Università degli studi di Roma
La Sapienza

FACOLTÀ DI INGEGNERIA

Tesi di laurea in Ingegneria Informatica

a.a. 2002-2003
maggio 2004

Pianificazione del percorso per un robot tramite mappe topologiche probabilistiche e adattamento della traiettoria

Daniele Calisi

Relatore
prof. Daniele Nardi

ai miei genitori

Desidero innanzitutto ringraziare il professor Daniele Nardi per avermi dato l'opportunità di lavorare su questa tesi in un campo così stimolante e per la disponibilità mostrata nel seguirne gli sviluppi.

Ringrazio inoltre Alessandro Farinelli per avermi fatto da "tutore" lungo tutto lo sviluppo della tesi e per la sua disponibilità a darmi consigli, fornirmi materiale su cui lavorare e seguirmi nella fasi più critiche.

E' doveroso ringraziare infine tutti i colleghi che in questo periodo, grazie al loro aiuto, ai loro suggerimenti o alla loro disponibilità, hanno accelerato e reso più piacevole lo sviluppo di questo lavoro.

"Un Maestro nell'arte della vita, non fa nessuna
precisa distinzione tra lavoro e gioco, fatica e riposo,
mente e corpo, educazione e ricreazione. Insegue
semplicemente la sua visione di Eccellenza attraverso
qualunque cosa sta facendo e lascia agli altri decidere
se sta lavorando o giocando.
A lui sembra sempre di fare entrambe le cose."

Indice

Introduzione

Il problema della pianificazione del moto è di fondamentale importanza in tutte le applicazioni che hanno a che fare con dei robot mobili. Attualmente esistono vari metodi per la risoluzione di tale problema, nessuno di essi è migliore degli altri in assoluto, ma dipende dal tipo di applicazioni in cui vengono utilizzati.

La possibilità di avere un robot in grado di eseguire in maniera autonoma i movimenti necessari alla risoluzione dei compiti che gli vengono assegnati ha molta importanza in un gran numero di applicazioni, come la domotica o quelle applicazioni in cui il robot deve esplorare autonomamente un ambiente sconosciuto, alla ricerca, ad esempio di superstiti ad un disastro naturale. Nel problema di pianificazione del moto non esiste tuttora un metodo che sia ottimo in generale, ma piuttosto un certo numero di metodologie diverse che sono applicabili solo a casi particolari e sotto certe ipotesi.

Obiettivi

In questa tesi verrà progettato, implementato e sperimentato un metodo di pianificazioni di percorsi per un ambiente dinamico e parzialmente conosciuto. In tali ambienti la situazione è più complessa perché le informazioni che vengono lette dai sensori e di conseguenza la loro stessa topologia, sono incomplete e variano nel tempo.

Il metodo di pianificazione sarà diviso concettualmente su due livelli, avremo così la possibilità di sfruttare sia le informazioni *globali* sull'ambiente, quali quelle che potrebbero essere fornite inizialmente da un utente esterno o quelle che vengono man mano registrate dal robot stesso, sia

quelle *locali* sui cambiamenti di tale ambiente e sulla presenza di ostacoli in movimento.

Per quanto riguarda il livello di pianificazione globale, cercheremo di ottenere un grafo che rispecchia le caratteristiche topologiche dell'ambiente. Tale struttura verrà costruita in maniera probabilistica, sfruttando le informazioni, anche parziali, riguardo all'ambiente in cui è immerso il robot, e dovrà essere *dinamica*, cioè essere in grado di modificarsi in relazione a variazioni individuate nella topologia dell'ambiente.

Tale algoritmo viene sviluppato partendo da due metodi già conosciuti:

- Le *probabilistic roadmap* sono uno schema di pianificazione dei percorsi che è emerso abbastanza recentemente e ha ottenuto buoni risultati anche per robot dotati di molti gradi di libertà. In questo metodo la pianificazione è divisa in due fasi: nella prima esso genera casualmente delle configurazioni e tenta di collegarle tra loro mediante semplici percorsi facilmente calcolabili, mentre nella seconda, detta di interrogazione, il grafo così creato può essere utilizzato per trovare percorsi all'interno dello spazio delle configurazioni.

- Il *growing neural gas* è in sostanza una rete neurale ad apprendimento non supervisionato che viene attualmente utilizzata in ambiti molto diversi da quelli della pianificazione di percorso, come ad esempio l'estrazione di informazioni dalle pagine web e la ricerca di strutture in grandi quantità di dati provenienti da esperimenti scientifici. Il suo concetto di "topologia", tuttavia, si presta molto bene ad essere utilizzato nel campo del *path-planning*, sebbene necessiti di alcune importanti modifiche.

La novità principale introdotta dal nostro metodo sarà la caratteristica di essere dinamico, cioè di essere in grado di modificare la propria struttura in relazione alle variazioni di topologia individuate dalle letture sensoriali.

L'algoritmo di pianificazione locale si occuperà di ricevere in *input* un cammino dal grafo del livello superiore e di adattarlo alle caratteristiche del robot e ad alcuni criteri basilari a cui deve rispondere un percorso in questo genere di applicazioni, quali la distanza minima dagli ostacoli

e l'assenza di angolosità. Inoltre tale percorso dovrà essere reattivo alle modifiche istantanee dell'ambiente, o comunque percepite dai sensori.

Per realizzare questo algoritmo abbiamo utilizzato le *elastic band*, un metodo che permette di considerare i percorsi come soggetti ad un insieme di forze virtuali grazie alle quali vengono deformati ottenendo percorsi ammissibili per il robot. Abbiamo apportato delle modifiche a tale algoritmo per evitare alcuni problemi riscontrati in sede di implementazione, e abbiamo aggiunto la capacità di incrementare, per quanto possibile, il raggio di curvatura di questo percorso, affinché sia più agevole da seguire da parte del robot.

Ambienti di sperimentazione

Per sperimentare i due algoritmi presentati in questa tesi sono stati utilizzati degli ambienti di simulazione, grazie ai quali siamo stati in grado di correggere e migliorare il nostro metodo prima ancora di applicarlo ad un robot reale:

- E' stato sviluppato un software di simulazione indipendente all'applicazione robotica con cui è possibile intervenire su tutti i parametri degli algoritmi e vederne graficamente l'evoluzione.
- Successivamente l'algoritmo è stato integrato in un software di simulazione per la base robotica con il quale è stato possibile studiare tutta una serie di problematiche ancor prima di cominciare a sperimentare sui robot reali.

Sono stati infine condotti una serie di esperimenti sui robot fisici, al fine di validare il metodo nell'ambiente di applicazione reale. Gli ambienti di sperimentazione sono stati due:

- All'interno del progetto *RoboCare*, è stato ricostruito l'interno di un ambiente domestico, in esso utilizziamo un robot Pioneer 2 della *ActivMedia Robotics*. Un ambiente di questo tipo è costituito da ostacoli con geometrie non semplici (ad esempio ostacoli cavi o

sopraelevati) o in movimento (persone) che costringono il robot a ripianificare il suo percorso per evitare di collidere con essi.

- E' stata preparata un'arena simile a quelle utilizzate nelle competizioni di *RoboCupRescue*, in tale ambiente utilizziamo un Pioneer 3-AT. In esso si possono studiare altre caratteristiche come la presenza di passaggi stretti che impongono una certa precisione al *path-planner* o la conoscenza imprecisa della mappa. Poiché la pianificazione di percorso si appoggia su queste informazioni, deve comportarsi in maniera robusta alla possibilità di *input* incompleti o errati.

Risultati conseguiti

In tutti gli esperimenti che abbiamo effettuato, l'algoritmo per la mappa topologica diviene in pochi secondi una descrizione topologicamente equivalente all'ambiente. Da quel momento in poi rimane stabile senza perdere le informazioni acquisite. La rete è dinamica e si modifica rapidamente in risposta a variazioni topologiche dell'ambiente. Grazie a tale rete topologica siamo sempre in grado di calcolare un percorso, qualora esso esista, tra due qualunque posizioni all'interno della mappa.

I percorsi ottenuti dall'algoritmo di pianificazione *locale* sono molto naturali e simili a quelle che percorrerebbe un essere umano nelle stesse situazioni, inoltre sono reattive ad eventuali ostacoli in movimento. La caratteristica di evitare raggi di curvatura stretti lungo il percorso rende tali cammini molto agevoli da essere seguiti per il modulo di controllo.

Traccia dell'esposizione

Capitolo 1 Nel primo capitolo verrà analizzato il problema della pianificazione del percorso in maniera generale.

Capitolo 2 Verranno qui presentati i vari argomenti e i metodi che saranno poi successivamente utilizzati per lo sviluppo degli algoritmi descritti nei capitoli successivi.

Capitolo 3 Si parlerà del primo dei due algoritmi presentati in questa tesi, quello di pianificazione *globale*, che si occupa di costruire una mappa topologica dell'ambiente in cui si trova immerso il robot partendo da informazioni globali.

Capitolo 4 L'algoritmo presentato in questo capitolo si occupa invece della pianificazione *locale*, prendendo come *input* il percorso topologico generato dal metodo descritto nel capitolo precedente. Lo scopo sarà quello di trasformare tal cammino in maniera che sia ammissibile (e agevole) per il robot e renderlo reattivo ai cambiamenti percepiti dai sensori (ad esempio ostacoli in movimento).

Capitolo 5 Verrà descritto brevemente come sono stati implementati gli algoritmi descritti nei capitoli precedenti e di come ad essi siano state aggiunte altre caratteristiche utili per tenere in considerazione una mappa dinamica su cui vengono annotate le letture dei sensori.

Capitolo 6 Illustreremo i vari esperimenti che sono stati effettuati, sia in ambiente simulato che reale, e discuteremo i relativi risultati raggiunti.

Capitolo 7 Nell'ultimo capitolo, infine, si delineeranno le conclusioni della tesi e si illustreranno i possibili sviluppi futuri dei metodi descritti.

Capitolo 1

Problematiche e metodi per la pianificazione delle traiettorie per robot mobili

1.1 Introduzione al *path-planning*

Un sistema robotico ha molto spesso la necessità di interagire e muoversi nell'ambiente fisico in cui si trova. In tale ambiente sono presenti oggetti che esso deve evitare durante il movimento (ostacoli) o che deve raggiungere e manipolare. La pianificazione del moto (*path-planning* in inglese) si occupa appunto di trovare sequenze di movimenti per raggiungere un certo obiettivo evitando, in particolare, collisioni con gli ostacoli. E' opportuno, inoltre, che il robot sia in grado di svolgere questo compito in maniera autonoma e che l'utente possa assegnare i compiti in maniera *dichiarativa* (cioè specificando cosa vuole ottenere) e non *procedurale* (cioè specificando come svolgere tali compiti). E' dunque il robot stesso che deve essere in grado di pianificare i suoi movimenti dati la sua configurazione iniziale e l'obiettivo che si vuole raggiungere.

I campi di applicazione della pianificazione di percorso, in inglese *path-planning,* sono molteplici:

- l'utilizzo più comune è naturalmente quello legato alla pianifica-

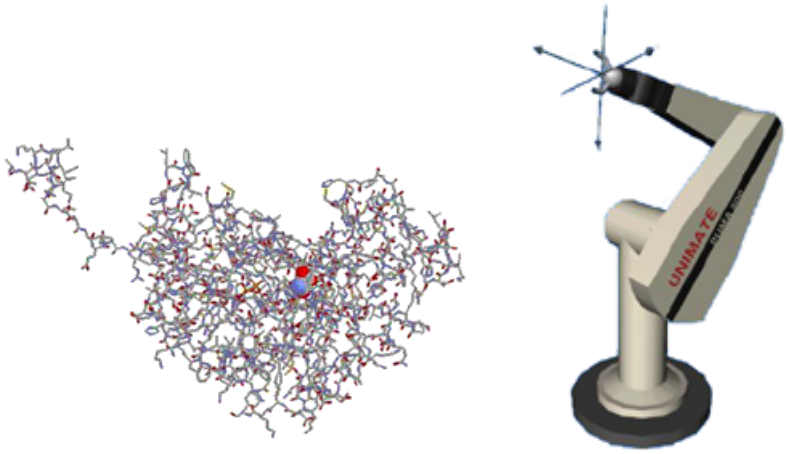


Figura 1.1: alcune applicazioni della pianificazione di movimento (biologia molecolare, braccio robotico articolato)

zione del percorso che deve seguire un robot in un ambiente, ad esempio un robot che deve portare il caffè dalla cucina all'ufficio;

- possiamo utilizzare il *path-planning* nell'analisi e nell'animazione di complessi modelli CAD tridimensionali: in questo caso la pianificazione di movimento può ridurre drasticamente il numero di dati che gli animatori devono immettere nel sistema (piuttosto che descrivere dettagliatamente un movimento di un modello umanoide, potrebbero semplicemente specificare "porta il braccio destro alla fronte");

- nella biologia molecolare e nella chimica è possibile modellare le molecole come collegamenti spaziali con molti gradi di libertà: in questi campi la pianificazione di movimento viene utilizzata per calcolare il movimento delle molecole per, ad esempio, progettare nuovi farmaci;

- in ambito industriale è spesso necessario trovare le sequenze di movimenti che devono eseguire i bracci meccanici che assemblano gli autoveicoli, gli elettrodomestici, ecc.;

- esistono alcuni studi sulla progettazione di veicoli autonomi in grado di guidare senza la presenza di un pilota umano;

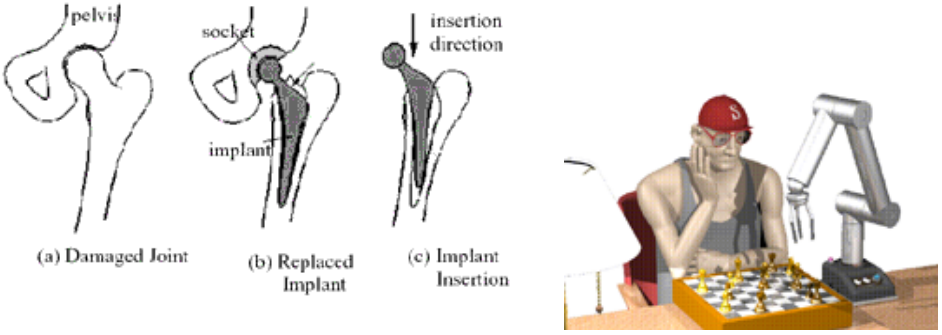


Figura 1.2: alcune applicazioni della pianificazione di movimento (studi di inseribilità di anca artificiale, animazione automatica di modelli 3D)

- anche nella medicina il *path-planning* ha una vasta applicabilità, ad esempio nella chirurgia assistita roboticamente o nello studio di inseribilità di anca artificiale.

Il problema di base, in cui l'ambiente è conosciuto e statico, può essere ulteriormente esteso al caso di ostacoli in movimento o che è possibile spostare, robot multipli che necessitano di movimenti coordinati, incompletezza o incertezza nella geometria degli oggetti coinvolti e/o nelle letture dei sensori, considerazione dei vincoli di non olonimia, ecc.

E' opportuno chiarire immediatamente che lo sviluppo di metodi automatici di pianificazione del moto presenta aspetti di elevata complessità. Infatti, il tipo di *ragionamento spaziale* che l'uomo usa in maniera istintiva quando deve muoversi o interagire con l'ambiente si è rivelato estremamente difficile da duplicare in un algoritmo interpretabile da un computer. A confermare la pluralità di aspetti coinvolti in questo problema c'è il fatto che contributi fondamentali provengono da discipline diverse, come l'Intelligenza Artificiale, la Geometria Computazionale e la Teoria del Controllo. Allo stato attuale, la pianificazione del moto costituisce un settore di ricerca con un corpo di nozioni sufficientemente consolidato, ma lungi dall'essere completo [Oriolo93].

1.1.1 Il problema generale del *path-planning*

Nel seguito definiamo il problema generale del *path planning*. Sia dato innanzitutto uno ***spazio di lavoro*** del robot, cioè uno spazio euclideo $W \equiv \mathbb{R}^n$, con $n \in \mathbb{N}$ (generalmente 2 o 3). Sia $\mathcal{A}$ la descrizione geometrica del robot e siano $\mathcal{B}_1, \mathcal{B}_2, ...$ i modelli geometrici degli ostacoli.

Consideriamo una prima istanza del problema, in cui il robot è puntiforme ed è libero di traslare liberamente in W. Assegnamo al robot una ***posizione*** in W, data dalle coordinate del punto che lo rappresenta nello spazio di lavoro. Ogni ostacolo può essere definito da un insieme di punti, la cui posizione reciproca non muta nel tempo (gli ostacoli, cioè, sono corpi rigidi). Per questo motivo, per inserire un ostacolo nello spazio di lavoro dobbiamo indicare, oltre alla posizione di un suo punto (ad esempio il baricentro), anche il suo orientamento. Definiamo ***configurazione*** la posizione insieme all'orientamento. Nel seguito indicheremo le posizioni con p e le configurazioni con q. Nel caso semplice di un robot puntiforme immerso in uno spazio bidimensionale, una posizione è una coppia (x, y) e gli ostacoli, poligoni bidimensionali fissi, hanno una configurazione (x, y, θ) in W. Se W fosse tridimensionale avremmo bisogno di una terna (x, y, z) per le posizioni e di una sestupla $(x, y, z, \theta, \varphi, \alpha)$ per le configurazioni degli ostacoli.

Supponiamo data la posizione iniziale del robot p_{init} e una posizione di arrivo p_{goal}. Il problema del *path planning* si occupa di trovare un ***cammino***, cioè una sequenza continua di posizioni, che abbia inizio in p_{init} e termini in p_{goal} e che sia priva di collisioni o contatti tra $\mathcal{A}$ e gli ostacoli $\mathcal{B}_i$[1]. In caso questo cammino non esista, questo deve essere in qualche modo segnalato.

Nel seguito useremo i termini cammino, ***traiettoria*** e ***percorso*** come sinonimi.

Definiamo W_{obst} come l'unione di tutti gli ostacoli in W ($W_{obst} = \bigcup_i \mathcal{B}_i^W$) e con W_{free}, lo ***spazio libero di lavoro***, il suo complemento in W. In questi termini possiamo dire che il problema della pianifica-

[1] in realtà in alcune applicazioni sono necessari dei *contatti* con gli ostacoli, nel caso, ad esempio, in cui il robot è un manipolatore e gli "ostacoli" siano gli oggetti da manipolare

zione di percorso è teso a trovare un cammino da p_{init} a p_{goal} contenuto completamente in W_{free}.

Estendiamo il concetto ad un robot non più puntiforme, ma di forma e dimensione definite, sebbene ancora fisse. In tal caso per inserirlo in W non è più sufficiente soltanto la sua *posizione* p, bensì abbiamo bisogno della sua *configurazione* q. In questo il cammino da trovare è una sequenza continua di *configurazioni*, tutte prive di collisioni o contatti, cioè contenuta completamente in W_{free}, che porti da q_{init}, configurazione iniziale del robot $\mathcal{A}$, fino a q_{goal}, configurazione finale.

1.1.2 Lo spazio delle configurazioni

Una ulteriore estensione al problema si ha quando il robot non è più un corpo rigido ma, ad esempio, è formato da una serie di corpi rigidi uniti tra loro da giunti che possono ruotare più o meno liberamente, ad esempio un braccio meccanico. In questi casi i limiti dei giunti e le complessità delle possibili configurazioni rende l'approccio usato finora scarsamente utilizzabile. Una forte semplificazione del problema si ottiene introducendo un nuovo concetto: lo ***spazio delle configurazioni***.

Ogni configurazione del robot nello spazio di lavoro può essere descritta da un insieme finito di parametri indipendenti. Definiamo *spazio delle configurazioni* C lo spazio in cui questi parametri sono le coordinate. In questo spazio, dunque, qualunque configurazione del robot è indicata da un punto nello spazio C. Le configurazioni illegali, perché impossibili per la struttura del robot (ad esempio le limitazioni dovute ai giunti meccanici o all'impenetrabilità fisica degli elementi del robot), o perché in collisione con degli ostacoli, sono rappresentate da particolari regioni di C dette C-ostacoli. L'unione di queste regioni forma C_{obst}. In analogia con quanto fatto nello spazio di lavoro W, anche nello spazio delle configurazioni definiamo $C_{free} = C - C_{obst}$.

Un esempio semplice di spazio delle configurazioni si ha nel caso di un robot con geometria poligonale libero di traslare ma non di ruotare. Nella figura 1.3 è possibile vedere che in questo caso gli ostacoli subiscono semplicemente una "espansione".

Nello spazio delle configurazioni, il problema della pianificazione del

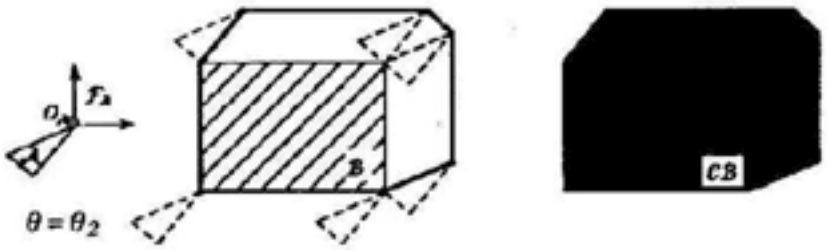
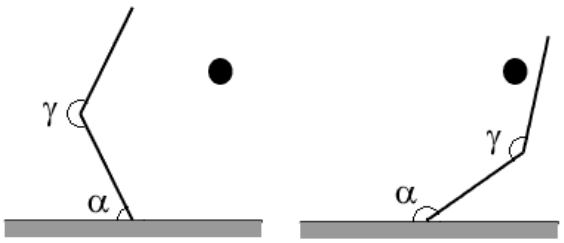


Figura 1.3: espansione degli ostacoli in C_{free}

movimento torna ad essere quello di trovare un cammino formato da punti, cioè una curva continua, come avveniva nella prima istanza del problema che avevamo definito (robot puntiforme). Sebbene questa sia una forte esemplificazione del problema, bisogna considerare che lo spazio C spesso ha un numero di dimensioni maggiore a quello di W e che le descrizioni degli ostacoli in C sono molto più complesse.



(a) configurazione iniziale

(b) configurazione finale

Figura 1.4: due configurazioni di un braccio meccanico

Per capirlo, ipotizziamo di dover trovare il percorso nello spazio di configurazioni affinché il braccio meccanico bidimensionale della figura 1.4(a) arrivi nella configurazione mostrata nella figura 1.4(b). Il braccio ha due gradi di libertà, in quanto ha due giunti (uno collegato alla base immobile e l'altro che unisce i due segmenti del braccio) che possono far ruotare i due segmenti. Lo spazio delle configurazioni, di conseguenza, avrà due dimensioni, poiché possiamo prendere come parametri, ad esem-

pio, l'angolo formato tra la base e il primo segmento, che chiameremo α, e quello formato tra i due segmenti, γ.

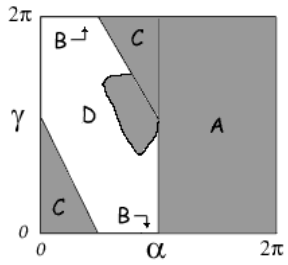


Figura 1.5: spazio C per il braccio meccanico

Nella figura 1.5 possiamo vedere la proiezione di questa situazione nello spazio C. Il primo C-ostacolo da considerare è quello dovuto al fatto che l'angolo α può variare solo tra 0 e π, proprio per la presenza della base (questo ostacolo è indicato dalla lettera A nella figura); l'angolo γ, invece, può variare tra 0 e 2π, ma non può scavalcare questi valori, poiché altrimenti i due segmenti si troverebbero ad essere sovrapposti (ostacolo B in figura). Un altro C-ostacolo è quello necessario a modellare l'impenetrabilità tra la base e il segmento non adiacente ad essa da essa (C nella figura).

Riguardo alla proiezione dell'unico ostacolo presente in W, essa, ha certamente una forma più complessa rispetto a quella in W (lettera D nella figura). Come si può vedere facilmente anche in un caso piuttosto semplice, il calcolo dei C-ostacoli è molto complesso.

1.2 Elementi di topologia matematica

Nel seguito faremo uso di concetti presi in prestito dalla topologia algebrica, una branca dell'algebra che studia i concetti di connettività, topologia, omotopia, ecc. Questi concetti sono applicabili a vari tipi di spazi algebrici, nel nostro caso siamo interessati solo a spazi euclidei. Molte delle definizioni in questa sezione sono state prese da [AlgTopWeb] e [DictWeb].

1.2.1 Introduzione alla topologia matematica

La ***topologia*** (dal greco $\tau o \pi o \varsigma$, luogo e $\lambda o \gamma o \varsigma$, studio) è una branca della matematica che si occupa dello studio degli ***spazi topologici*** e delle co-

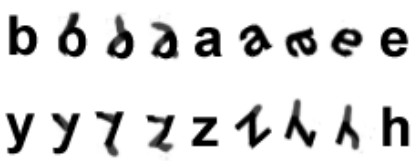


Figura 1.6: omeomorfismi di lettere alfabetiche

siddette proprietà topologiche delle figure, quali ad esempio la connettività, cioè quelle proprietà che non cambiano se sottoposte a trasformazioni continue (dette ***omeomorfismi***, da non confondersi con *omomorfismi* che in matematica hanno un altro significato). Due figure che possono essere deformate una nell'altra sono dette ***omeomorfiche*** o ***omeomorfe*** e sono considerate la stessa cosa dal punto di vista topologico. Ad esempio un cubo e una sfera sono omeomorfiche, mentre una sfera e una circonferenza non lo sono.

Un semplice esercizio introduttivo è quello di classificare le lettere dell'alfabeto latino secondo la loro equivalenza topologica. Così è possibile individuare la classe $\{a, b, d, e, g, o, p, q\}$ delle lettere con un foro, la classe $\{c, f, h, k, l, m, n, r, s, t, u, v, w, x, y, z\}$ delle lettere senza foro e la classe $\{i, j\}$ delle lettere che consistono di due parti. E' semplice trovare delle trasformazioni omeomorfiche che trasformino una qualsiasi lettera di una classe in un'altra della stessa classe (figura 1.6).

La motivazione dietro la topologia è che alcuni problemi geometrici non dipendono dalla forma esatta degli oggetti coinvolti, bensì nel "modo in cui sono connessi insieme". Uno dei primi scritti in topologia (nel 1736) è stato la dimostrazione, da parte di Leonhard Euler (Eulero), che è impossibile trovare un itinerario attraverso la città di Königsberg (ora Kalingrad) che attraversasse tutti e sette i suoi ponti esattamente una volta. Questo risultato non dipende dalla lunghezza dei ponti, né dalla distanza tra di essi, ma solo dalle proprietà di connettività: quale ponte è connesso a quale isola o argine. Questo problema, i *Sette Ponti di Königsberg*, è ora un famoso problema di matematica introduttiva.

Per affrontare questo tipo di problemi che non dipendono dalla forma

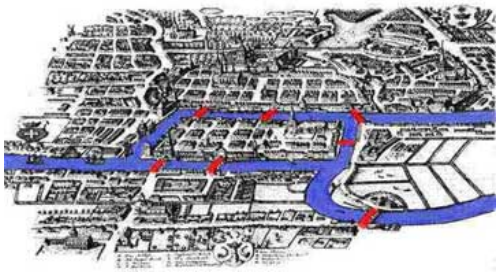


Figura 1.7: un'immagine che mostra Königsberg e i suoi sette ponti

esatta degli oggetti, bisogna essere chiari riguardo a quali proprietà questi problemi sono rivolti. La nozione di ***equivalenza topologica*** viene introdotta per questo motivo. L'impossibilità di attraversare ogni ponte solo una volta si applica ad ogni disposizione di ponti topologicamente equivalente a quelli di Königsberg. Formalmente, due spazi sono topologicamente equivalenti se esiste un omeomorfismo tra di essi. In questo caso i due spazi sono detti ***omeomorfici*** e sono considerati essenzialmente gli stessi per quanto riguarda la topologia.

