UNIVERSITÀ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria dell'Automazione

Corso di Progettazione di Sistemi di Controllo
a.a. 2005/06

Pianificazione del Moto
e
Controllo di un Uniciclo

Luigi Ricciato: 530208/IAM - ricciato@dei.unipd.it
Emanuele Siego: 530385/IAM - emanuele.siego@poste.it
Daniele Tamino: 530387/IAM - datamino@dei.unipd.it

Padova, 11 luglio 2006

Indice

1 Introduzione

I robot mobili su ruota (WMR) hanno conosciuto in anni recenti un notevole sviluppo grazie agli sforzi della ricerca nel campo dei sistemi non lineari e alle emergenti tecnologie a basso costo. In ambito accademico ritroviamo numerosi esempi relativi, in particolare, al movimento autonomo in uno spazio ben definito. Oltre al naturale interesse teorico, il problema della pianificazione del percorso e del controllo di WMR ha attratto grande interesse in vista delle possibili applicazioni nel settore dell'automotive, della sorveglianza e dell'esplorazione di ambienti sfavorevoli all'uomo. Tali problemi possono essere affrontati a vari livelli di generalizzazione e con opportune ipotesi più o meno semplificative.

In questo progetto ci occuperemo essenzialmente del movimento di un sistema su due ruote non sterzanti tra due diverse configurazioni assegnate, in un ambiente con ostacoli ma comunque noto. Una strategia comune di risoluzione di problemi complessi come quello descritto si basa sulla strutturazione a più livelli, due nel nostro caso. Il primo livello è predisposto per la generazione di una o più possibili traiettorie ammissibili che collegano i punti iniziale e finale assegnati. In questa fase abbiamo scelto di trascurare, per questioni di semplicità realizzativa, ogni vincolo di non-olonomicità del moto; naturalmente si possono prendere in considerazione numerose altre alternative. Il secondo livello, invece, si occupa del tracking della traiettoria calcolata precedentemente. L'algoritmo di controllo utilizzato permette l'inseguimento di traiettorie qualsiasi, purché opportunamente regolari, con errore asintotico nullo. Tuttavia se la traiettoria presenta forti irregolarità (punti angolosi, cuspidi, ...) è possibile che il movimento, per un transitorio più o meno lungo, non sia limitato ad un intorno della stessa, con conseguente pericolo di scontro con gli ostacoli. Una soluzione a tale inconveniente è auspicabile modificando il primo livello con l'introduzione dei vincoli sulle velocità istantanee.

2 Pianificazione del moto

2.1 Premesse

Algoritmi di pianificazione del moto (*motion planning*) possono trovare applicazione in moltissimi campi come *Intelligenza Artificiale*, *Teoria del Controllo* ma soprattutto, come nel caso di nostro particolare interesse, nel campo della *Robotica* dove molto spesso si richiede, ad esempio, di dover muovere un sitema meccanico da una configurazione (spaziale) ad un'altra, desiderata e ben specificata, senza collidere con l'ambiente circostante.

Originariamente, in tali problemi, dinamica e i vincoli differenziali venivano ignorati, a causa soprattutto delle limitazioni di calcolo, mentre l'attenzione veniva puntata maggiormente sulla semplice ricerca di quale traslazione e rotazione applicare al sistema, per essere in grado di trovare almeno una soluzione che per quanto semplice fosse, avrebbe potuto risolvere il problema. Ora, però, i problemi da risolvere sono diventati molteplici e ben più complicati, per la presenza di incertezze, vincoli differenziali, presenza di corpi multipli, possibilmente aventi anche una loro dinamica, e desiderio di ottimalità della soluzione cercata.

Vediamo, quindi, alcuni dei motivi per i quali diventa importante trovare una soluzione a tali problematiche, talvolta abbastanza complesse.

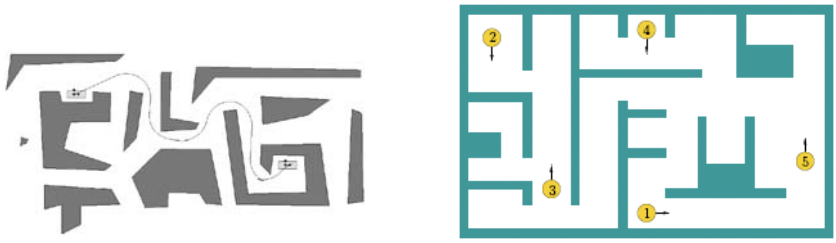
2.2 Esempi applicativi e motivanti

Come si potrebbe fin da subito intuire, limitando la nostra attenzione al solo campo della robotica, le applicazioni di un algoritmo di *motion planning* sono molteplici. In effetti, il problema di ricerca di percorsi in assenza di collisioni è di importanza centrale nelle applicazioni su robot mobili. Solo attraverso tali algoritmi è possibile risolvere problemi, altrimenti "difficili" per un umano, in maniera efficiente e robusta. Descriviamo qui brevemente alcuni esempi che motivano quindi la buona riuscita di tali algoritmi.

