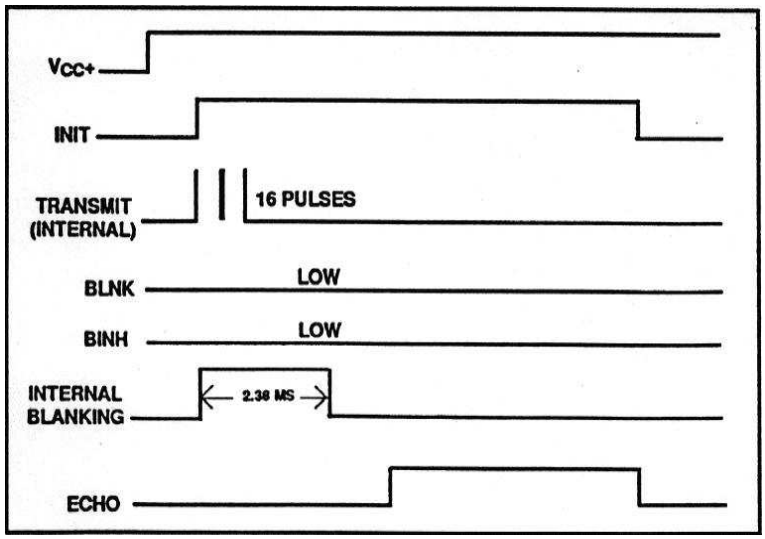

Figura 2.2: Diagramma temporale dei segnali logici di un trasduttore *Polaroid 6500*

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.

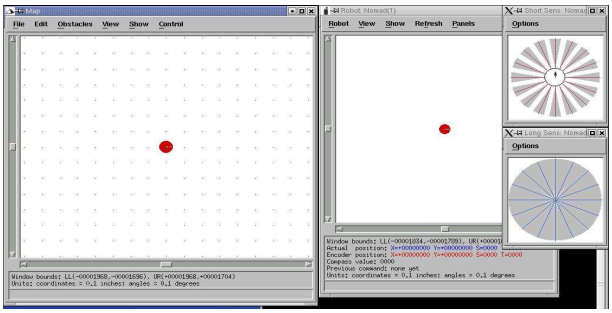

Figura 2.5: L'ambiente simulativo *Nserver*

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 [22]. 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 [19], 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.

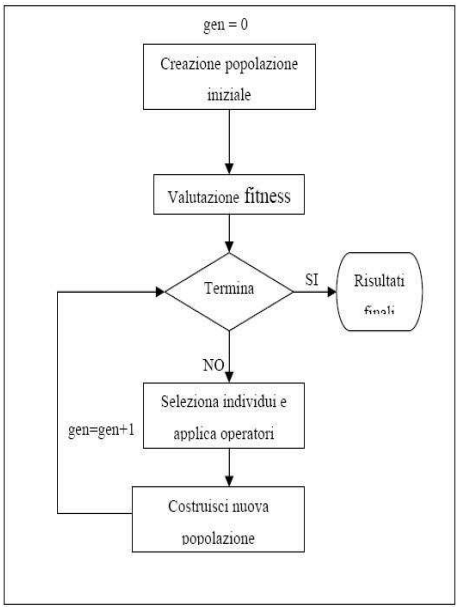

Figura 4.1: Schema di un *algoritmo genetico*

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 [15] 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.

Capitolo 5

SLAM Problem: analisi e soluzione genetica

5.1 Il problema della localizzazione e dell'acquisizione della mappa

Dal punto di vista dell'AI, un robot è un agente che opera in un ambiente, dal quale preleva informazioni tramite sensori, modificandolo per perseguire un proprio scopo tramite attuatori. Ogni automa viene infatti progettato per un obiettivo, sia esso quello di sollevare l'uomo da operazioni monotone, o di sostituirlo in circostanze pericolose (si pensi ad un reattore nucleare). Tutti questi ambiti richiedono spesso operazioni come la pianificazione del moto, la manipolazione degli oggetti o interazioni con l'uomo stesso; tutte queste cose sono possibili solo avendo una conoscenza appropriata dell'ambiente in cui si opera. Ovviamente, il grado di intelligenza e di autonomia di un robot è determinato da come tale conoscenza viene acquisita: in molti casi è il progettista stesso del sistema a fornirla, con il grosso limite che cambiando l'ambiente il sistema non è più in grado di lavorare bene. Si pensi, ad esempio, alle catene di produzione ed ai robot adoperati in esse; sono in grado di eseguire dei precisi movimenti in funzione del prodotto su cui si lavora, ma se cambiasse il prodotto andrebbero necessariamente riprogrammate. Per eseguire alcuni tasks, dunque, viene richiesta la conoscenza dell'ambiente, l'obiettivo del *SLAM* è proprio quello di non dare una conoscenza in fase di progettazione, adeguata solo per un ambiente, ma di rendere sistemi mobili autonomi ed in grado di risolvere tale problema da soli. Questo permette ad un robot di operare in qualunque circostanza, in quanto è il sistema stesso che apprende ciò di cui ha bisogno per poi poter perseguire i propri obiettivi. Si pensi al vantaggio di tale approccio in quei contesti dove neanche il

progettista del sistema possieda una mappa, come nell'esplorazione spaziale.