1.2.2 Definizioni formali

In topologia, uno ***spazio topologico*** è definito formalmente come una coppia (X, T) in cui X è un insieme e T è una collezione di sottoinsiemi di X, che soddisfano i seguenti assiomi:

1. l'insieme vuoto e X sono in T;
2. l'unione di qualunque collezione di insiemi in T è in T;
3. l'intersezione di qualunque coppia di insiemi in T è in T.

L'insieme T è detto ***topologia*** di X. Gli insiemi in T sono detti ***insiemi aperti*** e gli elementi in X sono spesso chiamati anche ***punti***.

E' possibile definire T, in maniera più agevole, usando una ***base***. Definiamo base, di uno spazio topologico X con topologia T, un sottoinsieme B di T ($B \subseteq T$) tale che ogni elemento di T può essere scritto come unione di insiemi di B. Diciamo in questo caso che la base B *genera* la topologia T.

Una funzione tra spazi topologici è detta ***continua*** se l'immagine inversa di ogni insieme aperto è aperto. Questo è un tentativo di catturare l'intuizione che non esistono "separazioni" nella funzione.

Per una definizione formale di ***omeomorfismo***, uno dei concetti a cui siamo più interessati, supponiamo X e Y spazi topologici e sia f una funzione da X a Y. Diciamo allora che f è un omeomorfismo se e solo se:

1. f è una biiezione;
2. f è continua;
3. la funzione inversa f^{-1} è continua.

Se esiste un omeomorfismo tra due spazi X e Y, allora Y è detto omeomorfico a X. In questo caso è anche vero il viceversa, poiché anche f^{-1} è un omeomorfismo. Diciamo dunque che X e Y appartengono alla stessa classe di omeomorfismo o, più semplicemente, che sono ***omeomorfici***.

Se due spazi topologici sono omeomorfici, essi hanno le stesse caratteristiche topologiche. Ad esempio la circonferenza unitaria e il quadrato unitario in $\mathbb{R}^2$ sono omeomorfici, l'intervallo aperto $(-1, 1)$ è omeomorfo a $\mathbb{R}$ e così via. E' importante notare che due spazi sono omeomorfici se è possibile trasformare uno nell'altro *e viceversa*: durante la trasformazione non deve andare persa nessuna informazione (f, infatti, è una *biiezione*).

Un criterio più intuitivo e informale può aiutare a visualizzare meglio il concetto: due spazi sono topologicamente equivalenti se uno può essere deformato nell'altro senza tagliarne via delle parti o incollarne dei pezzi insieme: i "buchi", se presenti, rimarranno, e non ne verranno creati di nuovi.

Uno spazio topologico è detto ***connesso*** se non è l'unione di una coppia di insiemi aperti disgiunti non vuoti. Uno spazio è ***path-connesso*** se per ogni due punti $x, y \in X$ esiste un ***cammino*** p da x a y, cioè una funzione continua $p : [0, 1] \to X$ con $p(0) = x$ e $p(1) = y$. Gli insiemi *path-connessi* sono sempre connessi. Per sottoinsiemi aperti di $\mathbb{R}^n$ (ad esempio gli spazi euclidei, che sono quelli a cui siamo interessati) vale anche il viceversa.

1.2.3 La topologia negli spazi metrici ed euclidei

Concentriamo adesso la nostra attenzione sugli spazi che hanno per noi interesse. Definiamo dunque uno spazio metrico e un suo caso particolare, lo spazio euclideo.

Uno ***spazio metrico*** è un insieme X in cui sia definita, tra ogni coppia di suoi elementi, una funzione detta ***distanza***, che deve soddisfare, $\forall x, y \in X$, le seguenti condizioni:

$d(x, y) > 0$;

1. $d(x, x) = 0$;
2. se $d(x, y) = 0$ allora $x = y$ (identità degli indiscernibili);
3. $d(x, y) = d(y, x)$ (simmetria);
4. $d(x, z) \leq d(x, y) + d(y, z)$ (diseguaglianza triangolare).

Un sottoinsieme A di uno spazio metrico è detto essere ***aperto*** se e solo se, dato un punto $x \in X$, esiste sempre un $\epsilon > 0$, tale che, qualunque altro punto y di X tale che $d(x, y) < \epsilon$, appartiene a A. Definiamo, in uno spazio metrico, ***sfera aperta*** con centro in x ($x \in X$) e raggio r il sottoinsieme di X così definito:

$$B_r(x) = \{y : d(x, y) < r\}$$

Uno ***spazio euclideo*** è uno spazio metrico in cui la funzione distanza è l'usuale distanza euclidea tra due punti:

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2$$

in cui n è la dimensione dello spazio. Possiamo estendere agli spazi euclidei le definizioni di sottoinsieme aperto e sfera aperta, essendo lo spazio euclideo un caso particolare di spazio metrico.

Le sfere aperte, rispetto alla metrica d, formano una base della topologia degli spazi metrici (o euclidei). Questo vuol dire che, tra l'altro, tutti gli insiemi aperti in uno spazio metrico possono essere scritti come unione di sfere aperte.

In uno spazio metrico o euclideo, è intuitivo definire una sua topologia come l'insieme delle sfere aperte centrate nei punti dell'insieme X. Definiamo *vicinato* di un punto $x \in X$ un qualunque insieme che contiene un insieme aperto che contiene x. Possiamo definire una topologia, in alternativa alle definizioni date precedentemente, con una relazione di vicinanza tra i punti di X e le sfere aperte.

1.2.4 Omotopia

Arriviamo dunque alla seconda definizione per noi più importante, quella di ***omotopia***. In topologia, due funzione continue sono dette ***omotopiche*** se e solo se una può essere "deformata in maniera continua" nell'altra. In tal caso la deformazione viene chiamata *omotopia*.

Formalmente, una omotopia tra due funzione continue f e g da uno spazio topologico X a uno spazio topologico Y è definita come una funzione continua $h : X \times [0,1] \to Y$ dal prodotto cartesiano dello spazio X con l'intervallo $[0,1]$ allo spazio Y, in modo che, per ogni punto x di X, $h(x,0) = f(x)$ e $h(x,1) = g(x)$. L'omotopia è una relazione di equivalenza sull'insieme di tutte le funzioni continue da X a Y. La relazione di omotopia è compatibile con la composizione di funzioni nel modo seguente: se $f_1, g_1 : X \to Y$ sono omotopiche e $f_2, g_2 : Y \to Z$ sono omotopiche, allora le loro composizioni $f_2 \circ f_1, g_2 \circ g_1 : X \to Z$ sono omotopiche.

L'omotopia non deve essere confusa con l'omeomorfismo. Mentre la prima ha a che fare con trasformazioni da una *funzione* ad un'altra, il secondo è definito essere una trasformazione tra *spazi*. Inoltre, nel caso delle funzioni omotopiche non c'è bisogno che l'omotopia tra di esse sia biiettiva.

1.3 Metodi di pianificazione

Attualmente la maggioranza dei metodi e algoritmi di *path-planning* esistenti ha due caratteristiche comuni: utilizza lo spazio libero della configurazioni e procede in due fasi, nella prima, detta di preprocessamento, ci si occupa di costruire un modello dell'ambiente, variabile da metodo a

metodo, grazie al quale nella seconda fase, detta di richiesta, è possibile trovare il percorso cercato.

E' importante, inoltre, iniziare a distinguere i metodi di pianificazione ***globali*** e quelli ***locali***. Un metodo di pianificazione globale sfrutta la conoscenza riguardante *tutto* lo spazio C e le sue caratteristiche geometriche, in maniera possiamo dire "più astratta". Per quanto riguarda i metodi locali, invece, essi lavorano più "a basso livello", interagendo con le letture dei sensori del robot fisico e con eventuali ostacoli imprevisti, ma tenendo in considerazione soltanto lo spazio visibile (o vicino) al robot. Inoltre, un metodo di pianificazione globale genera il piano *prima* di eseguirlo, mentre i metodi locali vengono eseguiti *durante* l'esecuzione del movimento. Per evitare un ostacolo che si viene a trovare, o transita, sulla traiettoria del robot, è più indicato un metodo di pianificazione *locale*, che sia in grado di modificare velocemente la traiettoria per evitare collisioni (questo spesso viene integrato nel modulo di controllo del movimento, quello predisposto per interagire direttamente con l'hardware del robot); al contrario, per trovare il percorso in un labirinto, di cui i sensori possono vedere solo una piccola parte, abbiamo bisogno delle informazioni derivanti da un'eventuale mappa già conosciuta dal robot, e quindi di un metodo di pianificazione *globale*.

Sebbene, dato un metodo di pianificazione, è difficile definirlo locale o globale in maniera netta, è tuttavia possibile, osservandone le caratteristiche, individuare a quale dei due usi esso si presta meglio. E' chiaro, infatti, che un algoritmo lento sarà poco adatto per occuparsi di pianificazione locale, che necessita di scelte veloci per evitare collisioni, mentre un metodo che richiede in ingresso la descrizione di tutto lo spazio C sarà un buon candidato ad occuparsi della pianificazione globale.

Spesso in un robot non esiste un vero e proprio modulo che si occupa della pianificazione locale, ma l'onere di evitare ostacoli imprevisti o mobili è relegato principalmente al modulo di controllo del movimento, quello, cioè, che interagisce direttamente con l'hardware degli attuatori (ad esempio i motori) del robot.

I metodi di *path-planning* attualmente conosciuti e utilizzati si possono dividere in tre grandi categorie:

- metodi che effettuano una *decomposizione in celle* di C_{free}
- metodi che prevedono la costruzione di una *mappa stradale* (*roadmap*) dello spazio C_{free}
- metodi basati su un *campo di potenziale artificiale*

1.3.1 Decomposizione in celle

Questi metodi prevedono la decomposizione dello spazio C_{free} in celle, tali che la generazione di un cammino all'interno di una cella sia immediata. Inoltre, se le celle sono insiemi convessi, ogni segmento che unisce due punti appartenenti alla cella è contenuto completamente in C_{free}. A questo punto si costruisce il cosiddetto ***grafo di connettività***, nel quale esiste un nodo per ogni cella e un arco tra due nodi se e solo se le celle associate ad essi sono adiacenti. Per trovare una traiettoria è dunque sufficiente utilizzare i metodi di ricerca di cammini nei grafi, come *Dijkstra* o A^*, per trovare un cammino tra la cella in cui è contenuto q_{init} e quella in cui è contenuto q_{goal}.

Esistono due tipi di decomposizione in celle: esatta e approssimata. Nel primo tipo l'unione delle celle, che sono insiemi disgiunti tra di loro, restituisce *esattamente* C_{free}. Sebbene questo metodo sia molto importante dal punto di vista teorico, se gli ostacoli non sono poligonali sono richiesti strumenti matematici la cui complessità di calcolo è spesso proibitiva.

Le decomposizioni approssimate, invece, prevedono l'uso di celle dalla forma prestabilita la cui unione, tuttavia, non restituisce generalmente interamente lo spazio libero delle configurazioni. Le celle hanno una forma prestabilità, ma non la dimensione. In questo modo è possibile effettuare una decomposizione ricorsiva, iniziando con una griglia sufficientemente larga e infittendola localmente in modo tale da adattarsi alla geometria degli ostacoli.

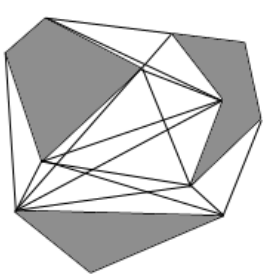
1.3.2 *Roadmap*

Una ***roadmap*** è una rete (grafo) di cammini monodimensionali contenuta in C_{free} e cattura la connettività dello spazio libero delle configurazioni. Una traiettoria, in questo caso, è ottenuta connettendo q_{init} e q_{goal} con dei nodi del grafo (generalmente quelli a loro più vicini) e poi ricercando nella rete stessa un cammino tra questi due nodi. Per ottenere quest'ultimo si possono naturalmente utilizzare i metodi di ricerca di cammini su grafi come ad esempio quello di *Dijkstra* o A^*.

I metodi con *roadmap* sono in qualche modo simili a quelli con decomposizione in celle, soprattutto per via del fatto che finiscono per fare una ricerca di cammino in un grafo. Comunque, invece di usare un grafo con nodi che descrivono sottoinsiemi di C_{free}, i grafi delle *roadmap* usano nodi che rappresentano configurazioni. Questo li rende più semplici da usare dei grafi della decomposizione in celle, tuttavia il costo da pagare è che ogni nodo del grafo in una *roadmap* contiene poche informazioni e ha quindi bisogno di più spazio per catturare la connettività di C_{free} adeguatamente.

Un pianificatore che usa una roadmap, dunque, cerca il percorso *nella roadmap*. Questo percorso è una sequenza di nodi di C_{free} usati come punti di riferimento. Questi punti di riferimento non sono, di solito, una traiettoria completa, poiché sono troppo lontani tra di loro e sono quindi richieste delle configurazioni intermedie. Questa situazione è paragonabile alle informazioni automobilistiche come "prendi la prima strada a destra, poi la seconda a sinistra e poi vai dritto fino alla fine della strada". Sono solo punti di riferimento. Una automobile non può teletrasportarsi da un punto all'altro, deve muoversi lungo le strade che collegano questi punti di riferimento.

Questi collegamenti corrispondono agli archi del grafo, per i quali, comunque, generalmente non viene memorizzata altra informazione che le due configurazioni che ognuno di essi collega. Le configurazioni intermedie verranno generate di volta in volta man mano che il robot si muove da un punto di riferimento al successivo. Questa funzione è generalmente effettuata dal cosiddetto *pianificatore locale*, che è un pianificatore a livello più basso rispetto a quello che usa la *roadmap*.



(a) un grafo di visibilità

(b) un diagramma di Voronoi

Figura 1.8: due metodi che generano *roadmap*

Il punto cruciale degli algoritmi di *roadmap* è, appunto, la costruzione della rete stessa che deve essere in grado di modellare le caratteristiche di connettività di C_{free}. Esistono vari metodi analitici per ottenere questo, i più comuni vengono detti ***grafo di visibilità*** e ***diagramma di Voronoi***.

Il primo di essi è applicabile solo al caso di ostacoli e robot poligonali (anche se è possibile estenderlo al caso di un robot sferico o ad un robot poliedrico libero di traslare ma non di ruotare) e costruisce il grafo connettendo i *vertici* degli ostacoli (nodi) se e solo se il segmento che li unisce è il lato di un ostacolo oppure se tale segmento è contenuto interamente in C_{free}. Naturalmente a tale insieme di nodi vengono aggiunti q_{init}e q_{goal}connettendoli ai vertici degli ostacoli quando questo sia possibile. La complessità degli algoritmi più efficienti per costruire il grafo di visibilità è $O(n^2)$, in cui n è il numero di vertici di tutti gli ostacoli presenti nell'ambiente.

Il ***diagramma di Voronoi***[2] si basa invece sul concetto di ***clearance*** e di ***ritrazione***. Si definisce ***clearance*** di un punto in uno spazio di configurazioni la sua distanza minima dagli ostacoli presenti nell'ambiente. Per ogni configurazione q in C_{free}, detta β la frontiera dello spazio libero delle configurazioni, vale quindi

[2]nel capitolo 2 verrà data una definizione più generale di diagramma di Voronoi, quella descritta in questo capitolo è un caso particolare

$$\text{clearance}(q) = \min_{p \in \beta} \|q - p\|$$

Abbiamo già definito il termine ***ritrazione*** nella sezione 2.1. Ricordiamo qui che essa è una applicazione suriettiva continua da uno spazio topologico X a uno spazio topologico Y, tale la sua restrizione a Y sia l'identità. In questa sede siamo interessati solo a ritrazioni che siano descrizioni topologicamente equivalenti di C_{free}, nel senso definito nella sezione appena citata.

Il principio del diagramma di Voronoi è quello di definire una ritrazione di C_{free} su una particolare rete che ha la proprietà di rendere massimo lo spazio libero (***clearance***) tra robot e ostacoli nello spazio di lavoro.

Sia dunque $\text{near}(q) = \{p \in \beta : \|q - p\| = \text{clearance}(q)\}$ il luogo dei punti che determinano il valore di clearance(q) per la configurazione q, si definisce dunque ***diagramma di Voronoi*** di C_{free} l'insieme

$$\text{Vor}(C_{free}) = \{q \in C_{free} : \text{card}(\text{near}(q)) > 1\}$$

in cui card(E) indica la cardinalità dell'insieme E. In definitiva il diagramma di Voronoi è il luogo dei punti equidistanti da due o più ostacoli.

1.3.3 Campo di potenziale artificiale

I due approcci fin qui esposti si occupano in definitiva di costruire un grafo che abbia proprietà di connettività simili a quelle dello spazio delle configurazioni (cioè sia una descrizione topologicamente equivalente).

I metodi che considerano il campo di potenziale artificiale, invece, considerano il robot soggetto ad un campo di potenziale artificiale U che viene scelto in modo da riflettere la struttura geometrica dello spazio. Una volta definito U il movimento del robot viene determinato da una forza fittizia che segue la direzione della massima discesa del potenziale.

L'idea dietro questo approccio è semplice. Quando un elettrone (avente carica negativa) è posto in un campo elettrico, sarà attratto da una carica positiva (che verrà posta dove è q_{goal}) e respinto da altre cariche negative (gli ostacoli). Partendo ad un certo punto, esso si muoverà verso

l'obiettivo, guidato dalla forza esercitata dal campo elettrico. Una configurazione è solo un punto nello spazio delle configurazioni, così come un elettrone è solo un punto in un comune spazio tridimensionale. Il campo U di potenziale artificiale viene generalmente ottenuto come sovrapposizione di un campo attrattivo verso la destinazione e un campo repulsivo generato dagli ostacoli. Questo metodo è stato applicato con successo a molti problemi di pianificazione di traiettorie. Esso si presta inoltre molto bene alla pianificazione locale. Tuttavia, aumentando il numero di gradi di libertà e la complessità degli ostacoli, trovare una appropriata funzione di potenziale diventa più difficile.

Uno dei problemi più comuni dell'utilizzo di questo metodo è la possibile presenza di *minimi locali* che farebbero arrestare il moto del robot in regioni anche distanti da q_{goal}. E' possibile risolvere questo inconveniente creando opportune funzioni di potenziale *prive* di minimi locali, oppure sviluppando procedure che permettano di evadere da tali minimi (ad esempio *simulated annealing*).

Capitolo 2

Metodologie per la mappa topologica e l'adattamento della traiettoria

2.1 La topologia nella pianificazione di percorso

Nel seguito useremo spesso le definizioni *equivalenza topologica* e *omotopia*, ma per quanto riguarda la prima bisogna fare delle importanti considerazioni. Ci troveremo spesso a che fare, per motivi di efficienza, con descrizioni dello spazio di lavoro che hanno dimensione differente rispetto a W stesso. La dimensione di uno spazio è una caratteristica topologica, infatti non è possibile trovare un omeomorfismo tra due spazi con dimensioni diverse, cioè, ad esempio, una sfera non può essere topologicamente equivalente ad una circonferenza, così come un cubo non può esserlo ad un quadrato. Abbiamo dunque bisogno di una nuova definizione che ci possa tornare utile per le applicazioni, che "rilassi" la proprietà delle equivalenze topologiche di utilizzare funzioni biiettive.

Consideriamo la superficie esterna di un cilindro $\mathcal{C}$ ed una funzione *suriettiva non biiettiva* f che la trasformi in una circonferenza $\mathcal{R}$, ad esempio proiettando il cilindro sul piano della base. Possiamo agevolmente dimostrare che se due punti sono *vicini* in $\mathcal{C}$, cioè uno appartiene

ad una sfera aperta centrata nell'altro, e viceversa, le loro immagini in $\mathcal{R}$ sono anch'esse *vicine*. Questo implica che se esiste un percorso (cioè una funzione continua) in $\mathcal{C}$ tra un punto c_{init} e un punto c_{goal}, esiste anche un percorso tra $r_{init} = f(c_{init})$ e $r_{goal} = f(c_{goal})$. Possiamo altresì dimostrare che il percorso in $\mathcal{R}$ è formato da punti che sono le immagini, secondo f, del percorso in $\mathcal{C}$. Se questo è vero per qualunque percorso, come in questo caso, diciamo che $\mathcal{R}$ è una ***descrizione topologicamente equivalente*** di $\mathcal{C}$ e si dice che la funzione suriettiva f *genera* tale descrizione.

La differenza con la definizione "canonica" di equivalenza topologica è che non è possibile, partendo da $\mathcal{R}$, ricostruire $\mathcal{C}$ mediante l'applicazione di una *funzione* (cioè una relazione che associ ad ogni elemento di $\mathcal{R}$ un solo elemento di $\mathcal{C}$). Infatti la relazione con cui abbiamo trasformato il cilindro in circonferenza, cioè la funzione generatrice della descrizione, *non è* biiettiva e di conseguenza non possiede una funzione inversa. E' da notare inoltre che con questa definizione, $\mathcal{R}$ è una descrizione topologicamente equivalente di $\mathcal{C}$, *ma non viceversa.*

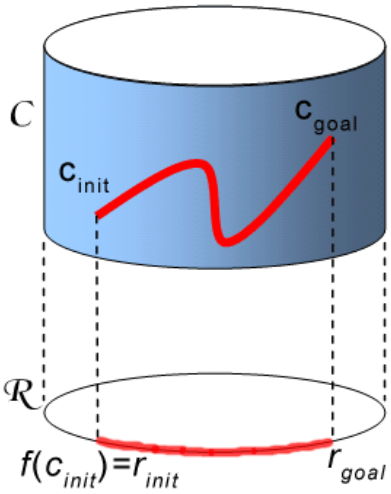


Figura 2.1: la trasformazione suriettiva da cilindro a circonferenza

Prendiamo ora il caso della superficie esterna di un cubo trasformata nei lati di un quadrato (ad esempio la base del cubo) in cui la funzione suriettiva considerata è quella che associa alle facce laterali del cubo le loro proiezioni sul perimetro di base, mentre per quanto riguarda la faccia superiore e quella inferiore, esse sono associate ad un lato scelto di tale quadrato. E' chiaro che la condizione esposte poco sopra in questo caso non valgono, cioè il quadrato *non è* una descrizione topologicamente equivalente della superficie del cubo. Infatti, ad esempio, i due punti a e b indicati in figura 2.2 sono *vicini* nel cubo, ma non lo sono nel quadrato

e quindi alla traiettoria tra a e b nel cubo non può essere associata una traiettoria valida tra le due immagini dei punti nel quadrato.

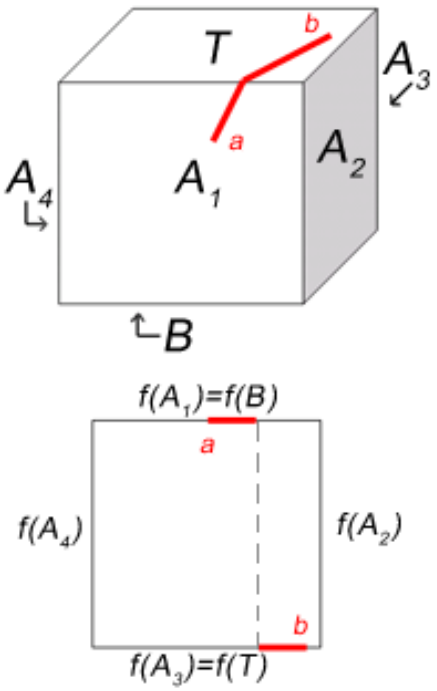


Figura 2.2: un cubo proiettato su un quadrato

Tornando al caso del cilindro, essendo f una funzione suriettiva, ne segue che esistono infinite traiettorie corrispondenti ad una stessa traiettoria sulla circonferenza. E' facile dimostrare che tutte queste traiettorie sono *omotopiche*. Questa è un'importante proprietà di questa equivalenza topologica. Dal punto di vista topologico, dunque, tutte le traiettorie sul cilindro corrispondenti alla traiettoria sulla circonferenza sono equivalenti.

Dati due punti c_{init} e c_{goal} possiamo proiettarli su $\mathcal{R}$ e cercare una traiettoria su di essa. Successivamente possiamo utilizzare utilizzare una funzione $g : \mathcal{R} \rightarrow \mathcal{C}$ per riportare tale traiettoria sul cilindro. La funzione g dovrà essere tale da associare ad ogni punto r della traiettoria sulla circonferenza, un punto appartenente alla controimmagine $f^{-1}(r)$ e in modo tale che tutti i punti di tale nuova traiettoria formino una curva continua. E' chiaro di nuovo che esistono infinite scelte per g e che tutte le traiettorie generate dalle varie g sono omotopiche.

Questo concetto diverrà importante, ad esempio, quando parleremo di ***ritrazione*** nel caso delle *roadmap*. Senza entrare, per ora, nel dettaglio di una *roadmap*, diciamo soltanto che una ritrazione $\rho : X \rightarrow Y$ tra due spazi topologici X e Y è una funzione suriettiva la cui immagine è contenuta nello spazio di partenza ($Y \subset X$) e la sua restrizione a Y è l'identità. Nel contesto di applicazione del *path-planning* siamo interessati

soltanto a ritrazioni che mantengano la topologia, cioè a ritrazioni che generino *descrizioni topologicamente equivalenti* allo spazio di partenza. Gli algoritmi descritti nei prossimi due capitoli, argomento di questa tesi, genereranno appunto questo tipo di descrizioni.

2.2 Il *path-planning* a più livelli

Il problema di permettere a un robot di essere in grado di eseguire movimenti in modo autonomo, è spesso stato affrontato in due modi diversi: pianificazione e controllo [ESBroKha00]. I primi tengono in considerazione delle informazioni *globali* dell'ambiente, mentre i secondi considerano principalmente il flusso di dati che proviene dai sensori, cioè informazioni *locali.*

I metodi di controllo spesso non sono in grado di calcolare percorsi in ambienti complessi oppure in cui la posizione finale o parte del percorso è fuori dalla portata dei sensori. D'altra parte la ripianificazione in tempo reale di una traiettoria calcolata da un pianificatore globale è necessaria per l'esecuzione del movimento in ambienti che possono cambiare in maniera imprevedibile. Questa ripianificazione deve naturalmente essere reattiva alla lettura dei sensori, che possono catturare ostacoli non previsti nella pianificazione globale.

Per questo viene spesso utilizzata una struttura a due livelli, lasciando al pianificatore globale il compito di generare un percorso plausibile secondo i suoi dati (la mappa statica) e successivamente utilizzando il *controller* per adeguare tale traiettoria alla situazione momentanea catturata dai sensori.

Il nostro pianificatore seguirà una metodologia simile: da una parte costruiremo una mappa topologica che catturi la connettività di C_{free} e dall'altra manipoleremo la traiettoria, ottenuta dalla mappa topologica, affinché abbia delle caratteristiche specifiche necessarie alla navigazione autonoma (lontananza dagli ostacoli, priva di punti angolosi, reattività a ostacoli imprevisti e/o in movimento, ecc.).

Chiameremo *pianificatore globale* l'algoritmo di livello più alto, mentre quello che ha a che fare con la traiettoria reale sarà il *pianificatore*

locale. Infine il modulo del *controller*, quello cioè che si occupa di seguire effettivamente il percorso dialogando con l'*hardware* del robot, sarà su un livello ancora più basso.

Nel seguito del capitolo descriveremo dei metodi e degli algoritmi già esistenti sia per costruire l'algoritmo di generazione della mappa topologica (sezione 2.3), che per generare la traiettoria reale (sezione 2.4).

2.3 Metodi per la mappa topologica

2.3.1 I metodi probabilistici e MonteCarlo

In alcune applicazioni i metodi analitici non sono sufficienti per trovare delle soluzioni in tempo utile. Alcuni degli algoritmi descritti nei paragrafi precedenti, ad esempio, sono molto onerosi in termini di calcolo e non possono essere utilizzati in maniera efficace per risolvere problemi di pianificazione di percorso nella realtà.

Un modo comune di risolvere questo tipo di situazione è il ricorso ai cosiddetti *metodi probabilistici*, detti anche *metodi MonteCarlo* (MC). Essi si basano sull'uso di strumenti statistici e generatori di numeri casuali per fornire soluzioni con approssimazione iterativamente decrescente.

Al contrario dei metodi analitici, che assicurano una soluzione ben definita, certa e stabile, ma solo alla fine dell'algoritmo, i metodi MC ci permettono di avere velocemente soluzioni che, sebbene siano imprecise, si avvicinano probabilisticamente (cioè aumentando il numero di iterazioni) a quella esatta. Per questo motivo i metodi MC non possono essere definiti *completi*, ma si dice che essi sono *completi probabilisticamente* (cioè per un numero di interazioni tendente a ∞).

Un esempio banale ci permette di spiegare meglio questi concetti. Supponiamo di dover calcolare l'età media di un gruppo di persone. Un metodo analitico, ovvio, è quello di sommare le età di tutti gli individui e poi dividere per la cardinalità del gruppo. Un metodo probabilistico potrebbe essere, invece, quello di selezionare casualmente una persona ad ogni iterazione e tenere una media parziale, che fornisce una soluzione

approssimata che si avvicina, con l'aumentare delle persone interrogate, a quella ottenibile col metodo analitico. Tuttavia, sebbene approssimata, essa è sempre disponibile.

Un classico esempio che viene utilizzato per comprendere l'utilizzo dei metodi MC è quello del calcolo di π. Supponiamo di avere un cerchio di raggio r e di dividerlo in quattro settori uguali. Iscriviamo uno di questi settori in un quadrato. Il lato del quadrato sarà chiaramente pari a r (figura 2.3). Se immaginiamo di prendere casualmente dei punti all'interno del quadrato, come se lanciassimo delle frecce in maniera casuale a questo ipotetico bersaglio quadrato, non è difficile comprendere che il numero di frecce che colpiscono una porzione della figura (ad esempio il quadrante del cerchio) è proporzionale alla sua area (legge dei grandi numeri). Sappiamo che l'area del quadrante del cerchio è $\frac{1}{4}\pi r^2$, quella del quadrato in cui è inscritto è, ovviamente, r^2. Di conseguenza, il rapporto tra le due aree, pari al rapporto tra frecce che colpiscono il settore circolare e il numero di frecce totali, è pari a $\frac{1}{4}\pi$. Per calcolare una stima del valore di π è quindi sufficiente moltiplicare per 4 il rapporto tra le frecce che colpiscono il quadrante del cerchio e il numero totale di frecce lanciate [MCWeb].

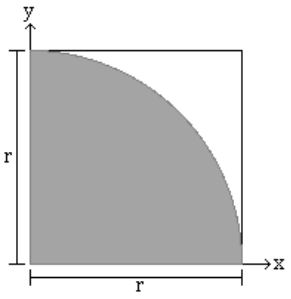


Figura 2.3: calcolo MonteCarlo del π

Attualmente i metodi MC vengono utilizzati in una grande varietà di situazioni, dall'economia alla fisica nucleare o al controllo del traffico. Chiaramente il modo in cui vengono utilizzati varia da una applicazione all'altra, ma i fattori comuni, cioè la generazione di numeri casuali e l'approssimazione iterativa del risultato, restano gli stessi. E' opportuno osservare che esistono alcuni problemi per cui i metodi MC rappresentano l'unica possibilità di risoluzione, poiché per via analitica non sono trattabili.

La caratteristica di avere una soluzione approssimata fin dalle prime iterazioni degli algoritmi MC può essere assimilata al caricamento delle pagine *web* durante la navigazione in *Internet*. I *browser* sono costruiti in modo da visualizzare immediatamente sullo schermo i dati ricevuti,

anche se incompleti. Questo si applica anche ad alcune immagini (le *jpeg progressive* contengono varie versioni della stessa immagine con fattori di nitidezza crescenti, per permettere di visualizzare immediatamente un'immagine poco chiara e, man mano che si ricevono ulteriori dati, di migliorarne i dettagli). In questo modo l'utente ha la possibilità di passare da una pagina all'altra velocemente, senza dover aspettare il caricamente totale di ciascuna di esse.

Abbiamo scelto di utilizzare un algoritmo probabilistico per un motivo simile. Come l'utente che naviga nelle pagine *web*, vogliamo essere in grado di prendere decisioni, anche se sulla base di dati imprecisi, in maniera molto veloce. Se poi abbiamo bisogno di maggiore dettaglio, sarà sufficiente attendere un certo numero di iterazioni (così come il navigatore in *Internet*, una volta raggiunta una pagina interessante, ne attende il caricamento completo).

Una seconda motivazione è il fatto che i dati che riceviamo in ingresso, ad esempio dai sensori del robot, sono dinamici (perché lo è l'ambiente) e imprecisi. Un metodo analitico classico necessita di tutti gli *input* all'inizio dell'algoritmo e fornisce la soluzione completa alla fine. In caso i dati subiscano delle variazioni, è necessario riavviare l'algoritmo dall'inizio, a meno di non cercare, analiticamente, i risultati ottenuti dagli *input* che hanno subito variazioni e modificarli, fatto che è certamente dispendioso dal punto di vista del tempo di calcolo.

Spesso è molto più utile una soluzione approssimata, ma *subito*, piuttosto che una soluzione esatta ma in ritardo. E' opportuno, comunque, accostare i metodi probabilistici a quelli analitici nelle situazioni in cui invece non possiamo permetterci di sbagliare. Utilizzeremo infatti un algoritmo probabilistico per il pianificatore *globale*, attraverso il quale prenderemo decisioni che dovranno comunque essere successivamente valutate dal pianificatore locale, prima di essere eseguite.

2.3.2 Le reti auto-organizzanti

Abbiamo bisogno di un algoritmo in grado di costruire una rappresentazione topologicamente corretta dell'ambiente di lavoro, che sia semplice e veloce da utilizzare. Una classe di algoritmi che riducono la dimensio-

nalità degli spazi di *input*, e quindi la sua complessità, sono le cosiddette reti auto-organizzanti.

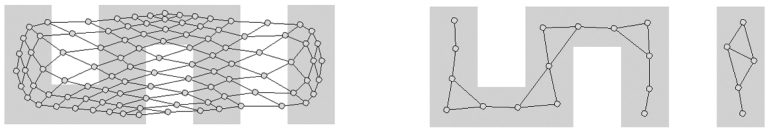
Il concetto di auto-organizzazione, che riguardi sistemi matematici o organismi biologici, è piuttosto complesso e sfocia nella fisica e nella filosofia ([SCOFoe], pp. 51-69 e 191-203). Per quel che ci riguarda, un sistema auto-organizzante è un sistema in grado di creare al suo interno una struttura ordinata a partire dal disordine. Il significato di "ordine" e "disordine" dipende dal tipo di contesto. L'informazione a cui ci riferiamo è naturalmente la struttura dello spazio di lavoro, o, meglio, la sua topologia.

Le reti auto-organizzanti sono un caso particolare di reti neurali che sfruttano il concetto di apprendimento non supervisionato. Questo significa che, al contrario delle comuni reti neurali a retropropagazione, ad esempio, non esistono *output* desiderati con cui si può addestrare la rete a rispondere dati certi *input*. Un possibile obiettivo, in questo caso, è proprio la riduzione della dimensionalità dello spazio di *input*. E' opportuno che nello spazio ridotto siano presenti tutti o gran parte degli *input* e che essi mantengano la stessa topologia che hanno nello spazio di partenza, cioè che due *input* che sono in qualche modo correlati nello spazio di partenza, lo siano anche nello spazio ridotto.

Gli *input* presentati alla rete sono elementi scelti casualmente dallo spazio di ingresso. Questo elemento di scelta casuale rende questi algoritmi *probabilistici*, nel senso descritto nel paragrafo 2.3.1. Esiste in questi ambiti anche il concetto di *distribuzione di probabilità* degli ingressi, $P(\xi)$, che porta le reti auto-organizzanti a "concentrarsi" (cioè usare una struttura più dettagliata o complessa) su regioni dello spazio di *input* i cui valori vengono presentati con più frequenza.

Le ***self-organizing map*** (SOM) di T. Kohonen [SOMKoh97, SOMWeb] sono uno dei più famosi esempi di reti neurali auto-organizzanti. Queste reti sono dette **mappe a preservazione della topologia** perché c'è una struttura topologica imposta ai nodi della rete.

Con una semplice SOM, ad esempio, è possibile strutturare un insieme di colori, presentati alla rete come vettori di tre componenti (rosso, verde e blu). La rete finale avrà colori simili posti in relazione tra di



(a) (b)

Figura 2.4: una SOM (a) e un GNG (b) usati sulla stessa topologia

loro (cioè, parlando in termini di grafi, connessi da un arco). Le reti di Kohonen vengono usate attualmente per ricavare informazioni dalle pagine web e per ottenere strutture da grandi quantità di dati, nonché nel processamento del linguaggio naturale, ecc.

Il fatto che queste reti mantengano una certa topologia potrebbe essere utile per la costruzione di una mappa topologica di C_{free}. In questo caso i nodi potrebbero essere posizioni o configurazioni e gli archi potrebbero collegare configurazioni vicine o, per lo meno, raggiungibili le une dalle altre.

2.3.3 Il *neural gas* e il *growing neural gas* (GNG)

Una delle limitazioni principali delle SOM di Kohonen (soprattutto per quanto concerne l'uso nel contesto della pianificazione di percorso) è la loro struttura fissa: il numero dei nodi e gli archi di connessione sono scelti all'inizio e non variano durante l'applicazione dell'algoritmo. Per risolvere questo problema sono state effettuate varie modifiche all'algoritmo iniziale (ad esempio, [SOMVleKokOve93]), per far sì che la topologia della rete potesse essere modificata.

Martinetz e Schulten, in [NGMarShu91], hanno presentato il *neural gas*, una rete auto-organizzante priva di struttura, sebbene il numero di nodi fosse deciso a priori. La mancanza di struttura, tuttavia, può portare ad altri problemi (i nodi vengono considerati correlati se si trovano vicini nello spazio di lavoro, questa assunzione, tuttavia, a volte può risultare errata). L'utilizzo della "legge competitiva di Hebb" ha permesso a B. Fritzke di estendere questo algoritmo, sviluppando il ***growing neu-***

Algoritmo 1 L'algoritmo del *growing neural gas*

$\mathcal{N} \leftarrow$ due nodi a e b con due posizioni casuali w_a e w_b; $\mathcal{E} \leftarrow (a, b)$
$\xi =$ un segnale di ingresso in accordo con $P(\xi)$
sia s_1 il nodo più vicino a ξ e sia s_2 il secondo nodo più vicino
$age(e) = age(e) + 1$ per ogni arco e collegato a s_1
$\Delta w_{s_1} = \epsilon_b(\xi - w_{s_1})$ e $\Delta w_n = \epsilon_n(\xi - w_n)$ per ogni n vicino di s_1
if s_1 e s_2 sono connessi da un arco e **then**
  si pone $age(e) = 0$
else
  $\mathcal{E} \leftarrow (s_1, s_2)$ (legge di Hebb)
end if
vengono rimossi tutti gli archi con età maggiore di un certo parametro, se durante questa rimozione qualche nodo rimane senza archi, si elimina anche questo nodo
se il numero di segnali di ingresso è multiplo di un certo parametro λ, si aggiunge un nuovo nodo nel modo descritto nel testo

ral gas (GNG, [GNGFri95]), in cui né il numero di nodi, né le connessioni devono essere stabilite a priori, perché vengono decise dall'algoritmo stesso.

La differenza è sostanziale, come si può notare nella figura 2.4, in cui sono visualizzate una SOM e un GNG utilizzati sullo stesso spazio di *input* (le figure sono generate utilizzando l'applet Java presente sul sito [GNGDemoWeb]). La SOM, al contrario del GNG, non solo considera correlati degli ingressi che nella realtà non lo sono, ma fallisce anche nella semplificazione dello spazio, in quanto alla fine dell'algoritmo sono presenti dei nodi che non corrispondono a nessun *input* proposto alla rete.

L'algoritmo di B. Fritzke ha due regole principali. La prima riguarda i nodi: per ogni *input* x vengono adattati (avvicinati) i k nodi più vicini ad esso (k è un valore decrescente). La seconda è che ad ogni ingresso x vengono connessi con un arco i due nodi ad esso più vicini (legge di Hebb), memorizzando dunque la loro correlazione nello spazio di *input*.

Il ***growing neural gas*** è poi completato da una serie di norme volte

a migliorare le sue caratteristiche.

Valore di errore locale Ogni nodo possiede una variabile che indica la distanza massima a cui un *input* ha selezionato il nodo come "più vicino". Questa indicazione sarà necessaria poi per decidere riguardo alla posizione dei nuovi nodi.

Rimozione di archi obsoleti Ogni volta che viene fornito un ingresso, viene selezionato il nodo più vicino w. A tutti gli archi che partono da questo nodo viene incrementata una variabile: se questa raggiunge un certo valore massimo, l'arco viene considerato obsoleto e rimosso dalla rete.

Aggiunta di nuovi nodi Quando il numero di iterazioni è un multiplo di un certo parametro λ, viene aggiunto un nuovo nodo con il seguente criterio: viene selezionato il nodo con valore di errore locale più alto a e, tra i suoi vicini (cioè i nodi collegati ad esso), ancora, quello con errore locale maggiore b, il nuovo nodo n viene inserito in una posizione media tra a e b e l'arco ab viene sostituito da due nuovi collegamenti an e nb. In questo modo non è necessario decidere a priori il numero di nodi da utilizzare, questo sarà funzione della complessità dello spazio di ingresso e del criterio di terminazione dell'algoritmo.

Criterio di terminazione dell'algoritmo Se questo non viene indicato, l'algoritmo continua a iterare all'infinito. Un possibile criterio potrebbe essere relativo alla dimensione della rete ottenuta (ad esempio un massimo numero di archi) o ad un minimo errore globale (media o somma di quelli locali) ottenuto.

2.3.4 Le *probabilistic roadmap* (PRM)

Il metodo delle ***probabilistic roadmap*** [PRMKavLat98] si prefigge di catturare la connettività dello spazio C_{free}, utilizzando un algoritmo probabilistico. Fin dalla sua introduzione, nel 1994, questo metodo ha ottenuto ottimi risultati soprattutto utilizzandolo con robot con molti gradi di libertà.

Come molti algoritmi di *roadmap* esso è costituito di due fasi: nella prima viene costruita la rete, nella seconda, detta di interrogazione, è possibile chiedere al pianificatore di trovare una traiettoria ammissibile tra due configurazioni date.

L'idea alla base dell'algoritmo delle PRM è quella di costruire la *roadmap* nello spazio delle configurazioni generando casualmente dei punti in C_{free}, che saranno i nodi del grafo, e connettendoli tra loro mediante un pianificatore locale non necessariamente completo, ma possibilmente molto veloce.

Algoritmo 2 L'algoritmo base delle PRM

$N \leftarrow \emptyset$, $E \leftarrow \emptyset$
loop
  $q \leftarrow$ una posizione di C_{free} scelta casualmente
  $N_q \leftarrow$ un insieme di nodi vicini a q scelti in N
  $N \leftarrow N \bigcup \{q\}$
  for all $q' \in N$, in ordine crescente di $D(q, q')$ **do**
    if $\neg$same_connected_component$(q, q') \wedge \Delta(q, q')$ **then**
      $E \leftarrow E \bigcup \{(q, q')\}$
    end if
    aggiorna le componenti connesse di R
  end for
end loop

L'algoritmo in pseudo-codice è indicato in figura. Come è possibile vedere, è necessario definire alcune scelte relativamente ad alcune funzionalità.

Generazione di configurazioni casuali Questa è una delle caratteristiche delle PRM che ha un impatto fondamentale sul risultato finale del metodo ed esistono infatti molte pubblicazioni sull'argomento. Nella prima versione vengono generate delle configurazioni in C_{free} con distribuzione di probabilità costante. Questo metodo, tuttavia, si è rivelato poco efficiente, soprattutto per la presenza dei cosiddetti "passaggi stretti", cioè regioni di C_{free} con un volume

piccolo (e di conseguenza una bassa probabilità che vengano generate configurazioni appartenenti ad esse) ma fondamentali per la completa connettività dell'ambiente.

La scelta di N_q Ogni volta che viene generata una nuova configurazione q, si deve tentare di collegarla, mediante il pianificatore locale, con i nodi "vicini" a q. La scelta di questi nodi può avere un impatto notevole sulle prestazioni dell'algoritmo. Uno dei criteri più semplici da utilizzare è la distanza. E' chiaro, infatti, che più due configurazioni sono distanti, meno è probabile che possano essere collegate dal pianificatore locale. L'algoritmo delle PRM aggiunge un altro criterio: si tenta di collegare due configurazioni soltanto se in quel momento esse non appartengono alla stessa componente connessa. Questo è un fatto importante, di cui discuteremo quando confronteremo questo metodo con il metodo DPTM.

Il pianificatore locale Il pianificatore locale è quello che si occupa di collegare la configurazione appena inserita con quelle appartenenti a N_q. Sono possibili naturalmente moltissime scelte, tuttavia molti esperimenti hanno dimostrato che le prestazioni migliori si hanno con pianificatori locali molto semplici e veloci, anche se non deterministici (che non assicurano di trovare un percorso tra due configurazioni se questo esiste). Il pianificatore utilizzato nella versione classica delle PRM si limita a tentare di collegare le due configurazioni mediante un segmento di linea retta, controllando che questa non subisca collisioni con gli ostacoli.

La costruzione della *roadmap* poi procede con una fase detta di "espansione" in cui vengono selezionati dei nodi che secondo un'euristica si trovano in posizioni critiche della mappa, e aggiunte delle configurazioni in prossimità di questi nodi. In questo modo la densità di configurazioni nelle regioni critiche tende ad aumentare.

La fase di interrogazione procede come la maggioranza di algoritmi di *roadmap*: q_{init} e q_{goal} vengono connesse a due nodi della rete (ad esempio utilizzando proprio il pianificatore locale) e successivamente viene cerca-

to un cammino all'interno della PRM. Tale percorso può necessitare di essere ottimizzato per permettere al robot di seguirlo.

2.3.4.1 Varianti delle PRM

Dato il loro particolare successo, le PRM sono state utilizzate in un gran numero di applicazioni diverse e sono state prese in considerazione alcune varianti dell'algoritmo iniziale, che ne incrementano l'efficienza almeno in applicazioni specifiche. La caratteristica che è stata maggiormente studiate è proprio il metodo di generazione casuale di configurazioni, poiché esso risulta essere una scelta fondamentale per ridurre, anche drasticamente, il tempo in cui l'algoritmo riesce a produrre una rete connessa. Tra le metodologie più utilizzate citiamo l'utilizzo di generazioni *quasi*-casuali di configurazioni (vedi [PRM-Q]), cioè sequenze deterministiche costruite in modo da migliorare l'efficienza dell'algoritmo (le tecniche di generazione *quasi*-casuale vengono attualmente utilizzate in molti algoritmi MonteCarlo, con ottimi risultati). E' opportuno notare, inoltre, che ogni "generazione casuale", all'interno di un calcolatore, è in realtà *pseudo*-casuale, cioè deterministica; è quindi logico voler cercare un determinismo più adatto agli algoritmi in cui viene utilizzato. Altri importanti strumenti nella scelta delle configurazioni sono il *bridge test* e il *gaussian sampling*, di cui parleremo diffusamente nel paragrafo 3.3.6.

Esiste poi un tipo di PRM chiamata "a interrogazione singola" (*single-query*) che costruisce un grafo ad ogni singola richiesta di un percorso. Questa tecnica viene spesso utilizzata nella cosiddetta pianificazione cinedinamica (*kynodynamic*), che tiene in considerazione i vincoli cinematici e dinamici a cui sono sottoposti i robot (ad esempio i loro vincoli di non-olonimia o la considerazione della massa e delle accelerazioni dei singoli componenti di cui sono costituiti). I *rapid-exploring random tree* ([RRTKufLV]) sviluppano questo concetto in maniera parallela alle PRM, giungendo a risultati praticamente identici.

2.4 Metodi per la generazione del percorso ammissibile

La traiettoria ottenuta dalla ricerca sul grafo della *roadmap* necessita di una serie di adattamenti perché possa essere agevolmente seguita da un robot mobile. Il fatto che il percorso topologico sia formato da una serie di segmenti appartenenti al grafo, infatti, rende questa traiettoria ricca di angolosità (difficili da seguire per il robot) e comunque non tiene in considerazione caratteristiche per noi importanti come ad esempio la distanza dagli ostacoli. Il percorso, dunque, deve essere ottimizato.

2.4.1 Le *elastic band*

L'algoritmo delle ***elastic band*** [ESQuiKha93] viene introdotto per "eliminare il divario tra la pianificazione globale e il controllo locale". Sebbene, infatti, sia possibile progettare un controllo che segua un percorso pianificato con uno dei metodi visti precedentemente, l'uso delle *elastic band* permette, con un solo algoritmo, di migliorare la traiettoria generata dal pianificatore e di modificarla *real-time* per eventuali ostacoli in movimento o imprevisti.

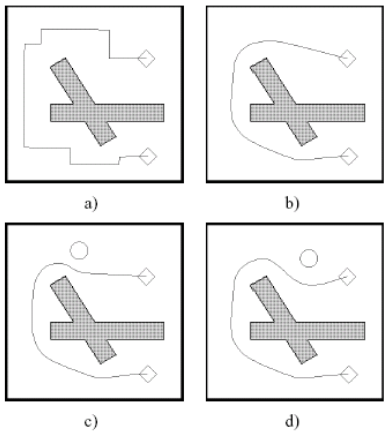


Figura 2.5: la deformazione di una *elastic band* dovuta ad un ostacolo in movimento

La struttura proposta dal citato [ESQuiKha93] è a tre livelli: nel primo livello opera il pianificatore di percorso globale, che genera una traiettoria nello spazio libero delle configurazioni (o nello spazio libero di lavoro); successivamente l'*elastic band* deformerà questo percorso per privarlo delle angolosità e per trattare gli eventuali ostacoli non considerati dal pianificatore; infine un controllo di basso livello seguirà la traiettoria generata dall'*elastic band*, comandando l'hardware degli attuatori.

L'algoritmo considera il percorso come se ogni suo punto sia soggetto a due forze virtuali. La prima simula la contrazione che si avrebbe se il percorso fosse fisicamente un elastico (da qui il nome del metodo) i cui estremi siano fissati su q_{init} e q_{goal}. In questo modo vengono smussate le eventuali angolosità della traiettoria. La seconda forza, invece, tende ad allontanare l'elastico dagli ostacoli, aumentandone dunque la *clearance*. L'applicazione di queste due forze deformerà il percorso finché non viene raggiunto un equilibrio. In questo modo si è in grado di trattare la comparsa di ostacoli imprevisti o in movimento, poiché l'*elastic band* si limiterà a deformarsi per allontanarsi da essi (vedi figura2.5) . Le trasformazioni effettuate sulle *elastic band* ad opera di queste forze sono omotopiche, quindi la traiettoria viene deformata all'interno della stessa classe di omotopia. Questo implica che nel momento in cui la topologia dell'ambiente cambia e non esista più nessun percorso ammissibile omotopico a quello corrente, sarà necessario richiedere un nuovo percorso topologico al pianificatore globale. Questo accade, ad esempio, quando viene chiusa una porta attraverso cui passa la traiettoria corrente. Se esiste un nuovo percorso, magari attraverso un'altra porta, l'*elastic band* non è in grado di trovarlo, perché questo non è omotopico al precedente.

2.4.1.1 Le "bolle"

Il problema principale da affrontare è che le *elastic band* sono delle curve, ma, per motivi di implementazione, devono essere rappresentate da un insieme finito di punti. Abbiamo dunque bisogno di un algoritmo che ci permetta, dati questi punti, di allontanare la curva nella sua interezza dagli ostacoli. Il concetto di "bolla" (*bubble*) offre un metodo efficiente per mantenere le traiettorie nello spazio libero. Invece di utilizzare e rappresentare l'intero spazio libero, ci limitiamo a generare, *run-time*, dei sottoinsiemi locali di tale spazio. Tali sottoinsiemi vengono chiamati ***bolle***.

Consideriamo la *clearance* del robot in un punto qualunque, come definita in precedenza (la distanza dall'ostacolo più vicino). Se un robot ha *clearance* ρ, questo implica che esso può muoversi in ogni direzione fino ad una distanza ρ garantendo di non collidere con nessun ostacolo. Data una configurazione $\mathbf{b}$, definiamo una bolla B come quel sottoinsieme

(chiaramente $B \subset C_{free}$)

$$B(\mathbf{b}) = \{q \in C_{free} : \|\mathbf{b} - \mathbf{q}\| < \text{clearance}(\mathbf{b})\}$$

Una *elastic band* è rappresentata da una serie finita di bolle. Per assicurarci che venga generato un percorso nello spazio libero imponiamo la condizione che le bolle consecutive si sovrappongano. Finché il percorso rimane all'interno delle bolle, sarà libero da collisioni.

Un'importante caratteristica della rappresentazione dell'*elastic band* con le bolle è che la complessità della rappresentazione è proporzionale alla complessità della situazione. Quando un robot è lontano dagli ostacoli, le bolle tendono ad essere grandi e distanziate (e quindi poche), in caso contrario divengono più piccole e numerose.

2.4.1.2 Manipolare le bolle

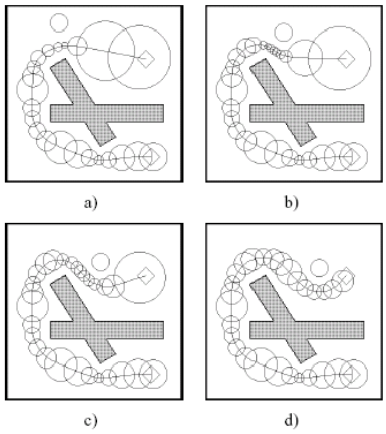


Figura 2.6: le bolle lungo l'*elastic band*

Descriveremo ora in che modo l'*elastic band*, rappresentata da una serie di bolle, viene deformata. L'implementazione più semplice è quella di considerare una bolla alla volta muoverle a turno. Per mantenere l'*elastic band* una traiettoria ammissibile, imponiamo che ogni bolla debba sovrapporsi sempre con le sue vicine. Questo vincolo può richiedere l'inserimento di nuove bolle man mano che il percorso viene deformato. Inoltre è preferibile rimuovere bolle ridondanti per incrementare l'efficienza. La direzione e il modulo della forza a cui ogni bolla è sottoposta è determinata considerando una forza artificiale. Essa è ottenuta come somma di una forza di contrazione (elastica), che chiamiamo interna, e una forza di repulsione (dagli ostacoli), che chiamiamo esterna.