Conoscere un ambiente equivale ad avere una sua mappa, ma creare una mappa in modo autonomo è tutt'altro che banale. Un robot che costruisca autonomamente il proprio ambiente, dovrebbe essere in grado di capire dove si trova; viceversa la localizzazione può avvenire solo tramite una mappa. Inoltre, bisogna definire un modo per muoversi per ottenere le informazioni necessarie a tale processo, ovvero una modalità di *esplorazione*; una soluzione per tale aspetto è stata proposta da Marino [14]. Il modo più semplice di agire, una volta stabilita una strategia di esplorazione, sarebbe quella di limitarsi a prendere informazioni con i sensori disponibili, e con essi elaborare la mappa; il problema è che tutto ciò non tiene conto degli errori di misura dei sensori, di quelli odometrici delle ruote e, nel caso del *Nomad 150*, della torretta. Il risultato di questa acquisizione senza localizzazione sarebbe una mappa inadeguata per il task successivo del robot. Viceversa, una corretta localizzazione richiederebbe una mappa, che però in tal caso non è disponibile, dato che il sistema lavora in ambienti a lui ignoti (altrimenti avrebbe una conoscenza fornita da un ente onnisciente, e dunque non sarebbe autonomo).

Per meglio comprendere il *SLAM* è conveniente separare i due aspetti: localizzazione e acquisizione della mappa. Un robot ha due tipi sostanziali di percezioni:

- *percezioni esterne*, che possono consistere in sistemi sonar, laser, infrarossi, che danno informazioni sull'ambiente di lavoro;
- *percezioni interne*, inerentemente ai sensori incrementali come gli encoders per l'odometria.

Pensare che le informazioni odometriche bastino a risolvere il problema della localizzazione, come detto prima, è errato, a causa degli errori di *drifting* (e non solo) delle ruote. Avere dei sensori che diano informazioni sull'esterno è importante sia al fine di correggere tali errori, sia al fine di recuperare la posizione quando il robot si smarrisce. Infatti, si può vedere la localizzazione a due livelli di astrazione diversi:

- *localizzazione globale*, che riguarda la conoscenza del robot nell'intero ambiente, non sulla base di precise informazioni metriche, ma sull'identificazione di un posto data una lista di possibili posizioni. È, ad esempio, il tipo di localizzazione che occorre quando il robot si smarrisce o sbaglia a muoversi e deve tornare indietro. In questi casi, non è importante che il sistema corregga l'errore prodotte dalle sue ruote, quanto che recuperi una posizione consistente con quanto ha appreso;

- *correzione della posizione*, è la correzione locale degli errori prodotti durante lo spostamento dell'automa, molto importante per la successiva fase di acquisizione della mappa.

Se il primo tipo viene di solito effettuato tramite l'uso di una cronologia degli spostamenti eseguiti, memorizzando tutte le informazioni acquisite, per il secondo si usa un filtraggio come quello esteso di Kalman, oppure un metodo di ottimizzazione come gli *algoritmi genetici*. Oltre a riuscire a comprendere dove si trova, un sistema autonomo deve anche essere in grado di comprendere cosa sta rilevando, comprendere se ha già visto un determinato ostacolo oppure no, riconoscere dei *landmarks* dell'ambiente. Ciò, non solo è molto importante nell'ambito dell'esplorazione (di cui non trattiamo), ma è un'ulteriore tecnica per correggere la posizione del robot.

Nell'acquisizione della mappa invece, a fronte della tecnica di esplorazione utilizzata, il problema è di avere delle informazioni che siano il più precise possibili, sulla base dei sistemi sensoriali a disposizione. I sonars ad esempio, richiedono una certa vicinanza all'ostacolo rispetto al laser, a causa di una maggiore imprecisione, lavorano all'interno di un cono e di solito sono disposti in corone circolari. Un laser è molto più preciso ma di solito i robot ne hanno solo uno, e quindi richiede un maggior tempo per il puntamento e la scansione. Esistono due modi di descrivere una mappa, ed ognuna di queste tecniche è poi strettamente legata al tipo di esplorazione scelta:

- *topologiche*, ovvero grafi, i cui nodi rappresentano posti liberi dove il robot è riuscito a passare, mentre gli archi sono le possibili traiettorie per passare da un posto all'altro. Hanno un grado di precisione minore rispetto alle mappe metriche, in quanto non possiedono informazioni dirette sugli ostacoli, o sulla loro esatta morfologia, però sono molto più veloci a livello computazionale, e hanno a disposizione tutti gli strumenti offerti dagli algoritmi sui grafi;

- *metriche*, come le griglie di occupazione, dove abbiamo informazioni di tipo metrico che identificano ogni oggetto rilevato, in coordinate cartesiane o polari. Sono molto precise ma anche molto onerose a livello computazionale, sia in termini di tempo che di spazio.

Esistono in realtà anche modelli ibridi, che associano ad una mappa topologica anche informazioni metriche, come la distanza in metri tra due posti e l'angolo della traiettoria che li unisce. Tramite le informazioni sensoriali esterne si fornisce una misura della correzione da apportare agli errori odometrici delle ruote, ottenendo così una localizzazione precisa del robot. Le

stesse rilevazioni usate per correggere gli errori vengono utilizzate al fine di costruire una porzione della mappa, che fornisce ulteriori informazioni per la localizzazione, come posti in cui già si è stati.