Per quanto riguarda la prima, essa modella la tensione in una banda elastica fisica. Possiamo dunque utilizzare la seguente equazione:

$$\mathbf{f}_c = k_c \left(\frac{\mathbf{b}_{i-1} - \mathbf{b}_i}{\|\mathbf{b}_{i-1} - \mathbf{b}_i\|} + \frac{\mathbf{b}_{i+1} - \mathbf{b}_i}{\|\mathbf{b}_{i+1} - \mathbf{b}_i\|} \right)$$

in cui k_c è il guadagno globale di contrazione. Possiamo interpretare questa forza come quella generata da una serie di molle tra le bolle. Essa è normalizzata per riflettere una tensione uniforme lungo la *elastic band*.

La forza repulsiva deve spingere via le bolle dagli ostacoli, per incrementare la *clearance* dei punti della traiettoria. La grandezza della bolla dà una indicazione di quanto il robot è vicino alla collisione. Definiamo dunque la forza repulsiva in modo che questa accresca la dimensione di ogni bolla:

$$\mathbf{f}_r = k_r(\rho_0 - \rho(\mathbf{b}))\frac{\partial \rho(\mathbf{b})}{\partial \mathbf{b}} \qquad \text{se } \rho(\mathbf{b}) < \rho_0$$

$$0 \qquad \text{altrimenti}$$

dove k_r è il guadagno globale di repulsione, $\rho(\mathbf{b})$ la *clearance* della bolla, cioè la distanza dall'ostacolo più vicino al suo centro, e ρ_0 la distanza fino alla quale vogliamo che la forza repulsiva sia applicata.

Per approssimare $\frac{\partial \rho(\mathbf{b})}{\partial \mathbf{b}}$ possiamo utilizzare

$$\frac{\partial \rho(\mathbf{b})}{\partial \mathbf{b}} = \frac{1}{2h} \begin{bmatrix} \rho(\mathbf{b} - h\mathbf{x}) - \rho(\mathbf{b} + h\mathbf{x}) \\ \rho(\mathbf{b} - h\mathbf{y}) - \rho(\mathbf{b} + h\mathbf{y}) \end{bmatrix}$$

in cui possiamo porre h, l'intervallo di discretizzazione, pari a $\rho(\mathbf{b})$.

Dopo aver calcolato la forza, abbiamo bisogno di un metodo per aggiornare la posizione di una bolla. Una scelta semplice è $\mathbf{b}_{new} = \mathbf{b}_{old} + \alpha \mathbf{f}_{tot}$. Un possibile valore di α è $\rho(\mathbf{b}_{old})$, in questo modo la bolla si muove di una distanza proporzionale alla sua dimensione. Questa idea è dovuta al fatto che poiché le bolle devono sovrapporsi, le piccole bolle dovrebbero muoversi meno di quelle grandi. L'equazione di aggiornamento sopracitata implementa una forma di ricerca a discesa del gradiente per trovare il punto di equilibrio per la *elastic band*.

L'aggiungere bolle per mantenere la connettività della banda elastica e il rimuovere quelle ridondanti può portare a un effetto collaterale indesiderabile: in certe situazioni è possibile che una bolla venga creata

in un punto, migri lungo la banda elastica e venga eliminata in un altro punto. La sequenza potrebbe continuare indefinitamente e dunque la traiettoria risulterebbe instabile. La soluzione a questo effetto indesiderato è di considerare, della forza totale, solo la componente perpendicolare al percorso.

2.4.1.3 Varianti delle *elastic band*

Le ***elastic strip*** ([ESBroKha00]) sono un miglioramento delle *elastic band* in cui le bolle sono rese più complesse di semplici sfere (o ipersfere). Se, infatti, il robot non è facilmente approssimabile ad una sfera, l'utilizzo indiscriminato delle *elastic band* porterebbe a scartare percorsi in realtà ammissibili. In[ESWanGra01] è implementato il metodo delle *elastic strip* per un robot di aiuto per la deambulazione. In questa pubblicazione è interessante notare da un lato la considerazione dei vincoli di non olonimia del robot e dall'altro una discussione su dettagli implementativi importanti.

Due di questi sono la condizione di disconnessione e quella di ridondanza. Avevamo detto che è necessario che le bolle lungo la traiettoria si sovrappongano, per assicurare che la traiettoria stessa sia sempre completamente contenuta all'interno delle bolle. E' altresì opportuno eliminare bolle ridondanti, per aumentare l'efficienza.

Nella pubblicazione sopra citata vengono indicate le seguenti condizioni. Sia r_{i-1} il raggio della bolla precedente a quella che stiamo considerando e sia $d(p_{i-1}, p)$ la distanza tra i centri delle due bolle (in [ESWanGra01] viene in realtà utilizzata la distanza non olonoma, $\mathrm{nhd}(p_{i-1}, p)$, perché, come abbiamo detto, vengono considerati i vincoli di non olonimia del robot). Dichiariamo che una bolla è connessa se il suo centro si trova all'interno della bolla che la precede (chiaramente questo non vale per la prima), dunque essa è disconnessa se

$$d(p_{i-1}, p_i) \geq r_{i-1}$$

In tal caso è necessario aggiungere bolle tra le due tra cui si è formata la disconnessione, per riconnettere la banda.

Analogamente dichiariamo una bolla p "ridondante" se

$$d(p_{i-1}, p_{i+1}) \leq r_{i-1}$$

in tal caso essa verrà semplicemente eliminata.

L'algoritmo risultante è dunque molto semplice:

1. Controllo della presenza di bolle disconnesse ed eventuale creazione di nuove bolle
2. Controllo di ridondanza ed eventuale eliminazione di bolle
3. Modifica della traiettoria

Per quanto riguarda quest'ultimo punto, il calcolo della forza agente su ciascuna bolla è leggermente differente da quello presentato in [ESQuiKha93]. Per la forza esterna, quella dovuta agli ostacoli, è infatti proposta la seguente formula: detta d la distanza dall'ostacolo più vicino e w la distanza massima a cui agisce la forza esterna, quest'ultima è data da

$$f_{ext} = \tan\left(\frac{\pi}{2}\left(1 - \frac{d}{w}\right)\right)$$

se $d < w$, 0 altrimenti.

Per la forza interna contrattiva, invece, viene utilizzata quest'altra formula:

$$f_{int} = \frac{\mathbf{q}_{i-1} - \mathbf{q}_i}{r_{i-1}} + \frac{\mathbf{q}_{i+1} - \mathbf{q}_i}{r_i}$$

Nonostante alcuni dettagli implementativi siano differenti, il comportamento delle *elastic band* è pressoché lo stesso. Variano i tempi e l'intensità di reazione, ma le proprietà intrinseche di questo algoritmo restano le stesse. Nel capitolo 3 saranno descritte le scelte, diverse da quelle appena riportate, utilizzate nel nostro algoritmo.

Capitolo 3

Costruzione della mappa topologica e dei percorsi topologici

In questo capitolo e nel successivo verranno sviluppati i due algoritmi, *dynamic probabilistic topological map* (DPTM) e *elastic sticks*, argomento di questa tesi. La struttura utilizzata, dunque, sarà a due livelli. Nel primo livello, la DPTM eseguirà una pianificazione di percorso globale, che verrà poi passata al livello sottostante dove gli *elastic stick*, che operano *localmente*, si preoccuperanno di generare traiettorie ammissibili e reattive per l'esecuzione da parte del robot.

L'obiettivo di questo capitolo è quello di costruire una rete topologica che catturi la connettività, cioè la topologia, di W_{free}, in modo da poter generare "percorsi topologici" (cioè cammini sul grafo topologico) che saranno successivamente gli *input* per l'algoritmo successivo.

In questo capitolo e nei successivi per ***posizione*** si intende un punto in uno spazio a due o tre dimensioni, lo spazio fisico in cui si muove il robot, *non* quello delle sue possibili configurazioni. La posizione, dunque, non ha orientamento, perché *prescinde* dal robot.

Il concetto di ***configurazione***, invece, è lo stesso che utilizzavamo nei capitoli precedenti. In questo contesto potremmo dire che una configurazione è un'estensione di una posizione, quando andiamo a considerare anche l'orientamento del robot.

La mappa topologica che genererà l'algoritmo descritto in questo capitolo, sarà una rete di *posizioni*, così come il percorso topologico sarà una lista di *posizioni*. Dimentichiamo per un attimo, dunque, il robot e la sua possibile configurazione, concentrandoci unicamente sullo spazio fisico W in cui il robot è immerso, cercando di catturarne la topologia utilizzando una rete col minor numero di archi e nodi possibile. Sebbene, dunque, alcuni dei concetti usati in questo capitolo (ad esempio quello delle *roadmap*) siano stati pensati per uno spazio di configurazioni, noi ci limiteremo ad usarli nello spazio di lavoro (è chiaro che lo spazio di lavoro è un caso particolare di spazio di configurazioni).

La motivazione di questa scelta è che non è sempre agevole, in generale, costruire uno spazio delle configurazioni dato lo spazio di lavoro, soprattutto quando l'ambiente è dinamico. Sebbene nel nostro caso questo sarebbe stato possibile (perché il robot è assimilabile ad un oggetto circolare e quindi l'espansione degli ostacoli nello spazio delle configurazioni è piuttosto semplice da calcolare), abbiamo preferito sviluppare un metodo generale valido anche nei casi in cui questo non sia possibile. Possiamo occuparci delle configurazioni in maniera *locale* nel momento di generazione di traiettorie ammissibili.

Sarà comunque possibile, comunque, utilizzare sia l'algoritmo argomento di questo capitolo, che quello descritto nel successivo, in uno spazio delle configurazioni generato a partire dalla descrizione di W: sarà sufficiente considerare nulla, o molto piccola, la distanza minima consentita dagli ostacoli in tale spazio.

3.1 Introduzione

3.1.1 Concetti principali

Ciò che vogliamo ottenere è una descrizione topologicamente equivalente dello spazio di lavoro, che sia semplice da gestire e che ne rifletta gli eventuali cambiamenti topologici nel tempo. Una DPTM è essenzialmente un grafo i cui nodi e archi rappresentano una *roadmap*. Ricordiamo che in questo tipo di algoritmi, il percorso è generato connettendo prima

il punto iniziale e quello di destinazione alla rete $\mathcal{R}$ e successivamente cercando in essa un cammino che colleghi i punti di connessione.

Definiamo ***cammini semplici*** quelle traiettorie che uniscono i nodi tra di loro e p_{init} o p_{goal} alla rete. In una *roadmap*, per via del fatto che per costruire un percorso bisogna connettere il punto iniziale e quello finale alla rete, è di fondamentale importanza che *ogni* punto di W_{free} possa essere connesso, mediante un cammino semplice, ad almeno un nodo della rete. In caso contrario non saremo in grado di applicare l'algoritmo di ricerca del percorso.

In una *roadmap* questi cammini possono essere di qualsivoglia tipo, ma nel caso di una DPTM, per motivi di efficienza, consentiremo soltanto cammini semplici rettilinei.

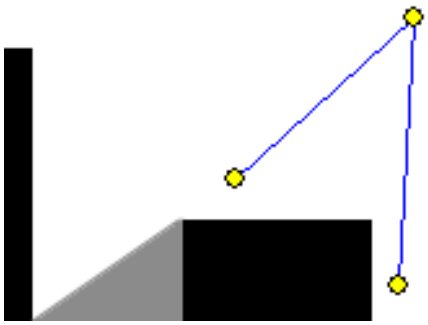


Figura 3.1: una regione di spazio non appartenente alla copertura della DPTM

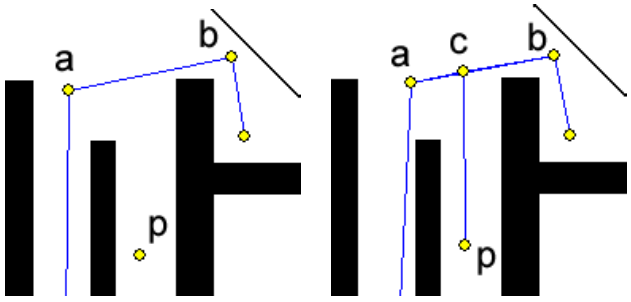
Definiamo ***copertura*** Γ di una *roadmap* quel sottoinsieme di W_{free} formato da punti che è possibile raggiungere, mediante un cammino semplice, da almeno un nodo della rete.

Affinché una *roadmap* $\mathcal{R}$ sia ***completa***, cioè sia in grado di trovare un percorso da un qualunque punto p_{init} a un qualunque punto p_{goal} qualora esso esista, devono valere le seguenti due proprietà:

1. $\mathcal{R}$ sia una *descrizione topologicamente equivalente*[1] a W_{free};
2. $\Gamma(\mathcal{R}) \equiv W_{free}$.

Per la DPTM utilizzeremo il concetto di ***visibilità***, piuttosto intuitivo. Un punto p si dice visibile da un altro punto p' se e solo se il segmento che unisce questi due punti non interseca nessun ostacolo. Definiamo dunque la funzione di visibilità $\gamma(p, p') \rightarrow \{0, 1\}$ che è uguale a 1 se e solo se p è visibile da p'. E' chiaro che la relazione di visibilità è una relazione di

[1] nel senso definito nel paragrafo 2.1



(a) (b)

Figura 3.2: il concetto di visibilita' tra nodi

equivalenza. E' chiaro, dunque, che esiste un cammino semplice tra p e p' se e solo se p è visibile da p', o, che è lo stesso, viceversa. Tale concetto è stato utilizzato anche in [PRMSimLauNis], di cui parleremo in seguito.

La copertura di una DPTM $\mathcal{D}$ è quel sottoinsieme di W_{free} definito in questo modo:

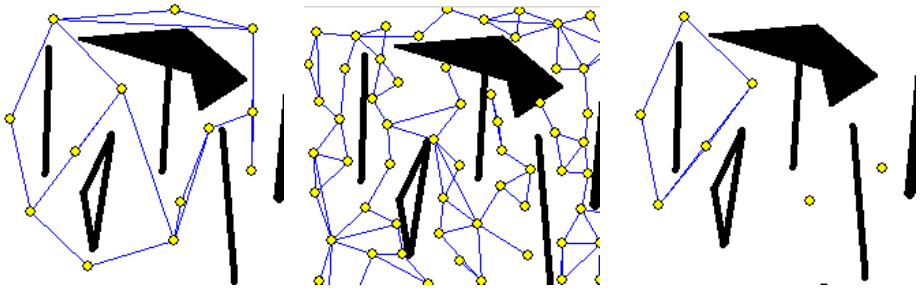
$$\Gamma(\mathcal{D}) = \{p \in W_{free} : \exists n \in \text{nodi}(\mathcal{D}), \gamma(p, n) = 1\}$$

Nella figura 3.1 possiamo vedere una piccola DPTM, formata da tre nodi, la cui copertura non è coincidente con W_{free}: la regione di spazio disegnata in grigio, infatti, non è visibile da nessun nodo della rete.

(a) (b)

Figura 3.3: un passaggio curvo e la necessità di aggiungere un nodo

Un'altra considerazione è che tutti i collegamenti devono essere fatti *tra nodi*, non è possibile, essendo la DPTM un grafo, avere degli archi che collegano un nodo a un arco. Per questo motivo, le relazioni di vicinanza proprie della topologia dello spazio W_{free}, devono riflettersi in archi nella DPTM. In figura 3.2(a) vediamo i due punti a e b che sono topologicamente vicini e sono infatti connessi da un arco ab; tuttavia il



(a) (b) (c)

Figura 3.4: densità differenti di nodi per la stessa mappa

punto p, che dovrebbe essere connesso sia con a che con b, sebbene possa vedere l'arco ab, non vede né a né b. In questo caso, per rendere la DPTM completa, è necessario aggiungere un nodo c come in figura 3.2(b).

E' inoltre chiaro che, sebbene avere solo cammini semplici rettilinei aumenta l'efficienza dell'algoritmo, questo può portare in alcuni casi particolari alla necessità di aggiungere nodi che avendo cammini semplici curvilinei non sarebbero necessari. Questa situazione è illustrata in figura 3.3, in cui la presenza di uno stretto passaggio curvo fa sì che i due nodi, sebbene siano vicini topologicamente, non si vedono e dunque non è possibile creare un arco tra di essi. Per collegarli è necessaria la presenza di un terzo nodo che faccia da ponte tra i due.

3.1.2 Obiettivi

Il primo obiettivo è ovviamente che la DPTM sia completa, come definito nel paragrafo precedente. Desideriamo inoltre che la rete sia *dinamica*, nel senso che deve modificarsi nel tempo e riflettere gli eventuali cambiamenti *topologici* dell'ambiente, o, meglio, la percezione che il robot ne ha.

Per motivi di efficienza, dobbiamo anche cercare di ridurre al minimo il numero di nodi e di archi, al limite dovremmo utilizzare solo quelli necessari a mantenere la caratteristica di equivalenza topologica con W_{free}. Questo significa che la densità di nodi non sarà costante in tutto lo spa-

zio. Nella figura 3.4(a) abbiamo un numero di nodi ragionevole (sebbene non minimo) per rappresentare la topologia di W_{free}, ma nelle figure (b) e (c) è chiaro che la densità di nodi è, rispettivamente, ridondante e insufficiente.

3.2 Gli algoritmi GNG e PRM

Entrambi gli algoritmi GNG e PRM sono volti a creare una rete che rappresenti la topologia di uno spazio di input o una *roadmap* che catturi le caratteristiche di connettività dell'ambiente. In essi possiamo individuare le due strategie di aggiunta di nodi e di archi alla rete.

3.2.1 Nodi

In GNG si parte con due nodi connessi tra di loro e altri nodi vengono aggiunti ad ogni λ iterazioni. In questo algoritmo è presente il concetto di errore locale: ogni nodo ha associata una variabile che viene incrementata di un valore che indica la distanza tra la posizione del nodo e quella dell'input ricevuto.

Chiameremo questa distanza ***influenza***, poiché essa stabilisce la massima distanza a cui un input influenza il nodo. L'algoritmo GNG, infatti, stabilisce che ad ogni ingresso venga selezionato il nodo più vicino e che quest'ultimo sia avvicinato all'input corrente. In questo modo i nodi della GNG tendono a raggiungere il centro delle loro regioni di influenza.

L'aggiunta di nodi nel GNG ha come obiettivo la minimizzazione dell'errore locale. Questi nodi nuovi, infatti, vengono inseriti tra due nodi il cui valore di errore locale è alto. Questo tende a minimizzare l'influenza di ciascun nodo e nella situazioni di equilibrio finale, l'influenza dei nodi è pressoché la stessa.

Per quanto riguarda le PRM, la condizione di inserimento di un nuovo nodo è molto più semplice: per ogni input, viene aggiunto un nodo, a prescindere dalla sua vicinanza con altri nodi o alla sua utilità.

3.2.2 Archi

Nel *growing neural gas* gli archi vengono aggiunti utilizzando una procedura che discende direttamente dalla legge di Hebb. La legge di Hebb deriva direttamente dall'osservazione dei neuroni biologici. Negli organismi animali le connessioni sinaptiche tra due neuroni vengono rafforzate quando essi si attivano contemporaneamente, cioè se esiste una correlazione tra i due eventi che attivano i due neuroni, questa correlazione viene "memorizzata" nel collegamento sinaptico. Nel GNG viene utilizzato un principio simile: ad ogni input vengono selezionati dalla rete i due nodi più vicini ad esso e tra questi viene creato un arco.

Nelle *probabilistic roadmap* per ogni nodo aggiunto, esso viene collegato a *tutti* i nodi entro una certa distanza che non appartengono alla stessa componente connessa del nodo appena aggiunto. Questo evita di avere archi inutili e di avere cicli nel grafo, ma rende i percorsi molto più complessi di quanto sarebbe necessario.

3.3 L'algoritmo *dynamic probabilistic topological map*

3.3.1 Nodi e archi

Abbiamo bisogno, dato un input generato casualmente, di alcuni criteri per inserire nodi e archi nel grafo. Per quanto riguarda l'aggiunta dei nodi, sia GNG che PRM non tengono in considerazione la ridondanza topologica di un nodo. In una situazione limite, quale ad esempio una stanza quadrata priva di ostacoli, entrambi gli algoritmi genererebbero un gran numero di nodi, tutti ridondanti per quanto riguarda l'acquisizione della banale topologia dell'ambiente. Questo discorso si applica in generale a qualunque mappa formata da una sola regione convessa. D'altra parte non possiamo permetterci di perdere input importanti e limitarci ad aggiungere un nodo soltanto ogni λ iterazioni.

Il primo criterio di aggiunta di un nodo è quello relativo al fatto che tutto lo spazio W_{free} deve essere visibile da almeno un nodo della

rete. Questo implica che qualora la posizione di input non sia visibile da nessun nodo della rete già esistente, un nuovo nodo viene aggiunto in questa posizione. In questo modo, probabilisticamente parlando (cioè con $t \to \infty$), avremo almeno un nodo rappresentante di (la cui influenza raggiunge) ogni punto dello spazio libero di lavoro. In questo criterio rientra anche l'aggiunta del primo nodo, che verrà posto nella posizione del primo input dato all'algoritmo.

In caso contrario, cioè se la posizione di input vede almeno un nodo della rete, viene selezionato il nodo *visibile* più vicino e viene spostato verso l'input seguendo la formula di aggiornamento propria del GNG (e, a dire la verità, di quasi tutte le reti neurali non supervisionate):

$$p_{new} = p_{old} + \alpha(n - p_{old})$$

in cui, naturalmente, p_{old} e p_{new} sono la posizione corrente e quella nuova del nodo, n è l'input ricevuto e α è una costante, detta *coefficiente di apprendimento*.

In questo modo i nodi tenderanno a raggiungere il centro delle regioni di influenza, incrementando, come effetto collaterale, la loro *clearance* (figura 3.5). Anche questo fatto è importante, in quanto permette ai nodi di vedere più probabilmente altre posizioni ([PRMWilAmaSti]).

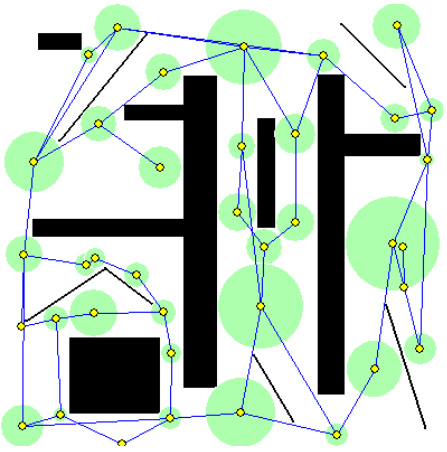
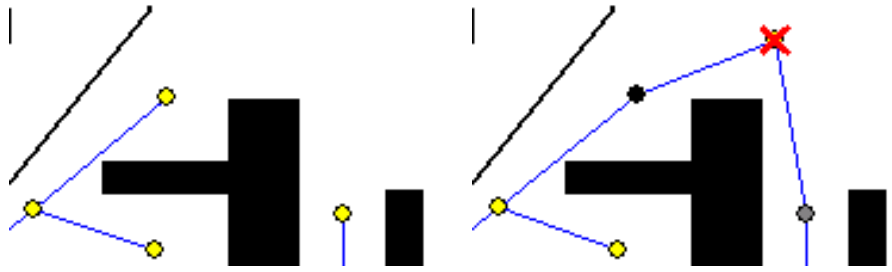


Figura 3.5: la *clearance* dei nodi della rete

Per quanto riguarda l'aggiunta di archi, anche in questo caso sfruttiamo inizialmente il criterio del GNG: ad ogni input vengono selezionati i nodi *visibili* più vicini e collegati da un arco (legge di Hebb), se questo è possibile. E' chiaro, infatti, che se una posizione seleziona due nodi come più vicini, essi non è detto che si vedano l'un l'altro e che quindi possa esistere un arco che li connette (gli archi devono giacere completamente in W_{free}).



(a) (b)

Figura 3.6: l'aggiunta di un nodo come "ponte" tra due che non si vedono

3.3.2 Ulteriori criteri di aggiunta di nodi e archi

L'aggiunta di un arco tra due nodi che si possano vedere sembrerebbe essere incoerente. E' apparentemente impossibile, infatti, che esistano due nodi visibili l'uno dall'altro, poiché l'algoritmo finora descritto impone che nel caso una posizione di input possa vedere un qualunque nodo, non venga aggiunto nessun nuovo nodo in tale posizione.

In realtà, dobbiamo considerare che la rete è dinamica, i nodi si spostano dalle loro posizioni di partenza ed è dunque possibile che un nodo diventi visibile da un altro, anche se questo inizialmente non era vero.

Un criterio utile per migliorare la connettività della rete è quello di inserire un nodo in una posizione in cui i due nodi più vicini non si vedono tra di loro. In questo caso tale nodo farà da "ponte" tra i due e quindi tra le due regioni rappresentate da essi. Una situazione in cui viene applicato questo criterio è mostrata in figura 3.6.

Tale criterio è simile all'utilizzo di nodi di connessione tra i nodi denominati "guardie" in [PRMSimLauNis], tuttavia nella DPTM non esiste nessuna precisa distinzione tra nodi, sia per la sua natura dinamica, sia perché non vogliamo precluderci la possibilità di connettere due nodi qualunque, qualora sia necessario per avere una descrizione topologica più completa.

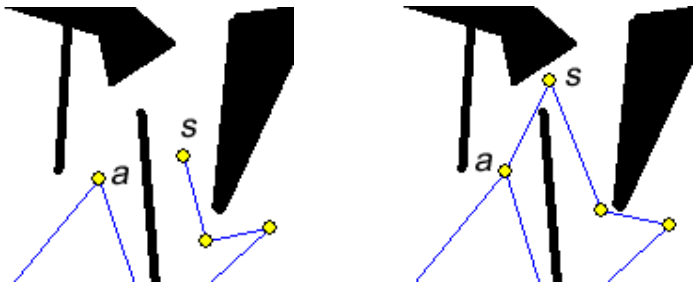


Figura 3.7: esplorazione di un nodo "scout"

3.3.3 Mantenimento della connettività raggiunta

E' fondamentale non perdere informazioni sulla topologia dell'ambiente analizzato dalla DPTM. La natura dinamica della DPTM, infatti, rischierebbe di farci spostare un nodo in una posizione in cui potrebbe non vedere più uno dei nodi ad esso collegato, che verrebbe quindi, coerentemente, eliminato.

Prima di spostare un nodo è dunque opportuno controllare se nella nuova posizione esso conserverebbe tutti gli archi a esso collegati, solo in tal caso procederemo all'effettivo spostamento del nodo.

3.3.4 I nodi "scout"

Un tipo particolare di nodi è quello da cui parte un solo arco. E' probabile che un tale nodo si trovi in regioni ancora inesplorate (cioè prive di nodi) o in prossimità di un passaggio stretto e che sia opportuno che vi si avvicini per poter vedere altri nodi dall'altra parte del passaggio. Per questo motivo un tale nodo deve *allontanarsi* dalla rete. Il modo più veloce per determinare se si sta allontanando dalla rete, sebbene esso non sia preciso, è quello di considerare il suo unico arco: se spostando il nodo il suo arco diventa più lungo, è probabile che ci stiamo allontanando dalla rete.

Per questo tipo di nodi, dunque, definiamo un ulteriore criterio per impedire di spostarlo (in aggiunta a quello descritto in precedenza): un

nodo scout non può essere spostato se il suo arco, nella nuova posizione, è più corto che nella posizione attuale. Gli aggettivi "più lungo" e "più corto" sono ovviamente relativi alla metrica dello spazio.