5.2 Un algoritmo genetico on-line per il problema dello SLAM

La soluzione proposta nel presente lavoro è un algoritmo genetico per correggere gli errori odometrici del robot; la novità rispetto alla letteratura precedente è che è *online*, e fornisce in ogni istante una mappa (seppur parziale) al sistema mobile, per permettere operazioni di *path planning*. Rispetto al Filtro Esteso di Kalman (EKF), non sono necessarie assunzioni sul dominio di lavoro, né alcun modello del sistema in questione, l'unico punto critico è il *tuning* dei parametri dell'algoritmo, che richiede un numero considerevole di prove. L'idea alla base è che gli errori odometrici sono direttamente proporzionali alla distanza percorsa dal robot, più facilmente da correggere quindi localmente che globalmente; ciò giustifica una versione *on-line* dell'algoritmo. Il robot alterna movimenti delle ruote a scansioni dell'ambiente; l'unione di un movimento ed una scansione costituisce uno *step*, ed ogni N step viene lanciato l'algoritmo genetico per rilocalizzare il sistema e costruire la porzione dell'ambiente scandito. L'algoritmo è il seguente:

La prima fase prevede la costruzione di una popolazione generazionale di 50 cromosomi, che viene fatta evolvere per 250 generazioni[1]. Ogni cromosoma rappresenta un insieme di correzioni (dx, dy) dei dati odometrici del robot, entro un range [-100,100] decimi di pollici; per non lavorare con numeri con segno però si preferisce traslare il range a [0,200]. La codifica scelta è di tipo binario, con una stringa di 16 bit, di cui i primi 8 per dx, i restanti per dy. Questa è l'unica generazione in cui la popolazione viene costruita casualmente, tutte le successive sono invece le discendenze frutto dell'incrocio e della mutazione. Usando 8 bit, si arriva a codificare fino a 255, esistono quindi 55 codifiche che non vengono utilizzate (perchè il sistema corregge fino a 200, come detto prima), né l'incrocio riesce a creare tali configurazioni illegali. Non è possibile assumere a priori che esistano delle configurazioni migliori di altre, ciò nonostante, assumendo che l'errore delle ruote sia di tipo gaussiano, le correzioni minori hanno comunque una probabilità maggiore di quelle più grandi.

L'unico modo per risolvere il doppio problema della localizzazione e della

[1]Si rimanda al cap. 6 per considerazioni sulla scelta dei parametri nei test simulativi e nelle prove con il *Nomad 150*.

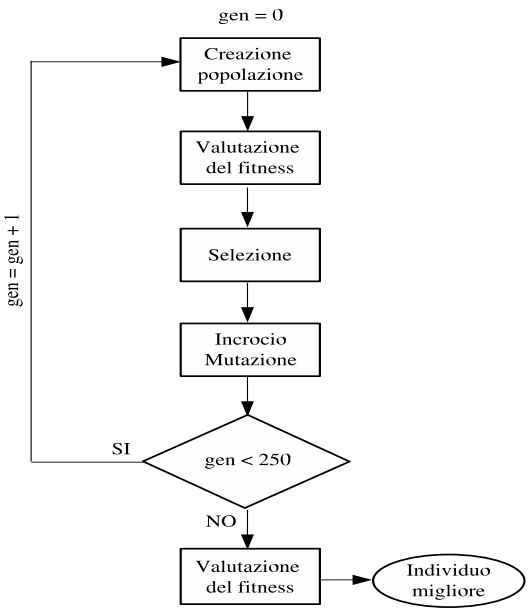

Figura 5.1: L'algoritmo genetico

creazione di una mappa precisa è fornire un metodo di acquisizione dell'ambiente, che da un lato permetta di correggere gli errori odometrici, e dall'altro garantisca la costruzione della mappa. A tal fine si utilizzano i dati proveniente dai sistemi sensoriali esterni, nel caso del presente lavoro i sonars del *Nomad 150*. I sonars non sono strumenti precisi come un laser: alle volte il segnale emesso da uno di loro viene intercettato da un altro sonar, oppure rimbalza sulle superfici in modo tale da non tornare, non rilevando così l'oggetto. Eppure, al tempo stesso, essi sono la base con la quale vengono corretti gli errori odometrici, quindi bisogna supplire alla mancanza di precisione del sistema con dei controlli a basso livello, come la rotazione della torretta al fine di aumentare la risoluzione, aumentando la ridondanza di informazioni e quindi anche avvalorandole (si veda il capitolo 6 per i dettagli implementativi di tale controllo). Al termine dell'algoritmo genetico il robot è posizionato correttamente, e dunque è conveniente utilizzare le informazioni sensoriali rilevate nel primo step, poichè hanno una credibilità decisamente maggiore. Viceversa, al quindicesimo step, l'ultimo, l'errore odometrico accumulato in tutti i precedenti movimenti necessita di essere corretto, ed i dati acquisiti al primo step possono tornare utili. Dopo aver generato una popolazione di correzioni possibili ed averle codificate, si passa alla valutazione del fitness sfruttando le informazioni acquisite esclusivamente al primo step (che serviranno per discriminare gli individui) e all'ultimo step. Partendo dalle co-

ordinate del robot nello step iniziale, si sommano le distanze rilevate da tutti i sonar, ottenendo le coordinate di una cella in una griglia di occupazione, e si pone un campo *occ* ad 1 ad indicare che questa è *potenzialmente* occupata. Il *Nomad 150* utilizza un sistema di riferimento cartesiano $(XY)_O$, mentre per la mappa si è operata una conversione ad un sistema di riferimento *riga-colonna*, la cui origine [0,0][2] rappresenta la posizione iniziale del robot, ed il cui verso delle righe è opposto al verso della y cartesiana canonica. Tale sistema, di dominio e codominio infinito, ha il vantaggio di essere discreto e non continuo, individuando così meglio le celle, e permettendo metodi di ricerca con tempo computazionle $O(n+m)$[3], con n il numero massimo di righe e m di colonne, inoltre i suoi assi hanno gli stessi versi delle codifiche grafiche più usate (come *bitmap*, o *JPEG*). Infine, per la codifica grafica, si utilizza una codifica *portable bitmap* o *PBM*, monocromatica, con sistema di riferimento *riga-colonna*, ad assi discreti e finiti, dove la cella [0,0] individua il pixel dello schermo in alto a sinistra. Sommando ai dati odometrici le correzioni di ogni individuo e le distanze riscontrate dai sonars all'ultimo step, si calcola quante delle celle rilevate occupate al primo step vengano riprese; maggiore è tale numero, maggiore sarà il fitness per quel cromosoma. Creata la griglia di occupazione, si riesegue la stessa procedura, questa volta sommando alle coordinate del robot nell'ultimo stato ogni possibile coppia di correzioni, più le distanze rilevate dai sonar in questo step. Ancora una volta quello che si ottiene sono le coordinate di una cella, che vengono utilizzate per entrare nella griglia precedentemente costruita ed acquisire il valore di *occ* ad essa assegnato. Il significato di tale euristica è nel confrontare tutte le possibili mappe candidate, frutto cioè delle correzioni, con la conoscenza più precisa a disposizione del sistema robotico, ovvero quella al primo step, subito dopo la correzione dell'algoritmo genetico e tramite un controllo sulla precisione dei sonars. Tanto maggiore sarà il numero di celle che un cromosoma condivide con la griglia di occupazione, tanto più forte esso sarà. Il fitness di una soluzione è dunque esprimibile come:

$$f = \sum_i occ_i$$

dove i rappresenta un sonar ed occ_i il valore della corrispondente cella individuata. Ci sono varie osservazioni da fare sul tipo di scansione usata, in quanto questa influenza molto la valutazione del fitness. Un vantaggio dell'avere molte informazioni, è che esse permettono di discriminare meglio la bontà di ogni soluzione candidata, senza dare convergenze errate all'al-

[2]Da questo punto in poi con (x,y) si intenderà una coppia in un riferimento cartesiano di tipo $(XY)_O$, con [x,y] una coppia in un riferimento *riga-colonna* come descritto sopra.

[3]Si rimanda al capitolo la descrizione della struttura della griglia di occupazione.

goritmo. Si pensi, a tal proposito, che l'algoritmo esegue una scansione al primo step, ed una all'ultimo; sulla base di quest'ultima vengono valutati i cromosomi. È ovvio, che in tal modo vengono favorite tutte le correzioni che tendono a spostare il robot verso il primo step, in quanto prenderanno più celle. Tutto questo però è vero nel caso in cui le scansioni agiscano solo localmente, ovvero nella zona vicina al primo step (cosa che potrebbe esser giustificata per avere una precisione ancora migliore). Per tale motivo è importante avere scansioni a 360 gradi, proprio per garantire tale equità nei confronti dei candidati, e poter al tempo stesso non perdere la qualità delle scansioni. Infine, individuare celle ad angolo giro, permette di avere informazioni anche mentre il robot si trova in curva, e quindi il primo step può trovarsi su un lato dell'ostacolo, e l'ultimo step su un altro.

La selezione utilizzata è quella proporzionale all'idoneità, con il metodo del Campionamento Stocastico Universale, in modo da non correre rischi di eliminare gli individui più forti. Tuttavia, per il calcolo del valore atteso, si è scelta la serie di Boltzmann per garantire esplorazione alle prime generazioni, e convergenza alle ultime. Si riporta in figura 5.2 lo pseudo-codice del suddetto algoritmo.

```
i := 0
sum := 0
generare un numero casuale m con distrib. uniforme [0,1]
while(i < numeroCromosomi) {
        sum = sum + valoreAtteso(i)
        while(sum > m) {
                selezionaIndividuo(i)
                m = m + 1
                sum = sum + valoreAtteso(i)
          }
  }
```

Figura 5.2: Pseudo-codice dell'algoritmo genetico proposto.

Una volta spostati gli individui prescelti nella *mating pool*, essi vengono ordinati sulla base del numero di volte che sono stati scelti, dopo di che l'accoppiamento procede nel seguente modo: il cromosoma scelto più volte si accoppia le k volte che gli spettano con i k individui che vengono dopo di lui in tale ordinamento, questa strategia viene riapplicata allo stesso modo per tutti gli altri cromosomi fino a che non si sono stabiliti tutti gli accoppiamenti. In questo modo, da un lato chi è più forte si accoppia con individui altrattanto forti, e dall'altro però avendo a disposizione tante possibilità, ne

dedica anche una porzione all'esplorazione di nuovi genotipi con individui deboli. Il seguente esempio chiarisce l'aspetto:

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

dove, la coppia *{x,y}* rappresenta il cromosoma x, selezionato y volte. Considerando solo i cromosomi scelti e ordinandoli diventa:

$$\{(2,3)(4,2)(6,2)(1,1)(3,1)(8,1)\}$$

Il cromosoma 2 si accoppia allora 3 volte e precisamente con gli individui 4,6 e 1, e la sequenza diventa:

$$\{(2,0)(4,1)(6,1)(1,0)(3,1)(8,1)\}$$

Come si vede, il cromosoma 2 ha finito le sue possibilità di riprodursi, e gli altri cromosomi coinvolti nella riproduzione hanno una possibilitìn meno; nel caso del cromosoma 1, esso in realtà ha terminato il proprio compito. Il cromosoma 4 si accoppia con il 6:

$$\{(2,0)(4,0)(6,0)(1,0)(3,1)(8,1)\}$$

Infine, l'ultima coppia è costituita dai cromosomi 3 e 8:

$$\{(2,0)(4,0)(6,0)(1,0)(3,0)(8,0)\}$$

L'incrocio utilizzato è a due punti, scambiando i bit tra essi compresi dei due genitori, come mostrato nel seguente esempio:

Cromosoma 1: 10011100
Cromosoma 2: 01101001

ammettendo di avere i punti di incrocio dopo il secondo ed il quinto bit, i discendenti saranno:

Figlio 1: 10101100
Figlio 2: 01011001

Utilizzando l'incrocio a due punti, si favorisce la creazione di più schemi nuovi di quanti ne permette quello ad un punto; inoltre, separando molto di meno gli estremi, permette agli schemi con lunghezza di definizione elevata di sopravvivere più a lungo. La tradizione dello SLAM associa le probabilità di riproduzione R, incrocio I e mutazione M secondo la relazione

$$p(I) + p(M) + p(R) = 1$$

nel seguente algoritmo invece l'incrocio viene sempre effettuato, mentre la popolazione, essendo *generazionale*, non prevede alcuna forma elitaria. L'operatore di mutazione viene applicato con una probabilità del 10%, e inverte un allele del cromosoma scelto a caso, come mostrato nel seguente esempio:

Cromosoma: 11010100

mutando il terzo bit, diventa:

Cromosoma: 11110100

L'incrocio ha la funzione di promuovere nuovi schemi, ma anche di creare sempre più istanze degli schemi migliori, come visto nel cap.4, mentre la mutazione ha il ruolo di esplorare sempre nuovi genotipi.

Terminate le 250 iterazioni dell'algoritmo, vengono calcolati per l'ultima volta i fitness, e l'individuo più forte viene scelto come la correzione da applicare ai dati odometrici, resettando gli encoder del robot; non bisogna infatti pensare che correggere equivalga a spostare il robot nel giusto punto che corrisponde ai suoi dati, altrimenti per tale spostamento sorgerebbero ulteriori errori. Quello che invece viene fatto, è settare ad un valore diverso i registri interni del sistema.

Una volta terminata la localizzazione si passa alla costruzione della mappa, sempre tramite le correzioni appena trovate e le rilevazioni dei sonars nell'ultimo step. Vengono ricalcolate tutte le celle trovate precedentemente, utilizzando anche le scansioni effettuate negli step intermedi, ma solamente quelle alla destra del robot vengono poi segnate come ostacoli veri e propri, questo fondamentalmente per un motivo: come detto prima, il sistema è stato testato con un'esplorazione proposta da Marino [14], che prevede di tenere il muro alla destra del *Nomad 150* rispetto al suo orientamento, quindi tutte le celle a destra sono più vicine e più precise in termini di rilevazioni. Mentre per correggere la posizione del sistema era importante avere una grossa

mole di dati, anche ridondanti, per la costruzione della mappa è più importante la precisione delle scansioni. Questa a sua volta fornisce informazioni necessarie al robot per l'esplorazione, come riconoscere posti già visti, calcolare le distanze per eventuali pianificazioni di moto, comprendere quali parti dell'ambiente mancano da esplorare. In questo modo il sistema acquisisce in modo autonomo la conoscenza dell'ambiente, che può essere sfruttata in seguito per altre operazioni.

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.

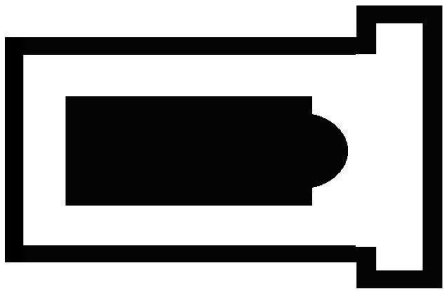

Figura 6.1: Dipartimento di Automatica

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.

Senza alcun filtro, il metodo più semplice per acquisire la mappa è quello di considerare come muri tutte le celle rilevate dal sonar più vicino all'ostacolo, come mostrato in fig.6.2.

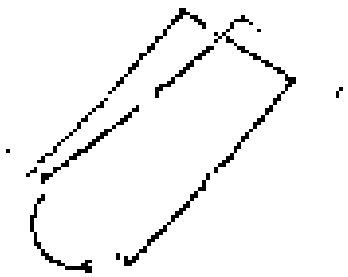

Figura 6.2: Mappa realizzata senza algoritmo genetico, usando solo le letture odometriche.

Risulta evidente che il *Nomad 150* non è riuscito a chiudere l'esplorazione interna e a passare all'esterno (infatti sulla mappa quest'ultima manca del tutto), questo a causa della localizzazione errata. Senza un'adeguata correzione infatti gli errori odometrici si sono accumulati, impedendo al robot di riconoscere che era già passato in alcuni punti, e che aveva finito la rilevazione della parte interna. Inoltre, a livello grafico, la mappa è conseguentemente venuta male, ed è inutilizzabile per qualsiasi pianificazione del moto.