Nella figura 3.7(a) vediamo un nodo scout (etichettato con s) che viene spinto in avanti verso la posizione che ha nella figura 3.7(b): da questa nuova posizione ha la possibilità di collegarsi al nodo etichettato con a, che prima non era nel campo di visibilità.

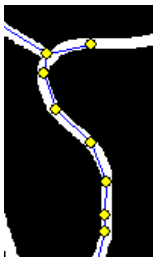


Figura 3.8: i nodi "scout" formano delle catene in passaggi curvi

Questa caratteristica è in qualche modo simile con la fase di espansione delle PRM classiche, in cui vengono generate delle configurazioni casuali vicine (seguendo alcuni "rimbalzi casuali") ad un nodo m che viene selezionato casualmente, ma con probabilità inversamente proporzionale al numero di archi.

Un'altra situazione in cui i nodi scout sono di fondamentale importanza è quando essi si trovano all'interno di passaggi stretti. In tali casi essi, per la loro caratteristica di spingersi nella direzione opposta al loro arco, percorrono il passaggio stretto finendo al capo opposto, rendendo dunque possibile la connessione tra i nodi al di qua del passaggio con quelli al di là. Esiste tuttavia un caso in cui un nodo scout non è in grado di comportarsi in questo modo. Se il passaggi stretto è curvo, infatti, non potendo esso perdere il suo unico arco, arriverà presto il momento in cui non sarà più in grado di esplorare il passaggio stretto. Per ovviare a questo inconveniente, stabiliamo di aggiungere un nuovo nodo se valgono tutte e tre le seguenti condizioni:

1. il nodo più vicino alla posizione di input è un nodo "scout";
2. tale nodo non può essere spostato perché altrimenti perderebbe il proprio arco;
3. dalla nuova posizione non è possibile vedere il nodo collegato al nodo "scout".

L'ultima condizione è necessaria per assicurarci che la posizione di input sia effettivamente in una zona inesplorata. In questo modo si vengono

dunque a formare, all'interno dei passaggi stretti e curvi, delle catene di nodi che ne catturano la connettività (vedi figura 3.8).

3.3.5 Eliminazione delle ridondanze

Nell'ottica di ridurre al minimo il numero di nodi e di archi presenti nella DPTM, dobbiamo considerare la situazione in cui la struttura che in un momento potrebbe essere accettabile, diventa eccessivamente ridondante, ad esempio perché l'ambiente è cambiato. In tal caso è opportuno semplificare la rete, perché siamo in presenza di nodi ridondanti.

Un semplice test per individuare se due nodi a e b sono ridondanti è quello di controllare se ogni nodo a' collegato ad a è visibile dal nodo b e viceversa. In tal caso possiamo procedere alla fusione dei due nodi, aggiungendo tutti gli archi necessari per avere la stessa connettività precedente. La posizione del nodo risultato della fusione viene intuitivamente calcolata come media tra le posizioni di a e di b (che vengono rimossi dalla rete).

3.3.6 Il *gruyere method*

Gli stessi inventori dell'algoritmo delle PRM notano che uno dei suoi problemi principali è la considerazione dei "passaggi stretti", quelle zone di W_{free}, cioè, che hanno un piccolo volume ma sono indispensabili per avere un quadro completo della connettività dello spazio di lavoro. Il problema è dovuto principalmente alla generazione di posizioni casuali, poiché tale generazione ha una densità di probabilità costante in tutto W_{free} e questo risulta in una bassa probabilità di generazione di posizioni all'interno di passaggi stretti e una alta probabilità in zone più estese, proprio per via del loro volume. Per catturare la connettività di W_{free} sarebbe necessaria la situazione opposta.

Il metodo esposto in [PRMBridge], denominato ***bridge test***, sostituisce alla generazione di posizioni casuali in W_{free}, utilizzato nell'algoritmo delle PRM canonico, la costruzione di "ponti" (da cui il nome del metodo), cioè segmenti che hanno gli estremi in W_{obst} (i pilastri del ponte) e il pun-

to mediano "sospeso" sullo spazio W_{free}. Le posizioni fornite all'algoritmo PRM, dunque, sono proprio i punti mediani di questi segmenti.

Dal punto di vista dell'implementazione vengono dunque generati in maniera casuale coppie di punti in W_{obst} e considerato il loro punto mediano, che viene fornito in ingresso alla PRM se e solo se esso giace in W_{free}. Dando una lunghezza massima possibile al "ponte" si aumenta la probabilità di fornire all'algoritmo di PRM delle posizioni all'interno di passaggi stretti e la si diminuisce nelle zone aperte (è molto più semplice costruire ponti corti nei passaggi stretti che negli spazi aperti).

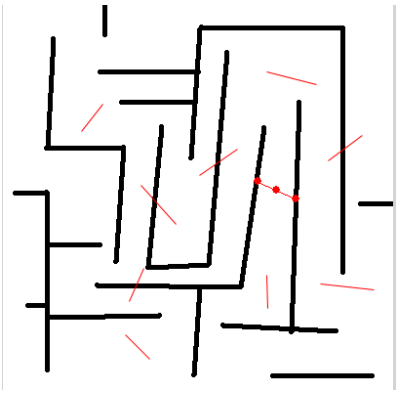


Figura 3.9: il *bridge test* è poco adatto in mappe con un volume di ostacoli piccolo

Tuttavia il *bridge test* presentato nella pubblicazione sopra citata ha delle lacune se esso viene utilizzato in mappe con molti ostacoli con volume ristretto (ad esempio molto lunghi e stretti). Tale metodo, infatti, ipotizza la presenza di una vasta area coperta da ostacoli intorno ai passaggi stretti, fatto che non sempre è vero. Alcune mappe da noi utilizzate, ad esempio, erano in forma vettoriale; in questa situazione il *bridge test* scarta ovviamente un'alta percentuale di *input*, poiché si pone nuovamente il problema di generare punti in uno spazio di volume ristretto, questa volta quello degli ostacoli (ma il problema sarebbe analogo). Nel caso della mappa "a labirinto" della figura 3.9, ad esempio, il *bridge test* è inadeguato, nel senso che, nonostante riesca a ottenere il risultato desiderato, gran parte degli *input* viene scartata, causando un peggioramento delle prestazioni.

Si può tuttavia estendere tale metodo nel seguente modo. Generiamo coppie di punti, questa volta in tutto W, dopodiché, invece di limitarci a considerare gli estremi e il punto mediano, calcoliamo tutte le intersezioni del segmento "ponte" con gli ostacoli presenti nell'ambiente. Gli input indirizzati alla rete saranno tutti i punti mediani tra due intersezioni con gli ostacoli. In questo modo otteniamo una serie di *bridge test* in linea, ma

senza la necessità di dover avere gli estremi in zone coperte dagli ostacoli. Per aumentare la probabilità di generare posizioni importanti, gli estremi del nostro "*bridge test* multiplo" vengono scelti sempre appartenenti alla frontiera di W.

In questo modo, al contrario del *bridge test* canonico, non verrà scartata nessuna configurazione di ingresso: *ogni* applicazione di questo metodo porta alla generazione di *almeno* una configurazione che, come vediamo nel seguito, si trova in una posizione che accresce la velocità in cui la rete raggiunge la connettività completa.

Per dare un nome a questo metodo, utilizziamo una metafora altrettanto colorita rispetto alla "prova del ponte" descritta nella pubblicazione citata all'inizio di questo paragrafo: possiamo considerare lo spazio W come una fetta di groviera, il famoso formaggio coi buchi; per riuscire a vedere i buchi presenti nel formaggio (i nostri "passaggi stretti"), il modo migliore è tagliarlo da parte a parte, in maniera molto simile a quello che fanno i segmenti che attraversano da frontiera a frontiera lo spazio W. Per questo chiamiamo questo procedimento ***gruyere method***.

L'aumento di efficienza dell'algoritmo finale, utilizzando metodi di generazioni di configurazioni come il *bridge test* o il *gruyere method* viene confermato anche in [PRMWilAmaSti], ove si sviluppa un algoritmo per generare configurazioni lungo l'asse mediale (detto anche diagramma di Voronoi generalizzato), osservando un forte aumento di prestazioni in termini di tempo necessario a raggiungere una connettività (o descrizione topologica) completa. Le configurazioni generate dal *bridge test* o dal *gruyere method*, infatti, si trovano, per costruzione, proprio su tale asse mediale.

Uno dei problemi, riscontrati anche nel *bridge test* tradizionale, è che si rischia di avere troppi punti nei passaggi stretti e troppo pochi o nessuno nelle zone "aperte". La stessa pubblicazione a cui stiamo facendo riferimento offre una semplice soluzione: alternare input generati nella maniera classica (ossia casuale con distribuzione costante) a quelli generati con il metodo appena descritto.

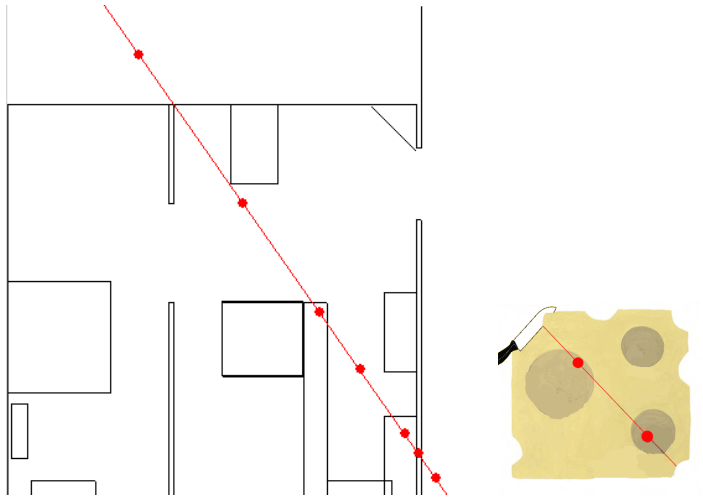


Figura 3.10: un esempio di applicazione del *gruyere test*

3.4 Analisi dell'algoritmo DPTM

3.4.1 Descrizione e pseudo-codice

Presentiamo ora l'algoritmo DPTM completo, che possiamo vedere in figura in pseudo-codice. Nonostante i numerosi criteri, esso risulta essere abbastanza semplice e, dai risultati ottenuti in simulazione, molto veloce.

Per quanto riguarda le funzioni utilizzate nell'algoritmo in figura, le loro definizioni sono le seguenti:

`generatore input()` Questa funzione sceglie una posizione casuale in W_{free}, utilizzando alternativamente il *gruyere method* e una generazione casuale con densità di probabilità uniforme.

`pos(n)` Dato un nodo n, questa funzione ne restituisce la posizione.

`scout_riduce_arco(n,pos)` E' una funzione binaria che restituisce `true` se e solo se il nodo n è uno scout (cioè esiste un solo arco che parte da esso) e nella posizione `pos` tale arco avrebbe una lunghezza minore rispetto all'attuale.

`vista(pos1,pos2)` Restituisce `true` se e solo se il segmento che unisce le due posizioni è contenuto completamente in W_{free}.

Algoritmo 3 L'algoritmo della *dynamic probabilistic topological map*

$N \leftarrow \emptyset$, $E \leftarrow \emptyset$
loop
  $\xi \leftarrow$ generatore_input()
  $n =$ vicino1(N, pos)
  $m =$ vicino2(N, pos)
  if $\neg\exists n$ **then**
    aggiungi un nodo nella posizione ξ
  else
    new_pos $=$ pos$(n) + \eta \cdot (\xi -$ pos$(n))$
    if aggiornamento_ammesso$(n,$ new_pos$)$ **then**
      pos$(n) =$ new_pos
    else if $\neg$ scout_riduce_arco$(n) \wedge \neg\exists r \in$ collegati(n) : vista$(\text{pos}(n), \xi)$ **then**
      caggiungi un nodo nella posizione ξ e collegalo con n
    end if
    if $\exists m$ **then**
      if vista$(\text{pos}(n), \text{pos}(m))$ **then**
        crea un arco tra n e m
      else if $\neg\exists q \in$ collegati$(n) \cap$ collegati(m) : ridondanti(ξ, q) **then**
        inserisci un nodo nella posizione ξ e collegalo con n e con m
      end if
    end if
    if ridondanti(n, m) **then**
      poni un nuovo nodo nella posizione media tra n e m, collegato a tutti i nodi in collegati$(n) \cup$ collegati(m) ed elimina n ed m
    end if
  end if
end loop

`collegati(n)` Restituisce l'insieme dei nodi collegati al nodo n mediante un arco.

`aggiornamento_ammesso(n,pos)` E' una funzione binaria che restituisce `true` se e solo se è possibile portare il nodo n alla posizione *pos*, cioè se nessun arco viene perso in *pos* e `scout_riduce_arco(n,pos)` è falsa.

`ridondanti(n,m)` Restituisce `vero` se e solo se da n è possibile vedere tutti i nodi collegati a m e viceversa.

3.4.2 Diagramma di Voronoi e triangolazione di Delaunay

Procederemo ora ad uno studio dell'algoritmo presentato, per cercare degli ulteriori miglioramenti, soprattutto per quanto riguarda la semplificazione della rete.

Nel paragrafo relativo alle *roadmap*, nel capitolo 1, abbiamo parlato di un particolare tipo di diagramma di Voronoi, che prendeva come *generatori* gli ostacoli (tutti i punti degli ostacoli). Avevamo definito, infatti, un diagramma di Voronoi, come quel luogo di punti equidistanti da due o più ostacoli.

In realtà la definizione di ***diagramma di Voronoi*** è più generale. Dato uno spazio E e un insieme di n punti G detti generatori, un diagramma di Voronoi è una partizione dello spazio E in n poligoni convessi, detti ***regioni di Dirichlet***, tali che ogni poligono contiene uno e un solo generatore e ogni punto in un poligono è più vicino al suo generatore che a qualunque altro generatore. E' chiaro che considerando gli ostacoli come generatori, la *roadmap* di cui abbiamo parlato nel capitolo 1 è formata dai confini delle regioni di Dirichlet, che, appunto, come ovvia conseguenza hanno quella di essere equidistanti dai due generatori più vicini.

Data una DPTM e considerando un diagramma di Voronoi dello spazio W_{free} (quindi escludendo gli ostacoli), usando i nodi della rete come generatori, invece che gli ostacoli, otteniamo una partizione dello spazio

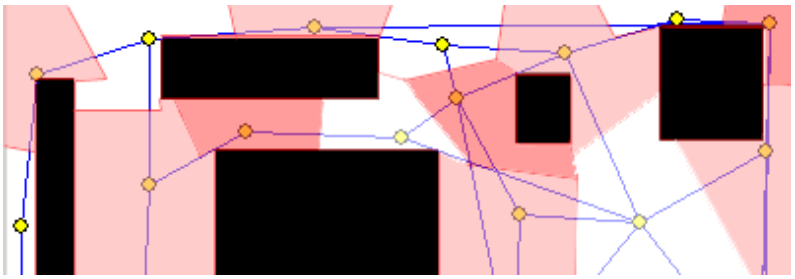


Figura 3.11: diagramma di Voronoi e regioni di Dirichlet

libero di lavoro in celle di dimensioni variabili. Poiché saremmo portati a pensare che esiste una relazione di vicinanza tra le celle, indotta dai collegamenti tra i nodi che le generano, potremmo considerare questo insieme di celle come una *decomposizione in celle esatta* di W_{free}. Da questa decomposizione possiamo più facilmente osservare la presenza di un gran numero di celle in regioni dello spazio particolarmente complesse e l'esistenza, al limite, di una sola cella in regioni libere convesse.

In realtà la relazione di vicinanza indotta dagli archi della DPTM non è adatta ad un algoritmo di decomposizione in celle. Si può notare dalla figura 3.11, che alcuni archi passano attraverso più celle, rendendo quindi *adiacenti*, nel *grafo delle adiacenze*, celle che in realtà non lo sono (sebbene esse siano comunque raggiungibili tra di loro utilizzando un cammino semplice).

Una ***triangolazione di Delaunay*** (vedi ad esempio in figura 3.12) è una struttura duale al diagramma di Voronoi. Dato uno spazio E e un insieme di punti G in esso contenuti, chiamiamo triangolazione di Delaunay quella triangolazione (che è unica) tale che nessun punto di G è contenuto nella circonferenza circoscritta a qualunque triangolo. Come è facile vedere, questo tipo di triangolazione evita i problemi riscontrati nella figura precedente. Se dunque le connessioni tra i nodi della DPTM fossero una triangolazione di Delaunay, avremo una rete ottima, cioè una rete contenente tutti e solo gli archi necessari per descrivere la vicinanza topologica tra nodi. Infatti, l'arco che si vede in alto alla figura 3.11, che attraversa ben cinque celle, non aggiunge dati alla descrizione della topologia dell'ambiente, in quanto l'informazione che esso rappresenta è

duplicata dagli archi che uniscono i nodi delle celle attraverso cui passa.

Purtroppo una triangolazione di Delaunay non può essere definita per spazi non convessi, quindi tantomeno in spazi (W_{free}) contenenti "buchi" (gli ostacoli). Per ottenere qualcosa di simile alla triangolazione di Delaunay in casi come questo, sono necessari altri algoritmi di complessità eccessiva per il nostro genere di applicazioni.

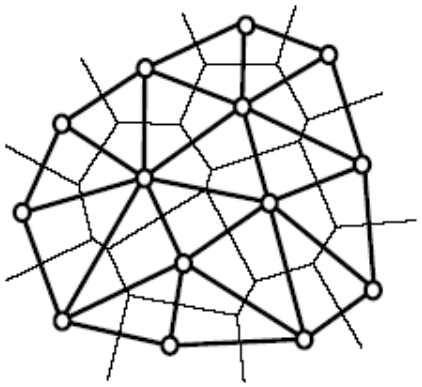


Figura 3.12: una triangolazione di Delaunay (segmenti a tratto spesso)

Possiamo tuttavia notare due dei problemi che impediscono alla nostra rete di definire una decomposizione in celle (esatta) di W_{free}: il primo è la presenza di archi che si intersecano tra di loro, il secondo di archi che attraversano più di due regioni di Dirichlet. Il primo problema è facilmente risolvibile aggiungendo un nodo nel punto di incrocio dei due archi e ricostruendo in maniera opportuna i collegamenti (i due archi iniziali devono essere sostituiti da quattro archi che collegano il nodo appena aggiunto ai quattro i nodi che formavano l'incrocio). Da notare che questo evento è molto comune in mappe bidimensionali, ma quasi impossibile in mappe di dimensioni superiori (la rete è sempre formata da cammini semplici monodimensionali).

Per quanto riguarda il secondo, sembrerebbe sufficiente verificare che il punto mediano dell'arco appartenga alla regione di Dirichlet di uno dei due estremi, se così non è, allora l'arco attraversa più di due regioni (il punto mediano dell'arco, infatti, dovrebbe appartenere alla frontiera delle due regioni dei due nodi che formano l'arco). In tal caso potremmo eliminare l'arco, in quanto non dà informazioni aggiuntive sulla topologia dell'ambiente. In realtà questo criterio porta ad eliminare anche archi utili per la topologia di W_{free}. Consideriamo un semplice esempio in cui vi sono tre nodi a, b e c. Gli archi attualmente esistenti sono ab e bc, inoltre a *non vede* c e quindi non è possibile che si formi un arco tra di

loro. Se utilizziamo il criterio precedentemente descritto è possibile che il punto mediano dell'arco ab sia nella regione di influenza del nodo c e quindi saremo portati a eliminare l'arco ab in quanto esso passa attraverso la regione di Dirichlet del nodo c. Tuttavia quest'arco è fondamentale per indicare che la regione di a e la regione di b sono connesse tra di loro attraverso la regione di c. Inoltre, sebbene a e c non siano collegati da un arco, le loro regioni di Dirichlet sono adiacenti. La soluzione potrebbe essere quella di aggiungere un nodo nel punto centrale dell'arco ab. Tuttavia non possiamo assicurare la presenza di un nodo "ponte" di questo tipo in ogni momento, essendo la rete dinamica.

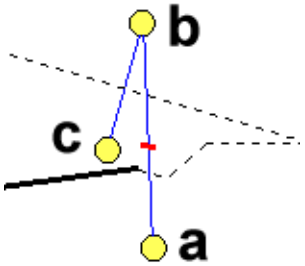


Figura 3.13: un caso critico per la decomposizione in celle della DPTM

Per un criterio più esatto di eliminazione di archi dovremmo prima controllare che *tutti* i nodi delle celle attraversate dall'arco siano già collegati tra loro da un arco. Questa verifica è piuttosto onerosa dal punto di vista computazionale e abbiamo deciso dunque di non implementarla.

L'aggiunta di un nodo all'intersezione di due archi, invece, è stata implementata e, essendo l'incrocio tra due archi un evento raro, viene effettuata una ricerca all'interno del grafo di questa eventualità con una frequenza temporale molto bassa. Bisogna ricordare, infatti, che a prescindere da una maggiore semplicità della rete (è stato osservato spesso che la rete si semplifica dopo l'inserimento di un nodo all'incrocio di due archi, grazie al criterio di semplificazione che unisce due nodi ridondanti), ciò di cui abbiamo discusso in questo paragrafo non è fondamentale per l'utilizzo dell'algoritmo. Avere archi ridondanti, infatti, può avere delle piccolissime ripercussioni sugli algoritmi di ricerca di cammini, ma non ne ha per quanto riguarda la completezza.

3.4.3 Confronto con gli algoritmi GNG e PRM

Rispetto ai due algoritmi di partenza (GNG e PRM) si può dire che la DPTM somiglia molto più al *growing neural gas* che alle *probabilistic roadmap*. Rispetto a queste ultime ha delle importanti differenze:

1. La DPTM è un algoritmo *dinamico*, esso cioè può essere eseguito senza soluzione di continuità. La rete è in grado di aggiornarsi, modificando la propria complessità e struttura in relazione alla situazione attuale dell'ambiente. Una PRM, invece, una volta costruita non definisce un metodo di aggiornamento. Sebbene esistano delle varianti delle PRM che vengono costruite *per ogni singola richiesta*, esse non sono simili alla DPTM, perché le informazioni che contengono vengono perse all'interrogazione successiva.

2. La PRM richiede come parametro la durata di esecuzione. L'utente può valutarlo osservando la rete dopo un certo periodo e decidendo se la connettività è stata raggiunta o meno (rendendo dunque inutile l'automazione del processo) oppure determinarlo mediante una serie di esperimenti, volti a conoscere il tempo necessario per ottenere la rete voluta in un certo numero di casi; dunque è possibile che un tempo *ragionevole* per un certo ambiente non lo sia per un altro, causando in quest'ultimo caso l'interruzione della costruzione della PRM troppo presto. La DPTM, invece, non necessita di questo tipo di intervento, né che il tempo di esecuzione sia fissato come parametro: una volta raggiunta la connettività dell'ambiente, essa rimane stabile finché non sopraggiunge quache cambiamento topologico.

3. Come verrà illustrato nel capitolo relativo alle sperimentazioni, il numero di nodi generato da una DPTM è nell'ordine del 5% rispetto ad una PRM che ottiene la stessa descrizione topologia; anche questo fatto è fondamentale, perché implica che tutte le ricerche (di percorsi topologici, ad esempio) eseguite nella rete hanno un tempo di esecuzione molto inferiore; dobbiamo tener presente che in alcuni casi, quando viene ricercato un nuovo percorso topolo-

gico perché si hanno archi non sicuri o improvvisamente eliminati per via di variazioni nell'ambiente, è possibile che si abbiano anche molte richieste consecutive di un percorso alternativo da ricercare nel grafo. Il vantaggio di avere un così basso numero di nodi e archi è importante anche e soprattutto per la dinamicità della rete: aggiornare un numero di nodi molto maggiore può avere gravi ripercussioni sull'efficienza dell'algoritmo.

4. L'algoritmo delle PRM canonico evita la costruzione di cicli nel grafo risultante, connettendo due nodi soltanto se essi non si trovano nella stessa componente connessa; questo metodo, infatti, non mira a raggiungere una descrizione topologicamente equivalente dell'ambiente, bensì a ottenere la sua connettività. Per una PRM, al contrario del DPTM, non ha importanza che una configurazione possa essere raggiunta mediante un percorso topologicamente migliore (più breve), ma che, se q_{init} e q_{goal} appartengono alla stessa componente connessa di C_{free}, possa essere trovato un qualunque percorso che le unisca. La PRM, infatti, mira a rappresentare la *connettività* dell'ambiente, non la *topologia*.

E' doveroso osservare, tuttavia, che la probabilità di connettere una nuova configurazione ad altre già presenti nella rete è proporzionale al numero di queste ultime. Avendo la DPTM un numero drasticamente inferiore di nodi, ci si aspetterebbero delle prestazioni altrettanto minori. Dagli esperimenti, invece, risultano, per quanto riguarda il tempo di raggiungimento dell'obiettivo, dei tempi superiori, nei casi peggiori, soltanto del 5-10% e situazioni in cui addirittura le DPTM sono più veloci delle PRM. Questo è facilmente spiegabile considerando che l'algoritmo DPTM tende a ridurre la propria complessità in maniera *intelligente*, evitando di inserire (o eliminando) solo quei nodi che non sono utili alla sua completezza.

Rispetto al *growing neural gas*, invece, mantiene gran parte delle caratteristiche dell'algoritmo, ma con delle importanti differenze:

1. Il GNG *può* non avere condizioni di terminazione e continuare a funzionare indefinitamente senza che questo risulti dannoso in termini di prestazioni (cioè raggiunge una condizione di equilibrio). La

DPTM è stata progettata appositamente per *non avere* condizioni di terminazione.

2. Mentre il GNG inserisce nodi solo quando il numero di iterazioni è un multiplo di un certo parametro λ, la DPTM aggiunge nodi *ogni volta che e solo quando questi siano considerati necessari.*

3. Nel GNG esiste un meccanismo di "invecchiamento degli archi", volto a eliminare quegli archi che non vengono selezionati da molto tempo. In questo modo si tende ad ottenere una triangolazione di Delaunay, come abbiamo visto nel paragrafo precedente. Per quanto riguarda la DPTM, invece, essa evita di eliminare archi, per non rischiare di perdere informazioni importanti riguardo alla topologia dell'ambiente.

4. La densità di nodi di un GNG, data una $P(\xi)$ costante, è costante nello spazio di *input*. Nella DPTM, invece, essa è variabile e funzione della complessità topologica locale (o, meglio, della necessità di avere più o meno nodi e archi per poterla rappresentare).

3.5 La generazione del percorso topologico

Abbiamo detto che una volta costruita la mappa topologica, cercare il percorso topologico diventa un semplice problema di ricerca di cammini in un grafo. Possiamo dunque utilizzare algoritmi come quello di *Dijkstra* o *A**. Un fattore importante da considerare è la connessione delle posizioni p_{init} e p_{goal} ai nodi della rete. Il metodo più semplice, quello considerato in gran parte degli algoritmi di *roadmap*, è di unire tali posizioni ai nodi più vicini. Tale scelta, sebbene molto semplice, presenta dei problemi, soprattutto nel caso delle DPTM, in cui il nodo più vicino può trovarsi in una posizione decisamente scomoda rispetto al percorso che verrà generato successivamente. E' infatti possibile che il percorso generato dall'algoritmo di ricerca del cammino sul grafo costringa il robot a tornare indietro rispetto all'effettiva direzione che segue il percorso.

3.5.1 I nodi temporanei

Per ovviare a questo inconveniente, prima di cercare il percorso con *Dijkstra* o *A**, vengono inseriti nella rete due nodi "temporanei", uno in p_{init} e uno in p_{goal}. Questo inserimento non segue nessuno dei criteri utilizzati per le posizioni presentate alla rete. Non viene fatto dunque nessun controllo sulla visibilità per evitare di inserire tali nodi. I nodi temporanei vengono *sempre* inseriti, a prescindere dalla visibilità con altri nodi della rete.

Inoltre, per quanto riguarda gli archi, questi nodi vengono connessi a *tutti* i nodi della rete visibili da tali posizioni. In questo modo l'algoritmo di ricerca su grafo avrà la possibilità di selezionare effettivamente il migliore nodo successivo a p_{init} e precedente a p_{goal}. Questo sistema evita anche di fare errori *topologici* del percorso, costringendo il robot a fare inutili giri intorno ad ostacoli che potrebbe invece evitare completamente (le PRM, sebbene solo in rari casi, presentano questo problema).

I nodi temporanei vengono eliminati dalla rete non appena è stato generato il percorso topologico e il loro tempo di vita è dunque limitato a tale generazione (i nodi temporanei non vengono considerati concettualmente come appartenenti alla DPTM, perché esistono all'unico scopo di calcolare un percorso).

3.5.2 Archi non sicuri

Per motivi che saranno più chiari in seguito, abbiamo bisogno di una procedura che ci consenta di cercare più di un percorso possibile all'interno del grafo. Questo è dovuto al fatto che, ad esempio, può esistere un arco tra due nodi perché vi è visibilità tra essi, ma il robot non può percorrere tale arco perché esso attraversa un passaggio stretto.

Ricordiamo che questa è una eventualità che nasce dal fatto che stiamo considerando lo spazio di lavoro e non lo spazio delle configurazioni. In caso pianificassimo in C_{free}, infatti, questa eventualità non accadrebbe, in quanto i passaggi non percorribili da parte del robot non esisterebbero affatto nello spazio delle configurazioni. Sebbene sia possibile, utilizzando questo algoritmo, pianificare in C_{free}, preferiamo procedere

in questo modo per ottenere un metodo più generale, che valga anche in situazioni in cui non abbiamo possibilità, o non è conveniente, costruire lo spazio delle configurazioni.

Dobbiamo avere dunque la possibilità di ottenere "un altro" percorso. La strategia che abbiamo adottato è quella di poter individuare quegli archi che provocano l'esclusione, da parte dell'algoritmo a livello più basso, della traiettoria. Questi archi vengono momentaneamente (o indefinitamente) considerati "non sicuri" (*unsafe*). Quando viene richiesto un altro percorso topologico, questi archi non vengono considerati (o, meglio, la loro lunghezza viene considerata infinita), ottenendo così un percorso diverso dal precedente. Si può procedere in questo modo finché non viene trovato un percorso "sicuro" o non sia più possibile generarne uno. In quest'ultimo caso si può dire che non esiste un percorso percorribile tra p_{init} e p_{goal}.

Capitolo 4

Dal percorso topologico alla traiettoria reale

Abbiamo detto che il percorso topologico deve essere modificato prima di poter essere seguito facilmente dal robot. Infatti, per costruzione, un percorso topologico è formato da alcuni archi della rete topologica e la successione di segmenti che ne risulta può avere dei punti di discontinuità ed essere molto più lungo del necessario. E' opportuno, infatti, che il robot passi ad una certa distanza dagli ostacoli, per evitare collisioni, ma è altresì bene che riduca la distanza percorsa al minimo. Il nostro scopo è trovare un punto di incontro tra queste due esigenze contrastanti.

4.1 Estensione dell'algoritmo delle *elastic band*

L'algoritmo delle *elastic band* riceve un percorso generato da un pianificatore globale e fornisce una traiettoria smussata che sia lontana dagli ostacoli abbastanza da non collidere con essi ma non eccessivamente. E' ovvio, a questo punto, che inizializzeremo l'*elastic band* proprio col percorso topologico ottenuto dalla DPTM (figura 4.1).

4.1.1 Modifiche all'algoritmo

4.1.1.1 Stabilità

Uno dei primi problemi riscontrati con l'utilizzo delle *elastic band* è quello relativo alla loro instabilità in passaggi stretti. La causa di questo è dovuta all'aggiornamento che viene effettuato sulle singole bolle. Data la forza, modifichiamo la posizione della bolla utilizzando la formula $\mathbf{b}_{new} = \mathbf{b}_{old} + \alpha \mathbf{f}_{tot}$. Se α è un valore troppo grande, o se la bolla si trova in un passaggio stretto, la formula di aggiornamento rischia di farla finire *oltre* l'ostacolo (vedi figura 4.2), con la disastrosa conseguenza di avere l'*elastic band* che attraversa ben due volte l'ostacolo. Un altro evento che può capitare (figura 4.3) è che la bolla oscilli indefinitamente e quindi renda il percorso instabile.

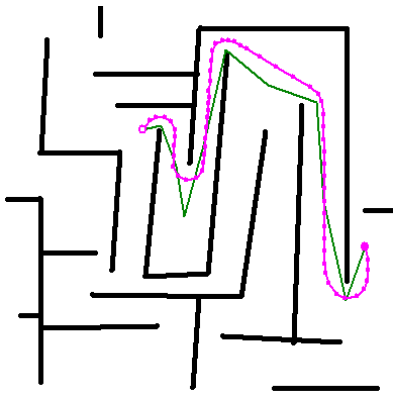
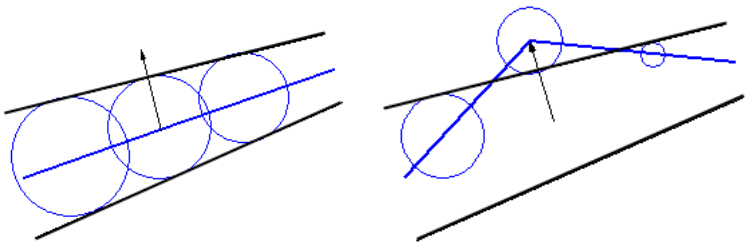


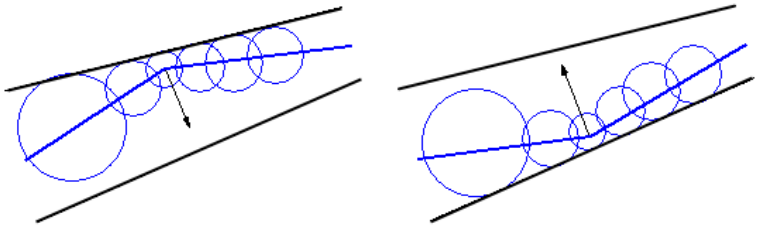
Figura 4.1: il percorso topologico, in verde, e l'*elastic stick* corrispondente, in viola

Per evitare questi spiacevoli casi, dopo aver calcolato l'intensità, la direzione e il verso della forza, controlliamo se questo spostamento porta la bolla oltre un ostacolo. Per questa verifica è sufficiente controllare se la



(a) **(b)**

Figura 4.2: spostamento eccessivo di una bolla



(a) (b)

Figura 4.3: oscillazione di una bolla

nuova posizione *è visibile* (la definizione, banale, di visibilità è stata data nel capitolo 2) dalla precedente. In caso di potenziale passaggio attraverso l'ostacolo, la nuova posizione della bolla verrà calcolata come media tra la posizione corrente e il punto in cui la bolla colliderebbe con l'ostacolo. In questo modo, dopo qualche oscillazione, la bolla raggiungerà una posizione di equilibrio tra i due ostacoli.

Per evitare il fenomeno dell'oscillazione, invece, il criterio da utilizzare è più complesso. Possiamo renderci conto se una bolla sta oscillando, controllando la sua *clearance* prima e dopo la modifica. Se questa aumenta, vuol dire che la bolla si sta spostando verso lo spazio libero e quindi non ci sono problemi. Viceversa, se la bolla nella nuova posizione ha una *clearance* minore, vuol dire che si è avvicinata ad un altro ostacolo e che quindi è possibile che stia oscillando. In questo caso, per evitare il fenomeno, spostiamo la bolla in una posizione media tra l'attuale e quella precedentemente calcolata come nuova. In questo modo la *clearance* certamente aumenterà e, sebbene anche in questo caso sia possibile qualche oscillazione, verrà presto raggiunto un equilibrio.

4.1.1.2 Distanza dagli ostacoli parametrizzabile

E' fondamentale che l'*elastic band* non sia mai troppo vicina agli ostacoli. Infatti stiamo lavorando nello spazio delle posizioni, non in quello delle configurazioni, dobbiamo dunque necessariamente tener conto delle dimensioni del robot. Questo, nelle *elastic band*, si traduce nel fatto

che la *clearance* del percorso, cioè la dimensione di ogni bolla, non deve mai scendere sotto una soglia che in prima approssimazione (o per robot circolari) possiamo considerare uguale al raggio del robot. Nelle *elastic band*, invece, la forza contrattiva interna potrebbe ridurre questa *clearance*, con la spiacevole conseguenza che il robot finirebbe contro gli ostacoli.

Anche questa modifica all'algoritmo di partenza è piuttosto banale. Durante l'inizializzazione, dopo averle rese connesse, spostiamo tutte le bolle lontano dagli ostacoli (qualora questo sia possibile) e ad ogni iterazione successiva dobbiamo sempre evitare di portare le bolle dove avrebbero una dimensione minore di questa soglia.

Concettualmente, la dimensione del robot è l'unico parametro che è indispensabile inserire manualmente. Non c'è modo, infatti, di conoscerla durante il funzionamento dell'algoritmo (a meno di avere sensori che lo misurino). Tutti gli altri parametri (compresi quelli per la DPTM), possono, almeno teoricamente, essere calcolati automaticamente.

4.1.1.3 Modifica delle formule per l'aggiornamento delle bolle

Come abbiamo appena detto, vogliamo cercare il più possibile di evitare parametri per l'utente. Per questo motivo sarebbe opportuno evitare di utilizzare le formule descritte nella pubblicazione di riferimento per le *elastic band* ([ESQuiKha93]). Queste formule, infatti, contengono delle costanti che spetta all'utente definire.

Per quanto riguarda la forza di contrazione, essa deve certamente essere proporzionale alla distanza tra le bolle (più le bolle sono lontane, più la forza di attrazione è grande, il che rispecchia ciò che accade intuitivamente per un elastico). Il nostro fattore di scala sarà proprio la dimensione del robot, come intuitivamente ci si potrebbe aspettare. Se il robot ha un raggio di 10 cm, infatti, e due bolle ad una distanza di 100 cm hanno una certa forza di attrazione, ci si aspetta che per un robot di 50 cm, due bolle ad una distanza di 500 cm abbiano la stessa forza di attrazione.

Dunque il modulo della forza di contrazione f_c è pari a:

$$f_c = \frac{\|\mathbf{b}_{i-1} - \mathbf{b}_i\|}{d_m} + \frac{\|\mathbf{b}_{i+1} - \mathbf{b}_i\|}{d_m}$$

in cui d_m è il raggio del robot (considerandolo circolare), cioè la distanza minima desiderata dagli ostacoli.

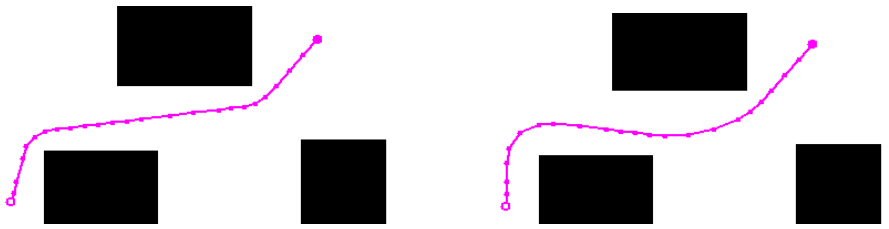
Per quanto riguarda la forza esterna di repulsione dagli ostacoli, dobbiamo considerare che esiste una distanza minima oltre cui le nostre bolle non possono andare, che è proprio d_m, la dimensione del robot. Qualunque bolla venga trovata ad una distanza inferiore viene immediatamente portata a questa distanza minima (se possibile). Oltre questa distanza consideriamo un'altra zona in cui il modulo della forza è proporzionale alla distanza dagli ostacoli. La dimensione di tale distanza la prendiamo, ancora una volta, pari a d_m. In tale zona la forza esterna e quella interna sono libere di trovare una posizione di equilibrio, in modo da rendere la traiettoria curvilinea, priva di punti angolosi.

L'aggiornamento della posizione di ogni bolla viene effettuato seguendo la formula canonica per le *elastic band*, $\mathbf{b}_{new} = \mathbf{b}_{old} + \alpha \mathbf{f}_{tot}$, ma, piuttosto che considerare α proporzionale alla dimensione della bolla, lo manteniamo costante. Il fatto che le bolle piccole si muovano poco e quelle grandi molto ci sembra controintuitivo, infatti, considerando che le bolle piccolo sono quelle più vicine agli ostacoli e hanno quindi più urgenza di essere spostate.

4.2 Gli *elastic stick*

L'algoritmo finale per il livello "locale" del nostro *path-planner* è principalmente una variante delle *elastic band* a cui sono state fatte le modifiche descritte nei paragrafi precedenti e a cui è stato aggiunta un'altra importante caratteristica: abbiamo aggiunto una forza a quelle definite finora, che tende a raddrizzare la traiettoria.

Un *elastic stick* può essere pensato come un bastoncino elastico, che può essere piegato a piacimento ma le sue forze interne tendono a riportarlo nella posizione di riposo, cioè dritto. Questa caratteristica è importante, poiché alcuni tipi di robot consentono manovre con un raggio di curvatura limitato inferiormente. Sebbene attualmente gli *elastic*



(a) senza usare la forza aggiuntiva **(b)** usando la forza aggiuntiva

Figura 4.4: una traiettoria data dall'*elastic stick*

stick non permettano di indicare un minimo raggio di curvatura (come permettono, invece, di indicare la distanza minima dagli ostacoli), la traiettoria, considerando la forza aggiuntiva che tende a raddrizzarla, è molto più agevole da seguire, anche per robot che consentono un raggio di curvatura 0 (come quelli utilizzati negli esperimenti).

Inoltre la scelta di evitare traiettorie con raggio di curvatura piccolo è dovuta anche alla struttura del modulo di controllo, descritto nel prossimo capitolo, che costringerebbe il robot a manovre poco efficienti, in questo caso, per consentirgli di seguire tale traiettoria senza uscire dalle "bolle" di sicurezza.

4.2.1 Bolle "non sicure"

Sebbene l'algoritmo tenda a portare le bolle ad una certa distanza minima predefinita dagli ostacoli, questo chiaramente non è sempre possibile. Per costruzione, l'algoritmo calcola per ogni spostamento la *clearance* corrente di ogni bolla, se questo valore è inferiore alla soglia minima predefinita, la bolla viene dichiarata "non sicura" (*unsafe*). Questo indica al robot che nel punto indicato dalla bolla *non può andare*, perché c'è un passaggio troppo stretto.

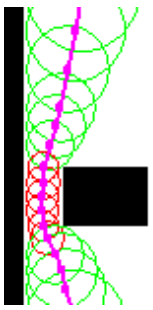


Figura 4.5: bolle "non sicure" in un passaggio stretto

La presenza di bolle non sicure è possibile (sebbene improbabile) anche durante le prime oscillazioni della traiettoria; anche in questo caso, tuttavia, il robot non si dirigerà

verso tali luoghi finché la bolla non diventerà "sicura". Un meccanismo a più alto livello dovrebbe far sì che, qualora sia individuata una traiettoria con bolle non sicure, venga richiesto un nuovo percorso topologico alla DPTM e inizializzato un nuovo *elastic stick* su esso.

4.2.2 Le forze rettificanti

Abbiamo detto che negli *elastic stick* esiste un'altra forza, quella che tende a ottenere una traiettoria con un raggio di curvatura il più ampio possibile, al limite infinito. Dobbiamo tuttavia tener conto che questa è soltanto *una* delle forze che agiscono sulla traiettoria. Essa deve trovare una situazione di equilibrio con le altre due, il che implica che la traiettoria finale *non sarà* a raggio di curvatura massimo.

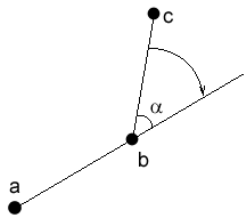


Figura 4.6: la forza rettificante che agisce sulle bolle

Per ogni bolla vengono considerate le due precedenti, come vediamo in figura 4.6. Il nostro obiettivo è quello di portare la bolla c sulla retta che unisce le due bolle precedenti. L'intensità della forza sarà dunque proporzionale all'ampiezza dell'angolo α (al limite essa sarà nulla per $\alpha = 0$, cioè quando la bolla si trova già in linea con le sue due precedenti). Come sempre, per evitare migrazioni lungo l'*elastic stick*, di tale forza dobbiamo considerare soltanto la componente perpendicolare alla traiettoria (e per questo abbiamo bisogno anche di considerare la bolla successiva).

Poiché la prima e l'ultima bolla, corrispondenti alle posizioni iniziale e finale del percorso, non possono essere spostate, il procedimento viene ripetuto anche a ritroso iniziando dall'ultima bolla.

4.2.3 Discesa del gradiente

L'utilizzo di forze per l'aggiornamento della traiettoria fa sì che l'algoritmo esegua una ricerca a discesa del gradiente nella classe di percorsi omotopici a quello di partenza. La differenza con gli algoritmi di piani-

ficazione che utilizzano i campi di potenziale artificiale è ovvia: nel caso degli *elastic stick* l'oggetto manipolato è *la traiettoria*, o, meglio, tutti i suoi punti; al contrario, parlando di *path-planning* mediante l'uso di potenziali artificiali, stiamo considerando *un punto*, cioè la configurazione del robot, e il percorso viene *costruito* muovendo tale punto mediante la forza ricavate dal potenziale artificiale.

Capitolo 5

Implementazione ed esperimenti

5.1 Scelte implementative iniziali

L'implementazione dei due algoritmi ha dovuto seguire varie esigenze a volte contrastanti. La prima è che questo pianificatore doveva essere integrato nell'SPQR-RDK (vedi paragrafo 5.1.2), che funziona in ambiente operativo *Linux*, la seconda era che per ragioni pratiche e di conoscenze, era opportuno avere la possibilità di utilizzarlo anche all'interno di un sistema *Windows*, nel quale era più agevole e veloce sviluppare un software adatto per i test utili allo sviluppo stesso dell'algoritmo.

Il progetto si è dunque sviluppato seguendo queste due strade parallele, grazie alla possibilità di poter compilare uno stesso codice C++ standard sia su sistemi *Linux* che su sistemi *Windows*. I moduli principali di cui è composto il pianificatore sono tre:

- `dptm` è il modulo per il pianificatore globale, esso definisce una classe le cui istanze sono *dynamic probabilistic topological map;*
- `elasticstick` definisce una classe, invece, per il pianificatore locale, ogni sua istanza è un *elastic stick*;
- `ppmap` è la classe, a cui si interfacciano le due precedenti, che si occupa di memorizzare le informazioni sulla mappa e ha vari metodi per le operazioni che si possono fare su di essa, dal calcolo della

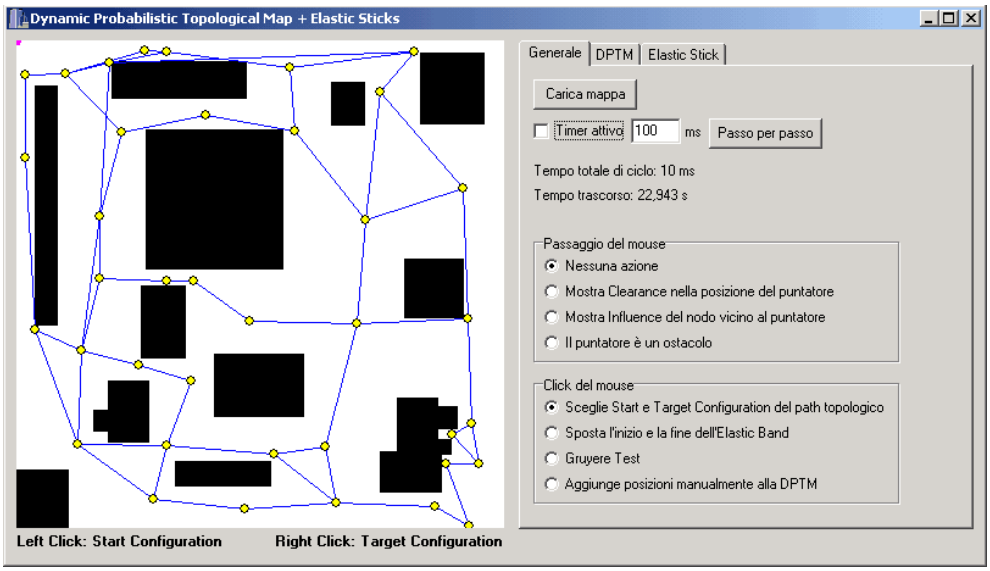


Figura 5.1: l'ambiente di simulazione *DEDemo*

distanza tra due punti fino al calcolo della *clearance* in un punto o se due posizioni sono visibili tra di loro.

Grazie alla presenza di `ppmap`, distinta dal resto, abbiamo la possibilità di fornire ai due algoritmi ambienti a più dimensioni o con rappresentazioni della mappa anche notevolmente differenti, senza che questo si debba riflettere in una modifica negli altri due moduli.

5.1.1 L'ambiente di simulazione *DE-Demo*

Come accennato, abbiamo sviluppato, su sistema *Windows*, un ambiente di simulazione (*DE-Demo*: DPTM and *elastic stick* demonstration) per rendere più agevole lo sviluppo dei due algoritmi. Alcune delle immagini presenti nei capitoli precedenti sono state prese da questo software. In questo ambiente abbiamo la possibilità di osservare, graficamente, le evoluzioni della *dptm* e degli *elastic stick*. E' inoltre possibile intervenire in tempo reale su tutti i parametri e caricare vari tipi di mappa, in modo da semplificare lo studio di quelle particolari caratteristiche che possono essere individuate soltanto nella pratica.

Grazie a questo software è stato anche possibile rendere gli algoritmi

più efficienti, poiché è stato possibile tenere sott'occhio i tempi di esecuzione e definire le situazioni in cui l'efficienza peggiorava. E' tuttavia chiaro che si tratta di un software che non tiene in considerazione dei fattori ambientali reali quali l'imprecisione dei sensori o della localizzazione, è dunque uno strumento per lo sviluppo *teorico* degli algoritmi, che poi devono necessariamente essere testati sui robot reali.

Sempre con *DE-Demo* abbiamo avuto la possibilità di confrontare varie scelte per i molteplici aspetti degli algoritmi, nonché di comparare le prestazioni della DPTM con altri algoritmi, come, ad esempio, le PRM (i risultati di questi test e una discussione su di essi si trovano nella sezione 6.1.1).

5.1.2 L'ambiente *SPQR-RDK* e l'uso di *Stage/Player*

L'ambiente *SPQR-RDK* è stato sviluppato all'interno del DIS inizialmente come *framework* da utilizzare per le competizioni di *Robocup Soccer*. Successivamente le sue funzionalità sono state estese e viene utilizzato in vari ambienti e con tipi diversi di robot (dai *Pioneer* della *ActivMedia Robotics* ai *Sony AIBO*) e dispositivi di *input* (sonar, laser scanner).

Grazie a questo ambiente è stato molto agevole interfacciare il *path-planner* con il robot reale e guidarlo da una postazione remota grazie all'uso di *schede di rete wireless*. In tale ambiente di sviluppo il pianificatore di percorsi si configura come uno dei vari *task* che vengono avviati periodicamente dal processo centrale. Gli scambi di informazioni tra i vari *task* e i moduli di interfaccia con i sensori rendono disponibili in tempo reale i dati di cui il *path-planner* ha bisogno per funzionare: la posizione del robot, l'obiettivo da raggiungere e le letture dai sensori per individuare eventuali ostacoli.

Il progetto *Player/Stage* è uno strumento di sviluppo *Open Source* che offre sia un'interfaccia con gli attuatori e i sensori di un robot (*Player*), sia un ambiente di simulazione configurabile completo di robot, sensori e oggetti (*Stage*). Grazie a questo *software*, interfacciato con l'*SPQR-RDK*, possiamo agevolmente simulare i movimenti e le letture sensoriali

di un robot in ambienti simili a quelli reali, ottenendo situazioni più realistiche rispetto agli esperimenti effettuati con *DE-Demo*. Questo passo, precedente alle simulazioni reali, è di fondamentale importanza pratica perché riduce i tempi di *test*.

5.2 Estensioni dell'implementazione

In un ambiente reale i due algoritmi hanno dovuto confrontarsi con situazioni non considerate in simulazione. Ferme restando le caratteristiche peculiari dei due algoritmi, sono stati necessarie alcune piccole estensioni. La questione più importante è quella dovuta alla mappa che viene utilizzata.

E' opportuno che gli algoritmi siano costruiti in modo da funzionare in maniera indipendente dal tipo di rappresentazione che si ha dell'ambiente, a meno di ovvie considerazioni sull'efficienza. Questa proprietà è stata ottenuta grazie alla divisione dei moduli, come visto in precedenza, in modo tale da delegare al modulo `ppmap` tutte le funzionalità relative alla rappresentazione e alle operazioni che è possibile fare sulla descrizione dell'ambiente.

Il modulo `ppmap` è in grado di gestire sia mappe vettoriali che *bitmap*. Sebbene siano preferite le prime per via della loro compattezza e della maggiore efficienza di funzioni quali il *ray-tracing*, è necessario considerare anche il supporto alle seconde, poiché, ad esempio, molti strumenti che sono in grado di fornire una mappa partendo dalle letture dei sensori, forniscono un output sotto forma di *bitmap*.

E' inoltre opportuno poter fare una distinzione tra mappa statica e mappa dinamica. Mentre la prima è fornita al robot dall'utente, ed è immutabile, la seconda riflette le eventuali modifiche dell'ambiente stesso, cioè la presenza di ulteriori ostacoli non previsti. Tutte le funzioni richieste al modulo `ppmap` dovranno considerare entrambe le mappe come se fossero fuse in una sola, che rispecchia l'ambiente in un preciso istante.

L'utilità di una mappa statica fornita dall'utente è chiara se si pensa che i sensori del robot potrebbero non essere in grado di rilevare tutti gli ostacoli. Un *laser scanner* ad esempio, non percepisce la presenza di un

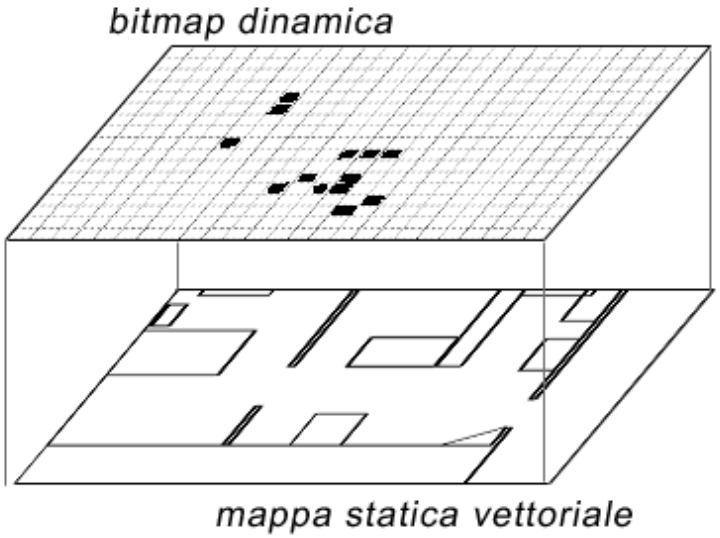


Figura 5.2: i due strati della mappa

vetro trasparente o di ostacoli che si trovano più in alto o più in basso rispetto al piano su cui viene effettuata la scansione.