Al fine di capire l'attuale implementazione, è utile mostrare le precedenti versioni dell'algoritmo, spiegare i loro limiti, e scendere più in dettaglio nel codice solo con la versione finale. In una prima versione, la soluzione proposta era diversa: al fine di ridurre notevolmente il tempo computazionale, si era scelto di lanciare l'algoritmo genetico solamente quando il robot si trovava in curva. Questo perchè si assumeva che lì dove il robot andava dritto, il muro fosse effettivamente dritto, e quindi non solo non ci fosse bisogno di alcuna correzione, ma neanche di acquisire informazioni, oltre quelle necessarie alla pianificazione. In pratica, considerando il *Nomad 150* come un agente, tramite i dati odometrici acquisiti dall'ambiente, e i movimenti che esso effettuava in funzione di questi, si risaliva al concetto di *linea dritta*. All'interno di una struttura dati chiamata *Cell*, c'erano due campi, *occ* ed *emp*; il primo rappresentava il numero di volte che una scansione aveva rilevato quella cella

come piena, il secondo il numero di scansioni che quella cella era risultata vuota. Questa tecnica è la stessa utilizzata da Duckett [8], ma nel caso di *emp* si rivela alquanto ambigua; se un sonar mostra che non esiste alcun ostacolo nel suo cono infatti si potrebbe incrementare tale campo per tutte le celle sulla sua direzione. Ma, anche nel caso che una cella risulti occupata, tutte le celle precedenti ad esse lungo la direttrice del sonar potrebbero esser viste come vuote (altrimenti il sonar avrebbe segnalato un ostacolo a minor distanza). In ogni caso, quello che si è riscontrato è che tale tecnica portava un valore di *emp* molto più elevato di quello di *occ*; fattore molto importante per la funzione fitness usata. Questa infatti, si basava sull'aggiungere alle coordinate del robot le possibili correzioni (create sempre come mostrato nel cap. 5) e le distanze rilevate dai sonars, entrare nella cella risultante e prendere il valore più piccolo dei due campi, iterando tale processo per tutti i sensori. Il significato dietro questa euristica è il seguente: se una cella è palesemente occupata, avremo un valore alto di *occ* ed uno basso di *emp* (se la cella è vuota basta invertire), questo implica che prendere il minimo tra i due valori equivale a prendere un valore molto basso. Se invece la cella è ambigua, il valore dei due campi è pressoché identico, e prendere il minimo equivale comunque a prendere un valore relativamente più alto che nel caso precedente. È ovvio che, in tal modo, minore è la sommatoria finale di tali valori, più il cromosoma sarà idoneo. Questa tecnica è stata comunque scartata a causa della difficoltà di unire graficamente le parti dritte degli ostacoli, che come detto prima, non vengono trovate dall'algoritmo genetico, con le curve (si veda in merito fig.6.3). Infatti, non esiste alcuna assunzione che permetta di trovare un algoritmo per capire quali due pixel delle curve vadano presi come estremi del segmento congiungente da realizzare.

Figura 6.3: Mappa realizzata con algoritmo genetico solo nelle curve.

A causa di questo problema, si è applicato l'algoritmo genetico non più solo alle curve, ma anche ai tratti rettilinei, con una strategia di scansione

però diversa da quella proposta nella versione finale. Si eseguivano scansioni in tutti gli steps, non con tutti i sonars però, solamente con quello più vicino all'ostacolo[1] e con i suoi due adiacenti. Questa tecnica ha evidenziato, nelle prove con il *Nomad 150*, il suo limite principale: l'euristica scelta non era adatta alla natura on-line dell'algoritmo. Poichè i sensori scandivano solo le celle vicine al *sonar minimo*, le informazioni disponibili erano una linea di celle alla destra del robot, che non permettevano di discriminare bene tra le varie soluzioni candidate. Anzi, le correzioni che riportavano il robot indietro erano ancor più favorite, perchè prendevano un numero di celle ancora maggiore. In sostanza era sbagliata l'euristica utilizzata per valutare il fitness; non era discriminante e non era equa.

L'ultima versione è il frutto delle due precedenti, ed è già stata descritta ad alto livello nel capitolo 5, si cercherà ora di scendere ad un livello di dettaglio maggiore, visualizzando i risultati ottenuti. L'euristica non prevede più di utilizzare solo i dati relativi ad una porzione dell'ambiente, quella vicina al sonar minimo, bensì una scansione a 360°, aumentando la risoluzione tramite dieci rotazioni di 2 gradi della torretta. Essendo i sonars 16, in tutto si hanno $16 * 11 = 176$ dati, che vengono memorizzati nella struttura *State*, che tiene traccia dello stato del robot e delle informazioni sull'ambiente.

```
while(i<11){
  pr(0,0,-20) ; // rotazione della torretta in senso orario
  ws(0,0,1,20) ;
  gs() ; // aggiornamento dei sensori

  for(j=0; j<SONARS; j++)  {
    tmPtr->sonar[0][j+16*i] = State[STATE_SONAR_0 + (min_son+j)%16] ; // sonars
    tmPtr->sonar[1][j+16*i] = State[STATE_CONF_TURRET] ;
  }
  i++ ;
}
```