5.2.1 Ostacoli non previsti nella mappa statica

Entrambi gli algoritmi sono stati pensati e costruiti per reagire in maniera solida ai cambiamenti dell'ambiente, dunque non hanno bisogno di grosse modifiche. Ciò che invece bisogna tenere in considerazione è l'aggiunta dei nuovi ostacoli percepiti dai sensori. Abbiamo deciso di rappresentare questi ostacoli su una *bitmap*, per semplificare l'implementazione. In gran parte degli esperimenti abbiamo una mappa *vettoriale* che rappresenta la mappa statica, fornita in ingresso prima dell'esecuzione, dobbiamo considerare la rappresentazione finale dell'ambiente formata da due strati sovrapposti, come mostrato in figura 5.2. Nel primo stato sono rappresentati tutti i segmenti della mappa statica, nel secondo, sotto forma di *bitmap*, tutti gli ostacoli rilevati dai sensori.

5.2.1.1 Mappa statica e mappa dinamica

Possono essere presenti nell'ambiente degli ostacoli non previsti per tre principali motivi:

1. è presente un oggetto inanimato che non veniva considerato nella mappa;
2. un oggetto si sta muovendo, ad esempio una persona;
3. è stato spostato un oggetto che nella mappa statica aveva un'altra posizione.

Nella mappa statica vengono indicati generalmente tutti quegli ostacoli che si possono considerare fissi, come i muri, gran parte dei mobili, ecc., ma è possibile che sia stato indicata la posizione di una sedia o di un altro oggetto che successivamente è stato spostato. Non essendo in grado di stabilire se l'oggetto *non è più* nella posizione in cui ci si aspettava di vederlo, o semplicemente i sensori non sono in grado di percepirlo, gli ostacoli nella mappa statica vengono considerati presenti *sempre*, a prescindere che i sensori ne confermino l'esistenza o meno.

Per questo motivo, non siamo in grado di comportarci adeguatamente per il caso 3, tuttavia è doveroso considerare che ciò è dovuto principalmente all'incapacità dei sensori di fornire una descrizione completa dell'ambiente. In alternativa, se i sensori fossero completamente affidabili (o fossero l'unica fonte di informazioni) sarebbe sufficiente non fornire al robot la mappa statica, o, meglio, fornirgliela sotto forma di mappa dinamica, cioè aggiornabile. La mappa statica, per i motivi appena visti, *non è aggiornabile*.

5.2.1.2 Strategie di aggiornamento della mappa dinamica

Sono possibili varie strategie per l'aggiunta di ostacoli nella mappa dinamica e per la loro rimozione. La scelta di un metodo piuttosto che un altro deve essere fatta in relazione al tipo di ambiente e soprattutto al tipo di sensori che vengono utilizzati. Uno degli aspetti fondamentali

è la considerazione delle letture errate che i sensori possono fornire, che rischiano di falsare la conoscenza che il robot ha dell'ambiente.

Abbiamo individuato alcuni criteri che sono poi stati utilizzati, contemporaneamente o meno, nei vari esperimenti con robot e ambienti reali.

1. E' possibile impostare un *time to live* (TTL) per gli ostacoli: ogni ostacolo, se non viene aggiornato (cioè viene individuata nuovamente la sua presenza), verrà eliminato dopo TTL millisecondi. In questo modo le eventuali letture errate e gli ostacoli "fantasma" scompariranno senza creare descrizioni falsate dell'ambiente per troppo tempo. Uno dei problemi principali del TTL è che un valore troppo basso rischia di far "dimenticare" troppo presto al robot degli ostacoli reali che sono fuori dal campo di lettura dei sensori, un valore troppo alto, invece, lo fa attardare inutilmente in attesa della scomparsa di un ostacolo inesistente.

2. La pulizia degli ostacoli non presenti nel campo di lettura dei sensori. Se riusciamo a "vedere" un ostacolo *dietro* uno che avevamo precedentemente posto nella mappa dinamica, questo indica che l'ostacolo precedente non è più lì e può essere eliminato. Anche questo criterio porta ad alcuni problemi, soprattutto se vengono usati sensori che possono, anche di rado, fallire nell'individuare ostacoli presenti in realtà.

3. La lettura di alcuni sensori spesso oscilla, soprattutto se gli ostacoli sono molto lontani. In questo caso possiamo rendere necessarie un certo numero di letture consecutive prima di aggiungere l'ostacolo nella mappa dinamica.

4. I sensori hanno spesso un intervallo di lettura ben preciso, specificato nelle caratteristiche tecniche del dispositivo. Questo implica che tale sensore non darà mai valori inferiori o superiori alle due soglie, oppure che i valori oltri questi limiti devono essere scartati perché non affidabili. Dobbiamo dunque prevedere anche la possibilità, di implementazione banale, di scartare eventuali letture fuori

dall'intervallo consentito. Riguardo a quest'ultima eventualità, bisogna tuttavia far attenzione che mentre non è fondamentale avere letture lontane, poiché il robot può limitarsi a pianificare globalmente utilizzando le informazioni già memorizzate in precedenza, è importante che non vengano mai cancellate le letture troppo vicine al robot, quelle, cioè, inferiori alla soglia minima di lettura del sensore. E' possibile che esistano ostacoli, visti in precedenza, che, ad esempio, il sonar o lo scanner laser non siano più in grado di vedere. Se questi ostacoli vengono eliminati, il robot si scontrerà con essi, in quanto non ha più nessuna possibilità di vederli. L'unica strategia possibile, in questo caso, è quella di non eliminare mai ostacoli troppo vicini al robot, anche se il loro TTL imporrebbe di cancellarli.

5.2.2 Comportamento dinamico

Gli *elastic stick* e le DPTM sono state pensate e progettate per funzionare in ambiente dinamico. Essi, in particolar modo gli *elastic stick*, sono in grado di aggiornarsi per riflettere le modifiche man mano individuate nell'ambiente. Queste modifiche possono essere sia reali che dovute a errori di letture dai sensori, dai dati dell'odometria, dalla localizzazione o, infine, a settaggi errati nei parametri di aggiornamento della mappa dinamica.

5.2.2.1 Gli *elastic stick*

Cominciamo col prendere in considerazione l'algoritmo di livello più basso, che è anche, per convenienza, il più reattivo. E' fondamentale che le traiettorie tracciate dall'*elastic stick* vengano modificate in tempo reale, in caso contrario il robot finirà presto contro un ostacolo, sebbene questo sia stato percepito. Fortunatamente questo algoritmo, per come è stato costruito, compie molto bene questo compito, deformando la traiettoria elastica in caso venga individuato qualche oggetto, anche in movimento. La caratteristica che obbliga l'*elastic stick* a restare distante di una certa misura da qualsiasi ostacolo, impedisce al robot di scontrarsi con essi.

Dichiariamo l'*elastic stick* "non sicuro" (*unsafe*) qualora una delle bolle di cui è formato ha una dimensione minore alla distanza minima consentita dagli ostacoli. Questo implica che in quel punto della traiettoria il robot non può passare, perché il passaggio è troppo stretto. In tal caso abbiamo due scelte: attendere che eventualmente il passaggio diventi più largo, permettendoci di passare, oppure richiedere un altro percorso e dichiarare questo non utilizzabile.

La nostra scelta è di comportarsi alternativamente in entrambi i modi, dato che sono possibili entrambe le situazioni (il robot non ha nessuna possibilità di valutare se un certo ostacolo potrebbe spostarsi oppure no, a meno che questo appartenga alla mappa statica). Inoltre, durante la ricerca di nuovi percorsi il robot continua a seguire la traiettoria "non sicura", fermandosi, eventualmente, prima della bolla *unsafe*. Questo comportamento è dovuto al fatto che sono possibili delle situazioni transitorie per quanto riguarda gli ostacoli, dovute anche alla discretizzazione degli ostacoli dinamici, alla non perfetta odometria e così via: se il robot si fermasse ogni volta che qualcosa non va nella sua traiettoria, avrebbe un andamento "a scatti" prima che la traiettoria si stabilizzi (ammesso che lo faccia). Nell'implementazione esistono dunque due istanze di *elastic stick*, una è quella che il robot segue, l'altra è quella con cui cerchiamo un percorso alternativo, solo nel caso questo avvenga le due traiettorie vengono scambiate e il robot inizia a seguire quella "migliore" (cioè priva di tratti non sicuri).

E' anche possibile che l'*elastic stick* si "spezzi", cioè che compaiano degli ostacoli tali che la traiettoria vi passi attraverso. In questo caso viene semplicemente richiesta al modulo al livello più alto (DPTM) un nuovo percorso topologico.

5.2.2.2 La DPTM

La DPTM, lavorando con informazioni globali, è più lenta a recepire i cambiamenti. Tuttavia ogni arco che viene trovato a passare attraverso un ostacolo viene immediatamente eliminato e nuovi collegamenti creati intorno ad esso (dando in *input* alla rete delle posizioni vicine all'ostacolo che ha causato l'eliminazione dell'arco) in maniera da ricostruire la nuova

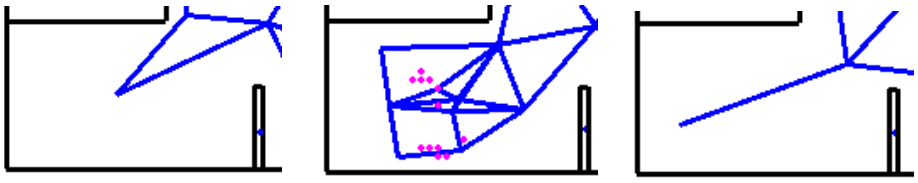


Figura 5.3: adattamento e successiva semplificazione della DPTM in relazione alla presenza di ostacoli aggiuntivi

topologia dell'ambiente. Quando appare qualche ostacolo nuovo, dunque, la rete incrementa il suo numero di nodi e archi; nel momento in cui l'ostacolo non viene più individuato dai sensori (o ha terminato il suo tempo di vita e non è stato aggiornato) e viene quindi eliminato, la DPTM, grazie ai suoi criteri di semplificazione, è in grado di semplificarsi e tornare alla situazione iniziale (figura 5.3).

5.2.3 Il modulo di controllo

Il modulo di controllo è piuttosto semplice. Esso, infatti non deve curarsi di evitare eventuali ostacoli, dato che questo compito è assolto dall'*elastic stick*. Il suo unico scopo è dunque quello di seguire il percorso fornitogli dal pianificatore locale, che si trova concettualmente al livello superiore nella gerarchia.

Dal punto di vista del controllo i robot che abbiamo utilizzato possono essere assimilati al modello dell'*uniciclo*. Un ***uniciclo*** è formato da una singola ruota, libera di rotolare sul piano. Essa è sottoposta ad un vincolo di non olonomia, in quanto le ruote non possono slittare in direzione laterale. Detta (x, y) la posizione dell'uniciclo nel piano cartesiano e θ il suo orientamento rispetto all'asse x, tale vincolo è esprimibile con la seguente equazione:

$$\dot{x} \sin\theta - \dot{y} \cos\theta = 0$$

Essa esprime il fatto che il robot può traslare soltanto lungo la direzione data da θ. Per quanto riguarda le equazioni cinematiche del robot, esse sono:

$$\dot{x} = v \cdot \cos\theta$$
$$\dot{y} = v \cdot \sin\theta$$
$$\theta \dot{=} \omega$$

In cui v e ω, rispettivamente la velocità lineare e quella angolare, saranno i "comandi" che il modulo di controllo passerà all'*hardware* del robot.

Le problematiche relative al controllo di un robot si possono dividere in due grandi categorie: l'inseguimento di traiettoria e la stabilizzazione in un punto. Per motivi di semplicità realizzativa, abbiamo deciso di utilizzare il *controller* preesistente all'interno dell'*SPQR-RDK*, che realizza la stabilizzazione in un punto implementando il metodo basato sulla trasformazione in coordinate polari descritto in [WMRCtrl].

Un'altra motivazione riguardo all'uso di questa tecnica (stabilizzazione in un punto) è relativa al fatto che la pianificazione di movimento richiede il raggiungimento *di un punto*, sia esso nello spazio di lavoro o in quello delle configurazioni. Questo implica che se utilizzassimo l'inseguimento di traiettoria sarebbe necessaria un'altra legge di controllo per stabilizzare il robot nel punto di arrivo p_{goal} o q_{goal}. Infine, la "traiettoria" in uscita dal modulo di pianificazione di percorso, non è una curva su cui è definita una legge oraria, come sarebbe necessario per utilizzare l'inseguimento di traiettoria. Sebbene sia possibile, dato l'insieme di punti che rappresentano i centri delle bolle dell'*elastic stick*, ricavare una curva (ad esempio mediante l'uso di *spline*) e su di essa definire una legge oraria, questo accrescerebbe la complessità computazionale del modulo.

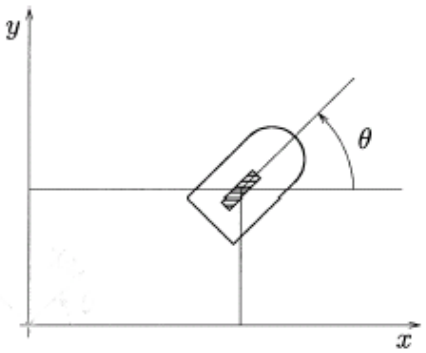


Figura 5.4: il modello dell'uniciclo

Per far sì che, utilizzando questo criterio, il robot segua la traiettora indicata dall'*elastic stick* fino all'obiettivo, dobbiamo definire *verso quale*

punto la legge di controllo deve stabilizzarlo. Tale punto non può ovviamente essere q_{goal}, in quanto le leggi di controllo di cui stiamo parlando non considerano la presenza di ostacoli (altrimenti non avrebbe avuto senso la pianificazione di percorso ai livelli più in alto nella gerarchia). Una scelta semplice può essere di prendere un punto presente sull'*elastic stick*, ad una distanza fissa predefinita d. Il robot, mediante la legge di controllo, segue dunque un punto virtuale che si muove sulla traiettoria rimanendo a distanza costante dal robot. Ovviamente tale distanza decresce quando il robot si trova ad una distanza $d' < d$ dal punto di arrivo.

Poiché l'*elastic band* è un insieme di punti, abbiamo comunque bisogno di un'interpolazione per ottenere un punto da seguire che abbia una distanza costante dal robot. La nostra scelta ricade su un'interpolazione lineare, effettuata come segue. Detta $\delta(\mathbf{b})$ la distanza di ogni bolla dal robot, e detti $\inf(d)$ e $\sup(d)$ gli insiemi di bolle rispettivamente con $\delta(\mathbf{b}) < d$ e $\delta(\mathbf{b}) \geq d$, selezioniamo le due bolle $\mathbf{b}_{inf} \in \inf(d)$ e $\mathbf{b}_{sup} \in \sup(d)$ tali che esse abbiano $\delta(\mathbf{b})$ massima e, rispettivamente, minima all'interno degli insiemi di appartenenza. Dopodiché effettuiamo un'interpolazione lineare tra le due bolle appena selezionate, in modo che il punto risultante sia a distanza d dal robot.

Infine, dette (x_d, y_d, θ_d) le coordinate di questo punto e definendo la seguente trasformazione in coordinate polari:

$$
\begin{aligned}
e &= \sqrt{(x - x_d)^2 + (y - y_d)^2} \\
\alpha &= \text{atan2}(y_d - y, x_d - x) \\
\gamma &= \alpha - \theta + \pi \\
\delta &= \gamma - \theta_d
\end{aligned}
$$

(in cui per atan2 si intende la funzione arcotangente definita su tutti e quattro i quadranti e indeterminata solo se entrambi gli argomenti sono uguali a 0) le leggi di controllo sono ricavabili con una tecnica simile a quella di Lyapunov e risultano essere:

$$v = k_1 \cdot e \cdot \cos\gamma$$
$$\omega = k_2 \cdot \gamma + k_1 \cdot \left(\frac{\cos\gamma \cdot \sin\gamma}{\gamma}\right) \cdot (\gamma + k_3 \cdot \delta)$$

ove k_1, k_2 e k_3 sono tre costanti positive. Il vantaggio di questa tecnica è costituito principalmente dalla semplicità di taratura dei parametri e di implementazione in genere.

Uno dei difetti dell'utilizzo di un metodo a stabilizzazione in un punto è che la traiettoria per raggiungere tale punto non è definita. Tuttavia bisogna precisare che innanzitutto il robot si trova *sempre* sulla traiettoria, perché essa è costruita prendendo la posizione del robot come suo punto iniziale. In secondo luogo, per evitare percorsi imprevisti e soprattutto manovre all'indietro, pericolose per il fatto che il robot non può vedere se vi sono ostacoli, l'algoritmo procede ad un controllo preliminare su γ (l'angolo tra la direzione attuale del robot e la direzione in cui si trova il punto di arrivo) e se esso è superiore ad una certa soglia, esegue una preliminare rotazione con $v = 0$ per ridurre l'angolo γ. Questo ci consente una certa sicurezza nei movimenti, soprattutto considerando che da quel momento in poi la traiettoria che viene seguita può essere considerata priva di punti di discontinuità e con un raggio di curvatura sufficientemente elevato per evitare comportamenti indesiderati.

5.2.4 Il tempo di ciclo

Le iterazioni dell'algoritmo DPTM hanno un tempo proporzionale al numero di nodi e archi contenuti nella rete. Poiché presto si raggiunge una condizione di equilibrio, questo tempo tende a stabilizzarsi. Per gli *elastic stick*, invece, la situazione è completamente differente. Innanzitutto la DPTM continua ad aggiornarsi senza soluzione di continuità, il pianificatore locale, invece, ha bisogno di essere richiamato soltanto quando viene richiesto un percorso. Il tempo di esecuzione di ogni iterazione, inoltre, è una funzione della lunghezza del percorso (cioè della distanza topologica tra la posizione iniziale e quella finale) e della sua complessità

(più ostacoli sono presenti e più essi sono vicini tra di loro, più numerose saranno le bolle da aggiornare).

Poiché, anche per l'interazione con gli altri moduli, è necessario che il tempo di esecuzione totale sia piuttosto costante, è stato deciso di variare il numero di iterazioni ad ogni ciclo dell'algoritmo in relazione alla durata delle stesse. In situazioni in cui non è richiesto alcun percorso, il tempo può essere impiegato completamente per le iterazioni della DPTM, che può essere invece anche fortemente ridotto qualora sia presente un *elastic stick* molto pesante in termini di calcolo.

Nell'architettura hardware e software (*SPQR-RDK*) utilizzata è stato posto un limite di 50 millisecondi al tempo di ciclo totale del modulo del *path-planner*. In tale tempo la DPTM è in grado di eseguire più di 60 iterazioni quando non è presente l'*elastic band*, mentre quando i due algoritmi lavorano insieme le iterazioni non scendono sotto le 20 (questi risultati sono stati ottenuti compilando il codice in maniera ottimizzata, altrimenti le prestazioni sono dimezzate). E' stato sperimentato che il modulo riesce a dare buoni risultati finché le iterazioni non scendono sotto le 10 per ogni ciclo.

Capitolo 6

Sperimentazione

6.1 Simulazioni

6.1.1 Esperimenti mediante *DE-Demo*

Grazie all'uso di *DE-Demo* è stato possibile paragonare le prestazioni dell'algoritmo DPTM a quelle delle PRM. Essenzialmente è stato sufficiente disattivare tutti i controlli presenti in DPTM e aggiungere nodi in ogni posizione di input. L'algoritmo PRM necessita anche di parametrizzare una distanza massima di collegamento, per evitare un'eccessivo numero di archi nel grafo finale. Abbiamo impostato questa distanza al valore che da una serie di esperimenti iniziali è risultato essere il migliore relativamente alle mappe utilizzate.

Inoltre, per poter paragonare effettivamente i due metodi, la generazione di posizioni casuali propria delle DPTM è stata mantenuta anche per i test con le PRM, cioè è stato utilizzato il *gruyere method* alternato a una generazione totalmente casuale. In caso contrario le prestazioni della PRM sarebbero state notevolmente inferiori. D'altra parte questo metodo di produzione di input ottiene un risultato molto simile a quello descritto in [PRMWilAmaSti].

Sono state prese in considerazione quattro mappe tra loro differenti per caratteristiche. I dati registrati riguardano il tempo necessario a raggiungere la descrizione topologica completa di W_{free} e il numero di nodi e archi in quel momento. Ancora una volta è opportuno ricordare

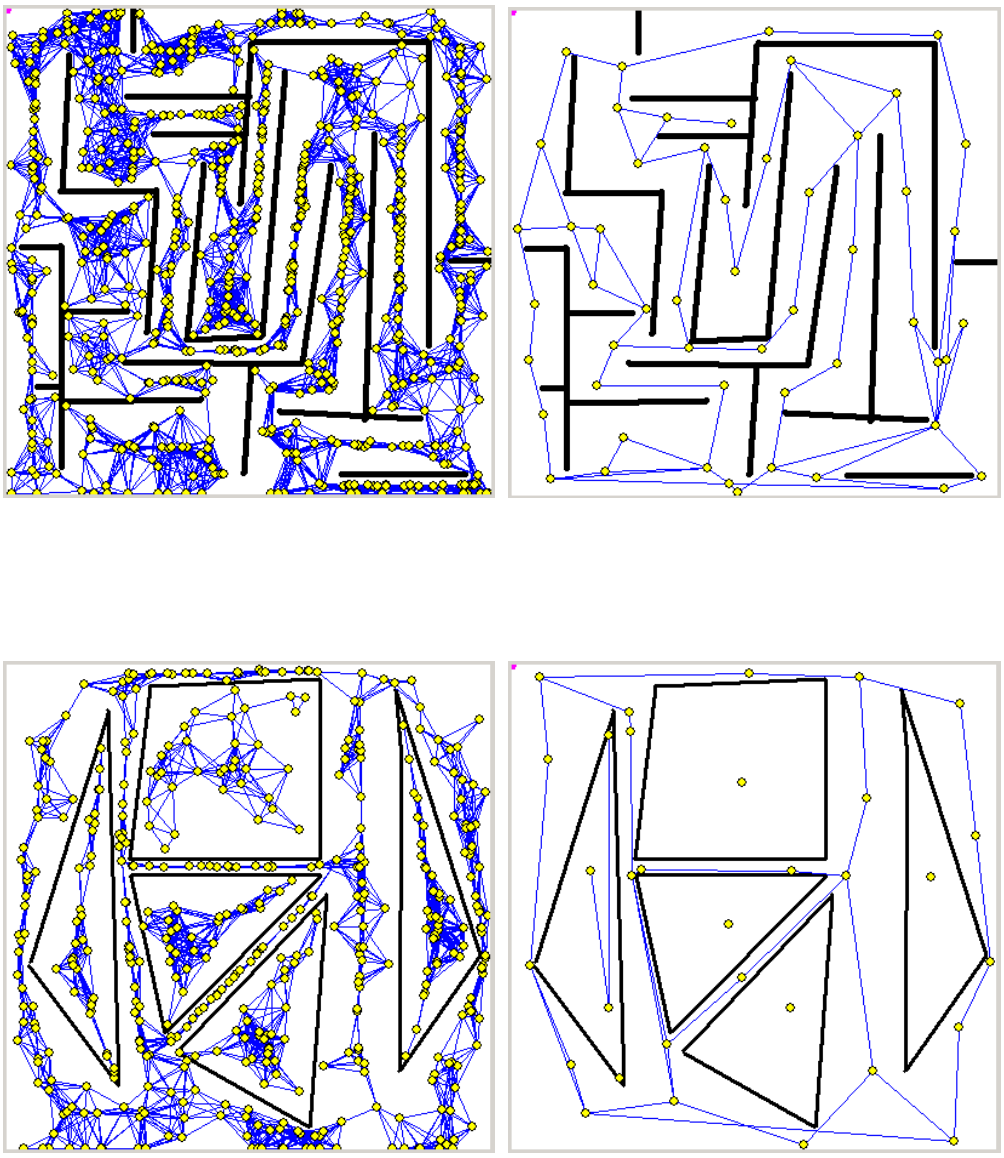


Figura 6.1: due esempi di mappe utilizzate (“Labirinto” e “Passaggi stretti”) a sinistra le PRM e a destra le DPTM

Mappa	t	N	A
Rettangoli e segmenti	9,4 s	597,7	5358,9
Labirinto	16,6 s	926,4	9659,9
Passaggi stretti	17,5 s	893,1	9565,1
Regioni differenti	25,9 s	858,7	11092,4

Mappa	t	N	A
Rettangoli e segmenti	19,7 s	42,5	67,0
Labirinto	31,5 s	53,2	70,0
Passaggi stretti	11,7 s	31,8	34,2
Regioni differenti	22,0 s	50,5	68,0

Tabella 6.1: risultati dei test utilizzando le PRM (sopra) e le DPTM (sotto)

che mentre PRM ha la necessità di essere bloccata da parte di un utente, DPTM può proseguire indefinitamente, raggiungendo una situazione di equilibrio (in termini di nodi e archi) quasi identica a quella che si ha al momento in cui si raggiunge la completezza topologica.

Come è possibile osservare dalla tabella 6.1, a fronte di un intervallo temporale molto simile, il numero di nodi e archi necessari per raggiungere l'obiettivo è notevolmente inferiore (il rapporto è pari a circa 5% per quanto riguarda i nodi e 1% per gli archi), il che si riflette in tempi decisamente più veloci per il calcolo, ad esempio, di un percorso. E' appena il caso di osservare nuovamente, inoltre, che tale caratteristica è importante proprio per permettere alla DPTM di essere una rete dinamica: avere troppi nodi e archi da aggiornare ad ogni ciclo, in caso di variazioni dell'ambiente, potrebbe risultare oltremodo oneroso in termini di calcolo.

Individuare con esattezza il momento in cui la rete, PRM o DPTM, ha raggiunto lo stato in cui la consideriamo una *descrizione topologicamente equivalente* dello spazio di lavoro non è una cosa semplice. Il criterio teorico è che *qualunque* traiettoria possibile nello spazio W sia *omotopica* ad un cammino sulla rete. Poiché non è possibile individuare un algoritmo in grado di valutare questo criterio (perché deve valere per

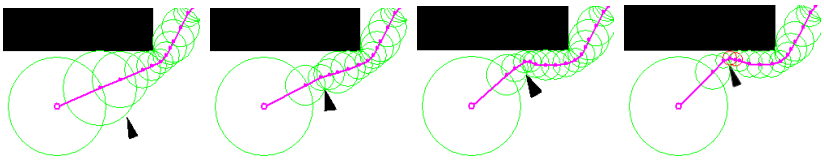


Figura 6.2: il comportamento degli *elastic stick* simulato con *DE-Demo*

tutte i percorsi ammissibili in W) , la condizione di terminazione è stata decisa "a occhio" da un utente.

Grazie a *DE-Demo* è anche possibile studiare il comportamento degli *elastic stick* con gli ostacoli in movimento. In figura 6.2 possiamo vedere come la traiettoria rimanga sempre ad una distanza predefinita dagli ostacoli, siano essi fissi o in movimento, mantenendo una traiettoria smussata; inoltre, nell'ultima immagine alcune bolle sono diventate "non sicure", a questo punto l'algoritmo chiederà alla DPTM un nuovo percorso topologico e con esso inizializzerà un nuovo *elastic stick*.