Queste scansioni riguardano però solo il primo e l'ultimo step, il numero di step eseguiti prima di lanciare l'algoritmo genetico è 15. Per quel che riguarda gli step intermedi, le scansioni continuano ad avvenire come nella versione precedente, ma i dati servono esclusivamente per ricostruire la mappa al termine dell'algoritmo genetico, esse non contribuiscono nella valutazione del fitness. In ogni caso, è stata sviluppata anche una versione che fa uso di questo approccio, il cui codice è mostrato qui sotto.

```
while(i<16 && flag && (!count || (count+1)%N!=0))
  {
    tmPtr->sonar[0][i]=State[STATE_SONAR_0 + i] ;
    tmPtr->sonar[1][i]=State[STATE_CONF_TURRET] ;
    x = floor(State[STATE_CONF_X] + (10*tmPtr->sonar[0][i]+R)*cos((2*
        M_PI*i/16.0)+(tmPtr->sonar[1][i]*2*M_PI/3600.0)) + 0.5);
```

[1]Da questo punto indicheremo tale sonar come *sonar minimo*.

```
  y = floor(State[STATE_CONF_Y] + (10*tmPtr->sonar[0][i]+R)*sin((2*
      M_PI*i/16.0)+(tmPtr->sonar[1][i]*2*M_PI/3600.0)) + 0.5);
  r=floor(-y/50.0+0.5) ;
  c=floor(x/50.0+0.5) ;
  i++ ;
}

while(ris_count != 2) {  //scansione torretta per risoluzione
  ++ris_count ;
  gs() ;
  minimum();

  if(State[STATE_SONAR_0+(min_son)] < old_min){
    old_min_son = min_son ;
    old_min = min ;
    old_turr = State[STATE_CONF_TURRET] ;
  }
  j = 0 ;
  while(j<16 && flag && (!count || (count+1)%N!=0))
  {
    tmPtr->sonar[0][i]=State[STATE_SONAR_0 + j] ;
    tmPtr->sonar[1][i]=State[STATE_CONF_TURRET] ;
    x = floor(State[STATE_CONF_X] + (10*tmPtr->sonar[0][i]+R)*cos((2*
        M_PI*j/16.0)+(tmPtr->sonar[1][i]*2*M_PI/3600.0)) + 0.5);
  y = floor(State[STATE_CONF_Y] + (10*tmPtr->sonar[0][i]+R)*sin((2*
        M_PI*j/16.0)+(tmPtr->sonar[1][i]*2*M_PI/3600.0)) + 0.5);
    r=floor(-y/50.0+0.5) ;
    c=floor(x/50.0+0.5) ;
    i++ ;
    j++ ;
  }
}
```

In questo caso, le rotazioni della torretta negli step intermedi sono di 7,5 gradi in entrambi i sensi, 2 in un senso e 2 nell'altro, rispetto al sonar minimo.

L'euristica non fa più uso del campo *emp*, ma solamente di quello *occ*; inoltre, il dominio di tale campo è ristretto ai soli valori {0,1}, ovvero, o la cella non viene mai rilevata come occupata, o accade almeno una volta. Per comprendere il motivo dietro questa scelta, bisogna pensare che la scansione ora avviene a 360 gradi, e può essere divisa in due categorie diverse:

- celle vicine al robot, in tal caso è difficile che i sonars sbaglino, e dunque non è necessario tener conto di quante volte risultino occupate per dare loro credibilità;
- celle distanti dal robot, in tal caso è molto più probabile di avere un errore sulla distanza reale, ma questo sarebbe accaduto anche con l'euristica precedente.

Quello che accadeva prima, è che lavorando solo sulle celle vicine, c'era troppo ridondanza e nessuna correzione era realmente migliore dell'altra. In questo

modo invece, tutte le celle, vicine e lontane, hanno la stessa importanza, e questo permette di individuare meglio i cromosomi più forti.

Testando l'algoritmo su *Nserver*, si è provveduto ad inserire due tipi di errori, uno odometrico ed uno sulla rotazione della torretta, entrambi con distribuzione gaussiana. Il risultato è visualizzato sulla seguente mappa[2]:

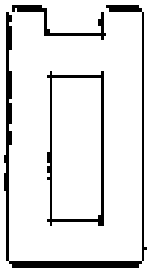

Figura 6.4: Mappa realizzata con ga e *Nserver*

Nella realtà invece, la mappa ottenuta è riportata sotto:

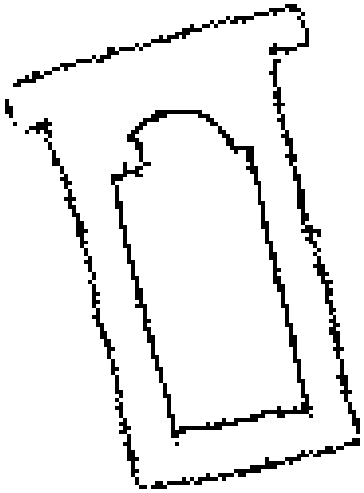

Figura 6.5: Mappa realizzata con ga e *Nomad 150*

[2]Il simulatore non permette di costruire ostacoli circolari, dunque non è stato possibile realizzare l'emiciclo presente nella realtà.