6.1.2 Esperimenti con *Player/Stage*

Il modulo della localizzazione in questo tipo di esperimenti è stato disattivato, dunque il robot si localizza utilizzando semplicemente l'odometria, che è ovviamente perfetta. Per quanto riguarda il sensore utilizzato, esso è un *range finder* ideale, che può essere dunque paragonato ad un *laser scanner*.

6.1.2.1 Esperimento 1: traiettoria semplice

Nel nostro primo esperimento in *Player/Stage* abbiamo scelto di utilizzare la mappa del *domestic environment* costruita per il progetto *RoboCare* (vedi paragrafo successivo). La situazione simulata è simile all'esperimento che verrà fatto successivamente nell'ambiente reale. Sebbene il percorso sia molto semplice, grazie ad esso siamo già in grado di osservare alcuni dei problemi che possono nascere dalle letture dei sensori. Poiché l'aggiornamento dell'odometria del robot non è immediato, soprattutto durante le rotazioni le letture dei sensori rispecchiano l'ambiente nell'at-

timo precedente, con la conseguenza di far apparire degli ostacoli in punti della mappa che in realtà sono liberi.

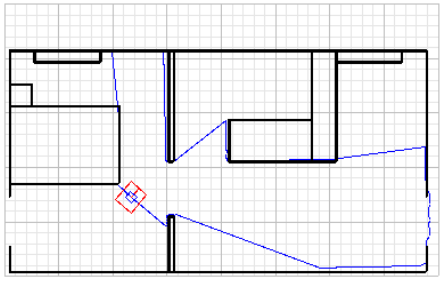


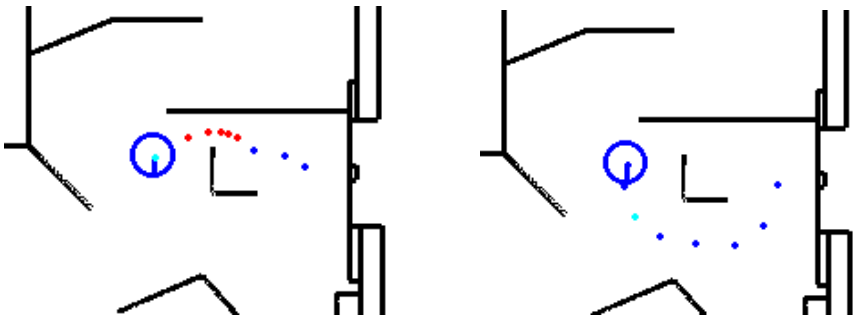
Figura 6.3: passaggio del robot vicino allo spigolo

La soluzione a tale problema, già considerato durante la progettazione, è l'uso combinato dell'eliminazione, all'interno del campo di lettura dei sensori, degli ostacoli non più percepiti e la necessità di più letture consecutive prima di consentire l'aggiunta dell'ostacolo sulla mappa. Un'altra possibile alternativa sarebbe quella di utilizzare il TTL, ma in questo caso rischieremmo di perdere (cioè "dimenticare") anche ostacoli in altre regioni della mappa che rispecchiano degli oggetti effettivamente presenti nella realtà.

Essendo la simulazione particolarmente precisa (a parte le letture imprecise durante le rotazioni), la distanza minima dagli ostacoli consentita all'*elastic stick* è stata settata particolarmente "stretta", il robot, infatti, esegue delle traiettorie che sfiorano gli spigoli degli oggetti nella mappa (figura 6.3). Questa caratteristica è indice che le traiettorie sono particolarmente ottimizzate. In ambienti reali, naturalmente, sarà necessario aumentare questo valore per evitare che gli errori di letture sensoriali causino delle collisioni.

6.1.2.2 Esperimento 2: percorsi alternativi

Un'altra caratteristica del nostro algoritmo, che è possibile provare in simulazione, è la capacità di cercare percorsi alternativi qualora un passaggio venga dichiarato troppo stretto, o comunque non sicuro. La mappa scelta in questo caso è quella dell'arena costruita all'Istituto Superiore Antincendio per gli esperimenti per la *RoboCupRescue* (paragrafo 6.2.2). Tale mappa è stata progettata proprio per lo studio di situazioni particolari. In questo caso, possiamo vedere che il passaggio topologico attraverso cui il robot dovrebbe passare, è troppo stretto. Tale giudizio viene effettuato dall'*elastic stick*, che si trova ad avere delle bolle troppo piccole



(a) (b)

Figura 6.4: passaggio non sicuro e traiettoria alternativa

e quindi non sicure. A questo punto, l'algoritmo dichiara "non sicuro" l'arco della DPTM a cui corrisponde la bolla troppo piccola e ricerca un altro percorso topologico. La ricerca continua finché non viene trovato un percorso sicuro (figura 6.4).

6.1.2.3 Esperimento 3: ostacolo fantasma

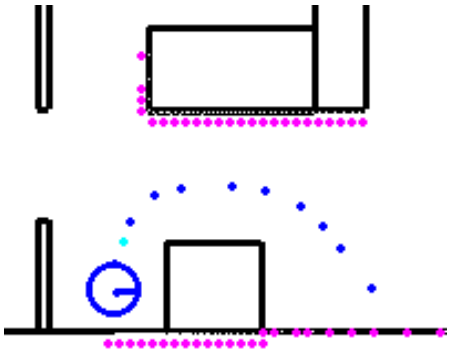


Figura 6.5: un ostacolo non individuabile dai sensori

Abbiamo già detto dell'importanza, in alcuni casi, della presenza di una mappa statica. In essa, infatti, possono essere inseriti tutti quegli ostacoli che non possono essere individuati dai sensori. Per questo esperimento utilizziamo nuovamente la mappa del *domestic environment* (nell'ambiente reale sono effettivamente presenti degli ostacoli non rilevabili con i sensori, come descritto nel paragrafo successivo). Dalla figura 6.5 è evidente che i sensori non rilevano il rettangolo in basso (non ci sono punti viola intorno al suo profilo) e che la traiettoria, invece, ne tiene conto.

6.2 Esperimenti in ambiente reale

6.2.1 *Domestic Environment* per la *RoboCare*

Il progetto RoboCare si pone come obiettivo di costruire un sistema multi agente che generi servizi di assistenza in ambiente domestico. L'obiettivo finale è quello di realizzare un sistema di robotica cognitiva (composto da più robot e dispositivi di ausilio all'utilizzo dei robot) in grado di svolgere un insieme di funzionalità per l'assistenza di persone anziane o diversamente abili sia in un ambito domestico che in una residenza attrezzata (casa di cura, ospedale).

In tali ambienti, il *path-planning* ha a che fare con eventuali esseri umani presenti, che chiaramente hanno la possibilità di muoversi. Sono stati fatti degli esperimenti per valutare la robustezza degli algoritmi in presenza di ostacoli in movimento.

Nei locali dell'Istituto di Scienze e Tecnologie della Cognizione - CNR è stato individuato e progettato l'ambiente di test di tipo domestico da utilizzare per gli esperimenti. La metratura è sufficiente per la realizzazione di un bilocale.

Il robot utilizzato per gli esperimenti in questo ambiente è un Pioneer 2-DX, della *ActivMedia Robotics*, su cui viene montato un *laptop* con processore a 2 GHz, in grado di interagire con essa e con il dispositivo *laser scanner Hokuyo*, utilizzato dal *path-planner* per individuare eventuali ostacoli sulle traiettorie non indicati nella mappa statica dell'ambiente.

D'altra parte in tale ambiente sono presenti alcuni oggetti non rilevabili dai sensori, perché ad una altezza maggiore rispetto a quella a cui è montato il *laser scanner*, (tavoli, sedie, lavandino), per i quali è necessaria la presenza della mappa statica.

Il *laptop* è provvisto di una scheda di rete *wireless* grazie alla quale è possibile connettersi ad esso da un PC fisso remoto.

Il tipo di *laser* utilizzato fornisce letture falsate a distanze superiori ai 2 metri. Anche questa possibilità era stata considerata in sede di implementazione e viene risolta considerando valide solo le letture entro un certo intervallo. Inoltre, poiché anche in questo modo le letture a volte subiscono delle oscillazioni, abbiamo deciso di applicare anche il criterio

che impone un certo numero di letture consecutive prima di inserire un ostacolo. Infine, il laser non individua ostacoli se questi si trovano ad una distanza di pochi centimetri da esso. Questo potrebbe avere effetti indesiderati, in quanto gli ostacoli vicini sono proprio i più importanti. In tal caso, anche se non più percepiti, consideriamo ancora valido qualunque ostacolo si trovi ad una distanza inferiore a una certa soglia scelta come parametro.

6.2.1.1 Esperimento 1: percorso semplice

Il primo esperimento è il più semplice, e ricalca il primo effettuato in simulazione con *Player/Stage*. In tal caso, come era stato spiegato in precedenza, siamo costretti ad utilizzare una distanza di sicurezza dagli ostacoli maggiore rispetto alla simulazione, per via del fatto che né l'odometria, né il sensore sono ideali.

6.2.1.2 Esperimento 2: uso del sonar al posto del laser

Abbiamo deciso di sperimentare anche il comportamento del sonar al posto del *laser*. Le considerazioni che ne sono conseguite sono che sebbene il primo dia letture meno instabili del secondo, il dettaglio che il *laser* riesce a dare, soprattutto riguardo all'ambiente vicino, è di fondamentale importanza per evitare ostacoli, soprattutto se questi sono in movimento.

6.2.1.3 Esperimento 3: navigazione in ambiente semi-sconosciuto

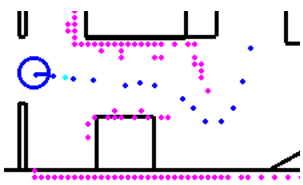


Figura 6.6: una regione della mappa con ostacoli non previsti

La regione della mappa davanti al robot in figura 6.6 contiene degli ostacoli non previsti dalla mappa statica. In questo caso il pianificatore di percorso, considerando anche tali ostacoli presenti nella mappa dinamica, deforma la traiettoria per evitarli.

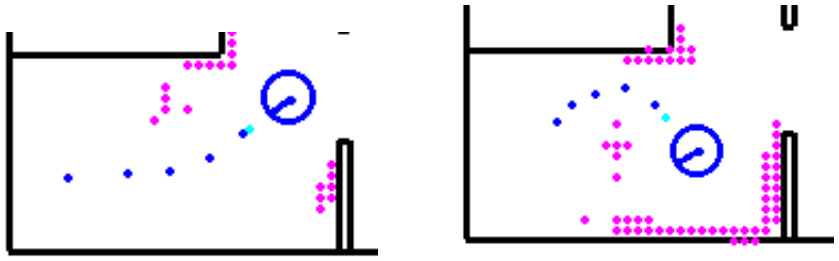


Figura 6.7: ostacolo (persona) in movimento

6.2.1.4 Esperimento 4: ostacoli in movimento

La caratteristica peculiare dell'ambiente *RoboCare* è la possibilità della presenza di ostacoli in movimento (persone). L'esperimento in questo caso consiste nel far muovere l'ostacolo in modo che intralci il percorso del robot, per osservarne il comportamento. Osserviamo in questo caso (figura 6.7) la deformazione dell'*elastic stick* per evitare l'ostacolo e il successivo cambiamento di cammino topologico dovuto al fatto che la presenza della persona in movimento ha reso il percorso precedente non sicuro.

6.2.2 Arena per la *RoboCupRescue*

La *RoboCupRescue* è una competizione internazionale il cui scopo è quello di incoraggiare lo sviluppo e la ricerca in quei campi della tecnologia utilizzati nella ricerca e nel salvataggio di esseri umani in strutture che hanno subito gli effetti di un terremoto, edifici in fiamme, incidenti sotterranei e così via.

Lo sviluppo, in questo ambito, di robot in grado di agire in maniera autonoma ed esplorare tali ambienti, individuando le eventuali vittime, è una sfida interessante.

Attualmente, durante le competizioni, vengono preparate delle "arene" che riproducono l'interno di edifici colpiti da terremoti o incendi, che i robot, teleoperati o autonomi, devono esplorare. In tale ambito il *path-planning* non deve aver a che fare con ostacoli in movimento (a meno che

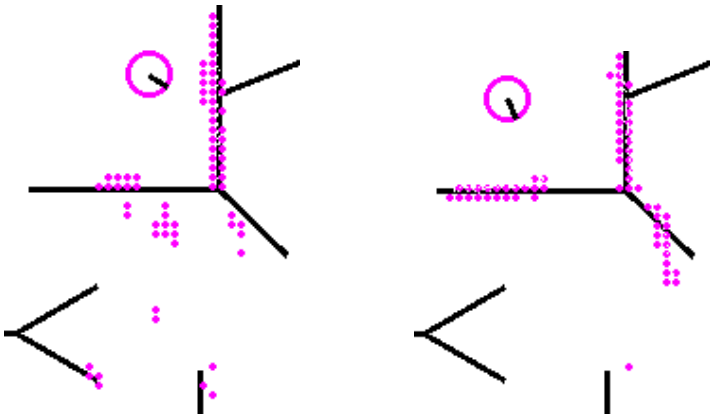


Figura 6.8: problemi di rifrazione e riflessione con il laser

non siano presenti altri agenti robotici), ma con una mappa che viene costruita man mano che prosegue l'esplorazione.

Abbiamo allestito all'Istituto Superiore Antincendio di Roma una "arena" simile a quelle utilizzate nelle competizioni *RoboCup Rescue*, per eseguire gli esperimenti. In questo caso il robot è un Pioneer 3-AT, sempre della *ActivMedia Robotics*, anche in questo caso governato da un *laptop* collegato anche ad un *laser scanner SICK*, più potente e preciso di quello visto nel paragrafo precedente. Il Pioneer 3-AT è provvisto di 4 ruote motrici e di un controllo che gli permette di ruotare su se stesso consentendo traiettorie con tratti a raggio di curvatura nullo.

Lo *scanner laser SICK* è notevolmente più preciso sulle lunghe distanze rispetto al modello *Hokuyo* descritto nel paragrafo precedente. Questo ci permette di poter aumentare la distanza utile a cui consentiamo le letture degli ostacoli, che, a sua volta, ci permette di avere una descrizione dinamica dell'ambiente più ampia.

6.2.2.1 Esperimento 1: percorso semplice

Nel primo esperimento il robot segue un percorso piuttosto semplice. In questa situazione bisogna notare soprattutto che la presenza di pannelli di materiale trasparente nell'arena provoca delle letture alterate del *laser*, al contrario di quanto ci si poteva intuitivamente aspettare. Gli ostacoli oltre questi pannelli vengono visti più vicini di quanto sono realmente, di

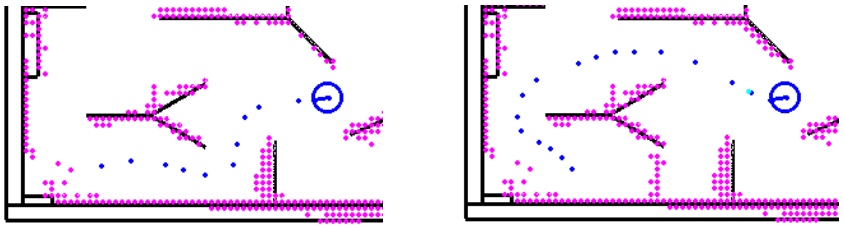


Figura 6.9: modifica della topologia dell'ambiente

un fattore che probabilmente dipende dall'angolo di incidenza del *laser* sul pannello stesso; in altre posizioni il *laser* viene addirittura riflettuto (figura 6.8). Questi *input* errati riguardo alla mappa dell'ambiente creano problemi soprattutto al modulo di localizzazione. Per questo motivo è stato deciso di fornire in ingresso a questo modulo i dati provenienti dall'anello di sonar di cui Pioneer 3-AT è dotato.

Il *path-planner*, invece, è risultato funzionare meglio con il laser. Nonostante le letture errate oltre il pannello trasparente, infatti, le informazioni per noi più importanti sono quelle relative agli ostacoli vicini al robot, per poterli evitare. La topologia errata dovuta al problema di rifrazione viene comunque risolta (cioè viene aggiornata la mappa dinamica) una volta che il robot si trova in una posizione da cui riesce a vedere le zone alterate senza pannelli frapposti. Questo è possibile grazie al comportamento che è stato definito in situazioni simili: il robot segue *comunque* la traiettoria non sicura, fino al tratto critico.

6.2.2.2 Esperimento 2: ostacoli in movimento

La presenza di ostacoli in movimento era già stata sperimentata in ambiente *RoboCare*. Tuttavia la maggiore affidabilità del sensore, in questo caso, porta ovviamente a risultati migliori. All'interno dell'arena abbiamo anche la possibilità di vedere come la DPTM si modifica per riflettere i cambiamenti di topologia dell'ambiente. In figura 6.9 è possibile vedere che il percorso iniziale del robot viene chiuso mediante un pannello mobile; a questo punto anche l'arco della DPTM viene eliminato e viene richiesto un nuovo percorso topologico su cui viene inizializzato il nuovo *elastic stick*.

Figura 6.10: caso di mappa statica inesistente

6.2.2.3 Esperimento 3: mappa differente da quella attesa

Se nell'arena vengono effettuati dei piccoli cambiamenti, cioè spostati di 20-30 cm alcuni pannelli, le prestazioni del robot rimangono pressoché le stesse. Gli ostacoli che vengono individuati in posizioni diverse da quelle previste vengono considerati come ostacoli *aggiuntivi*, perché, come abbiamo detto, la mappa statica non può essere modificata perché potrebbe contenere informazioni non percepibili con i sensori.

6.2.2.4 Esperimento 4: mappa statica inesistente

Abbiamo infine provato a far procedere il robot utilizzando un *path-planner* con una mappa statica vuota, o, meglio, contenente unicamente i muri dell'edificio. In questo caso è stato necessario disattivare il modulo di localizzazione, poiché esso si basa proprio sul confronto tra le letture dei sensori e la mappa statica dell'ambiente e abbiamo dovuto affidarci unicamente all'odometria.

Al robot, girato in modo che il laser non sia rivolto verso l'interno dell'arena, viene richiesto di eseguire un percorso verso un angolo della sala. Notiamo che non appena il laser comincia a leggere la presenza degli ostacoli, il percorso si modifica di conseguenza e continua questa deformazione anche durante il moto del robot.

Poiché l'odometria è imperfetta, il robot non è stato in grado di raggiungere l'obiettivo, ma, come è possibile vedere dalla figura 6.10, è stato in grado di raggiungere una posizione vicina ad esso, superando i pannelli iniziali.

Sebbene questa non sia probabilmente la migliore strategia di esplorazione, l'esperimento è stato importante per osservare il comportamento degli algoritmi presentati in di un ambiente la cui conoscenza si va via via formando mentre il robot si muove all'interno di esso.

Capitolo 7

Conclusioni e sviluppi futuri

7.1 Conclusioni

In questa tesi è stato progettato, sviluppato e sperimentato con successo un metodo di pianificazione dei percorsi per ambienti solo parzialmente conosciuti o con ostacoli di geometria incerta. Il metodo è risultato essere robusto alle modificazioni anche rapide della topologia dell'ambiente e a piccole imprecisioni che riguardano gli *input*, sia sulla posizione degli ostacoli che su quella del robot stesso. Come mostrato dagli ultimi esperimenti, esso presenta un comportamento adeguato anche qualora le informazioni iniziali sulla mappa siano inesistenti.

Il percorso viene generato ed eventualmente modificato in tempo reale, permettendo al robot di reagire prontamente ad eventuali modifiche dell'ambiente circostante.

Il metodo di pianificazione sviluppato è costituito di due livelli. Nel primo livello, quello *globale* viene costruita una mappa topologica dell'ambiente, grazie al quale è possibile prendere decisioni di pianificazione di percorso. Tale mappa è dinamica e si modifica quando avvengono cambiamenti topologici nell'ambiente. Al livello successivo, *locale*, le informazioni della mappa topologica vengono utilizzate per costruire un percorso eseguibile da un robot mobile, che risponda alle particolari caratteristiche richieste per una navigazione sicura e veloce.

Le *dynamic probabilistic topological map*, rispetto alle PRM, su cui si

basa, presenta delle importanti differenze:

1. le DPTM sono dinamiche, cioè sono costruite in modo da modificare la propria struttura qualora sopravvengano o vengano individuate delle variazioni topologiche, così da poter generare sempre un percorso ammissibile;

2. proprio per questo motivo il nostro algoritmo non necessita di una terminazione e, raggiunta una situazione di equilibrio, non rende la *roadmap* più complessa; le PRM, al contrario, se non fermate continuano ad aggiungere nodi e archi alla rete, anche se questi non portano informazioni necessarie;

3. il numero di nodi è nell'ordine del 5% rispetto alle PRM, a parità di informazione sull'ambiente, quello degli archi nell'ordine dell'1%; questo si riflette in tempi più rapidi per il calcolo dei percorsi all'interno del grafo ed è una condizione importante proprio per via della dinamicità della rete;

Rispetto all'altro algoritmo da cui prende spunto, il *growing neural gas*, la DPTM

1. non inserisce nodi soltanto quando il numero di iterazioni è multiplo di un certo parametro, bensì se e soltanto se essi vengano considerati utili a descrivere topologicamente l'ambiente;

2. in essa non esiste un meccanismo di "invecchiamento" degli archi per evitare che vengano considerate correlati due nodi che in realtà non lo sono; nella DPTM questa informazione è aggiornata in tempo reale;

3. ha una densità di nodi variabile, all'interno dello spazio di lavoro, posizionando più nodi ove sia necessario; al contrario il GNG, quando termina le sue iterazioni, ha una densità costante di nodi in tutto W.

D'altra parte gli *elastic stick* risolvono alcuni problemi riscontrati nelle applicazioni dell'algoritmo *elastic band* (quali, ad esempio, gli oscillamenti delle bolle) e vi aggiunge la caratteristica di aumentare, per quanto possibile, il raggio di curvatura dei percorsi, affinché siano più agevoli da seguire da parte di un robot mobile non olonomo.

7.2 Possibili miglioramenti e altri campi di applicazione

Per quanto riguarda le DPTM, il nostro obiettivo principale era quello di avere un algoritmo che costruisse una rete il più possibile *completa*. Sebbene tale rete risulti in gran parte dei casi molto semplice, esistono ancora delle situazioni in cui vengono aggiunti nodi ridondanti alla descrizione topologica dell'ambiente. Sebbene questo non comporti grandi perdite di efficienza, una rete topologica minimale sarebbe se non altro più semplice da gestire.

Una ulteriore estensione dell'algoritmo potrebbe essere quella di avere una rete topologica *gerarchica*. Sebbene questo non abbia quasi alcuna utilità negli ambienti da noi studiati, potrebbe averne per luoghi più ampi, come piani di edifici o intere città. La rete dovrebbe chiaramente essere in grado di decidere il livello di dettaglio autonomamente.

Il più importante miglioramento agli elastic stick è quello, già citato, riguardante un raggio di curvatura minimo preimpostabile come parametro. In questo modo, infatti, il metodo sarebbe più generalmente applicabile anche a robot e veicoli che non possono eseguire traiettorie il cui raggio di curvatura è nullo. Sarebbe anche opportuno un migliore accoppiamento col modulo di controllo, in modo tale che l'*elastic stick* generi percorsi o traiettorie che siano immediatamente eseguibili dal modulo di controllo. In questo modo il modulo di pianificazione locale avrebbe un controllo diretto sulla traiettoria che il *controller* effettivamente eseguirà.

Potrebbe essere interessante applicare i metodi qui descritti in ambiente *multirobot*, in cui alcune problematiche potrebbero essere risolte in maniera piuttosto semplice. Ad esempio per realizzare un *following* tra

due robot sarebbe sufficiente mantenere un capo dell'*elastic stick* coincidente con la posizione del robot da seguire, aggiornandolo quindi ad ogni suo movimento. Il percorso in questo modo sarebbe facilmente aggiornabile e, per le caratteristiche degli *elastic stick*, sempre sicuro da seguire.

Sempre in questo ambito, un'altra possibilità potrebbe essere quella di applicare questi metodi alla situazione di un passaggio stretto che più robot devono utilizzare. Tale passaggio potrebbe essere dichiarato "non sicuro", grazie alle DPTM che offrono questa possibilità, permettendo così ad uno dei due robot di richiedere un altro percorso topologico da seguire (qualora questo esista) per raggiungere il medesimo obiettivo.

Bibliografia

[Oriolo93] Pianificazione del Moto (1993) [G. Oriolo]

[ESQuiKha93] Elastic Bands - Connecting Path Planning and Control (1993) [S. Quinlan, O. Khatib]

[ESBroKha00] Integrated Planning And Execution: Elastic Strips (2000) [O. Brock, O. Khatib]

[ESWanGra01] Non-Holonomic Navigation System of a Walking-Aid Robot [J.M.H. Wandosell, B. Graf] (2001)

[PRMKavLat98] Probabilistic Roadmaps for Robot Path Planning (1998) [L. E. Kavraki, J.-C. Latombe]

[PRMBridge] The Bridge Test For Sampling Narrow Passages with Probabilistic Roadmap Planners [D. Hsu, T. Jiang, J. Reif, Z. Sun]

[PRMWilAmaSti] MAPRM - A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space [S. A. Wilmarth, N. M. Amato, P. F. Stiller]

[PRM-Q] Quasi-Randomized Path Planning [M. S. Branicky, S. M. LaValle, K. Olson, L. Yang] (2001)

[PRMSimLauNis] Visibility-Based Probabilistic Roadmaps for Motion Planning [T. Simeon, J-P. Laumond, C. Nissoux] (1999)

[RRTKufLV] RRT-Connect - An Efficient Approach to Single-Query Path Planning [J. J. Kuffner, S. M. LaValle] (2000)

[SOMKoh97] Self-Organizating Maps, New York : Springer-Verlag, 1997 [T. Kohonen]

[SOMWeb] Self-Organizing Maps [T. Germano] (http://davis.wpi.edu/~matt/courses/soms/)

[SOMVleKokOve93] Motion Planning Using A Colored Kohonen Network (1993) [J. M. Vleugels, J. N. Kok, M. H. Overmars]

[NGMarShu91] A "Neural-Gas" Network Learns Topologies (1991) [T. Martinetz, K. Shulten]

[GNGFri95] A Growing Neural Gas Network Learns Topologies (1995) [B. Fritzke]

[TesiFar] Tecniche di pianificazione delle traiettorie in ambiente dinamico (2001) [A. Farinelli] (Tesi di Laurea)

[GNGDemoWeb] DemoGNG at Institut fur Neuroinformatik (http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/GNG.html)

[WMRCtrl] WMR Control via Dynamic Feedback Linearization: Design, Implementation and Experimental Validation (2002) [G. Oriolo, A. De Luca, M. Vendittelli]

[DictWeb] TheFreeDictionary.com (http://encyclopedia.thefreedictionary.com/topology)

[AlgTopWeb] Algebraic Topology [A. Brouwer] (http://www.win.tue.nl/~aeb/at/algtop.html)

[MCWeb] The Basics of Monte Carlo Simulations (http://www.cs.cornell.edu/jwoller/samples/montecarlo/default.html)

[SCOFoe] Sistemi che osservano, Casa Editrice Astrolabio, 1987 [H. von Foerster]

Indice analitico

Elenco delle figure

”Infinite cose da fare. . . e così poco tempo”

Questa tesi è stata compilata utilizzando il software LaTeX2e