Riguardo il tempo necessario per acquisire la mappa, il *Nomad 150* impiega, con la tecnica di esplorazione proposta da Marino [14], circa 2 ore e 30 minuti, sebbene una mappa parziale sia disponibile fin da subito. A livello di memoria utilizzata, la struttura cardine per memorizzare le celle occupate ed i muri risultanti dall'algoritmo genetico è la struttura *Cell*, avente la dimensione di 44 bytes (si veda il codice sotto per la definizione della struttura). Questa struttura viene organizzata in una matrice dinamica, a colonne e righe variabili; per sprecare la minor quantità di memoria possibile vengono allocate solamente le celle necessarie.

```
struct Cell {
  int occ; // indica se la cella è occupata durante la scansione
  int row ;
  int col ;
  int visited ; // traiettoria del robot durante la scansione
  int wall ; // cella realmente occupata dopo ga()
  int erasable ;
  int new_obs ; // probabile zona ancora da esplorare
  struct Cell* row_prev ;
  struct Cell* row_succ ;
  struct Cell* col_prev ;
  struct Cell* col_succ ;
};
```

Ogni cella rappresenta uno spazio quadrato con lato di 12.5cm, un ambiente come il piano del dipartimento, con dimensioni esterne di 8m x 16m, richiederebbe con una matrice statica circa 8192 celle, mentre con la struttura dinamica utilizzata tale numero scende a meno di 5000 celle; in termine di memoria RAM occupata questo significa risparmiare quasi il 40% di spazio[3]. Per ciò che riguarda l'algoritmo, ogni cromosoma è rappresentato dalla struttura *Cromosoma*, mostrata sotto:

```
struct Cromosoma {
  double fitness ; // fitness totale
  int dx ;
  int dy ;
  int selected ;  //numero volte che il cromosoma viene scelto per il crossover
  unsigned cod : 16 ;  //codifica a 12 bit delle correzioni
  unsigned cod_dx : 8 ;
  unsigned cod_dy : 8 ;
} ;
```

La popolazione prevede 50 individui, per un totale di 2600 bytes; a questi si aggiunge un'ulteriore popolazione dove vengono memorizzati i figli, che alla fine di ogni iterazione vanno poi a sostituire i genitori. Il formato con cui viene creata la mappa è il *Plain Portable Bit Map*, una codifica monocromatica molto semplice da realizzare, che richiede un solo bit per identificare un pixel, 1 se nero, 0 se bianco.

[3]Tale calcolo dipende comunque dalla topologia degli ostacoli interni dell'ambiente.

Capitolo 7

Miglioramenti futuri e conclusioni

Il lavoro proposto tratta di una possibile soluzione allo *SLAM* tramite l'uso di un algoritmo genetico per ambienti di media scala statici, implementata su un sistema robotico mobile, il *Nomad 150*, grazie anche all'ausilio di una tecnica di esplorazione e di pianificazione proposta da Marino [14].

Rispetto alla letteratura passata in tale ambito, che fa uso del Filtro Esteso di Kalman, tale metodo non richiede alcuna conoscenza del dominio di lavoro, adattandosi ad ogni ambiente poichè non fa alcuna assunzione su di esso, ed inoltre non necessita di un modello del sistema, che, dato il gran numero di variabili in gioco, risulta molto complesso. Un vantaggio rispetto ai lavori precedenti che fanno uso degli *algoritmi genetici* è che questo metodo è *online*, permettendo di avere sempre una mappa, seppur parziale, a disposizione del robot. Si è cercato di integrare controlli automatici a basso livello con strumenti tipici del campo dell'intelligenza artificiale, per creare un sistema autonomo, in grado di agire razionalmente e di integrare alla propria conoscenza quella del suo ambiente di lavoro, senza utilizzare alcun metodo di *training*. Il dispendio di risorse computazionali, sia in termini di tempo, sia in termini di memoria RAM utilizzata, è stato notevolmente ridotto rispetto ai lavori precedenti, richiedendo solo due ore e 30 minuti per avere una mappa completa di un ambiente di oltre 100mq. Al fine di rendere l'idea, si pensi che nell'algoritmo *offline* proposto da Duckett [8], il solo tempo di computazione richiede due ore di tempo, a cui bisogna aggiungere il tempo di esplorazione dell'ambiente.

Tra i limiti, bisogna notare che si potrebbero ottenere mappe più precise con un opportuno controllo sulla torretta, che permetterebbe di utilizzare

ancora più rotazioni, al fine di avere una risoluzione maggiore ed euristiche più robuste. Un limite tecnologico invece è l'uso dei sonars, troppo imprecisi, che andrebbero sostituiti con dei laser o integrati con altri sistemi come quello ad infrarossi.

Alcuni spunti per miglioramenti futuri si possono individuare nella costruzione di un'applicazione che mostri real-time la costruzione della mappa e nell'adattamento del codice per una parallelizzazione dell'algoritmo. Inoltre, sarebbe interessante dotare il sistema di un ulteriore livello di astrazione, basato su metodi di apprendimento tramite il riconoscimento di patterns, insegnando al sistema i concetti di *porta* o di oggetti tipici come termosifoni, grate, etc.

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 Transcriptions of Robotics Automation, 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 Transcriptions Robotics Automation, 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] F.A.Marino, *Tecniche di esplporazione e pianificazione del moto per il problema dello SLAM*, Tesi di Laurea in Ingegneria dell'Automazione, Università Tor Vergata di Roma, novembre 2004.

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

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

[17] C. Pisani, *Gli algoritmi genetici ed alcune applicazioni*, Università degli Studi di Torino, Facoltà di Scienze Matematiche, Fisiche e Naturali, Corso di Laurea in Matematica, 2001.

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

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

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

[21] F.Romanelli, *Laboratorio di Automatica: Robot Nomad 150*, Tirocinio di Automatica, Università di Tor Vergata, 2002.

[22] 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.