Navigazione di robot mobili Il primo compito forse più comune richiesto a un robot mobile è quello di riuscire a muoversi in un ambiente chiuso, come mostrato in fig. 1. Le finalità possono essere diverse: si potrebbe, ad esempio, volere che il robot riuscisse ad arrivare in una certa posizione tenendo conto della planimetria della stanza oppure che più robot navigassero insieme senza collidere non solo con i muri ma anche tra di loro. In tal caso, si presentano ulteriori complicazioni: Come possono comunicare i robot tra di loro? Come si devono integrare le loro informazioni? Il controllo deve essere centralizzato o distribuito? Come si possono evitare le collisioni? E poi, devono avere compiti diversi oppure devono in qualche modo collaborare tra loro? Si può capire come le risposte a queste domande non

siano per niente banali.

Un'altro compito che si potrebbe richiedere a un robot mobile potrebbe essere quello



(a) (b)

Figura 1: (a) Esempio di agente che si sposta da una configurazione ad un'altra senza collidere con i muri. (b) Una serie di agenti che cercano di muoversi in un ambiente chiuso evitando di collidere con i muri e tra di loro.

di dover costruire una buona mappa dell'ambiente in cui si trova, determinando contemporaneamente una stima abbastanza precisa della sua posizione e del suo orientamento. Per capire di cosa si tratta, si guardi la fig. 2. Si intuisce subito come il problema sia estremamente più difficile a motivo di incertezze dovute alla mancata conoscenza dell'ambiente in cui ci si muove.

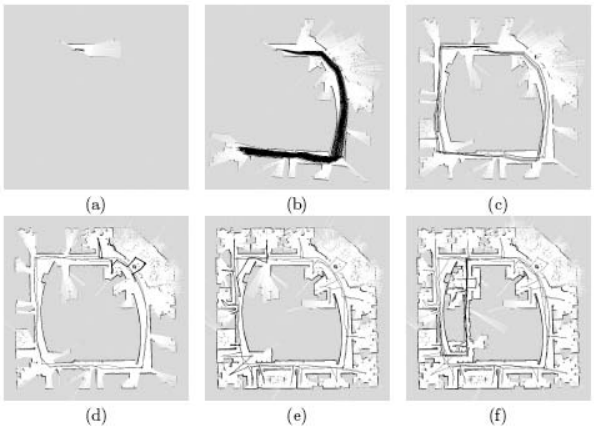


Figura 2: Un robot mobile che costruisce una mappa dell'ambiente in cui si muove, determinando al tempo stesso la sua posizione all'interno della stanza.

Navigazione autonoma di autoveicoli I problemi sopra considerati non tengono in conto la possibile presenza di vincoli differenziali dovuti alla natura fisica del

sistema. Nel caso, ad esempio, di pianificazione di agenti mobili su ruote, a volte non si può prescindere dai vincoli differenziali che legano la direzione istantanea della velocità (vincoli anolonomi), dato che l'effettiva soluzione dipende da essi. Si consideri a tal proposito il problema mostrato in fig. 3. Le auto sono soggette a

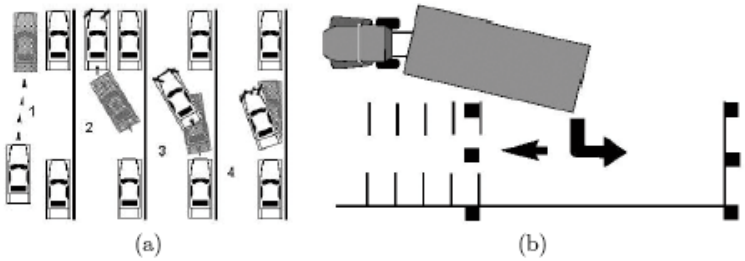


Figura 3: Esempi di parcheggi: (a) Parcheggio di un'automobile. (b) Parcheggio di un camion con rimorchio.

vincoli anolonomi dovuti al fatto che le ruote possono ruotare solo nella direzione nella quale sono rivolte. In fig. 4 sono poi mostrati esempi di navigazione autonoma

(a) (b)

Figura 4: Esempi di navigazione autonoma: (a) Una delle auto partecipanti al DARPA Grand Challenge per la navigazione autonoma. (b) Il *Mars Pathfinder* durante una missione di esplorazione su Marte.

eseguita perfino in ambiente ostile. Oltre alla presenza di incertezze e di vincoli, ci si presenta di fronte il problema di dover sueperare le asperità del terreno e dell'ambiente in cui si deve far operare il mezzo.

2.3 Introduzione all'algoritmo di *Path Planning*

Il problema che consideriamo è quello di trovare in maniera automatica ed efficiente un percorso sicuro per il robot mobile su ruote *Khepera*, libero di muoversi

in un ambiente 2-D contenente ostacoli. L'algoritmo, come vedremo, assume l'esistenza di una mappa di tutto l'ambiente, binaria, i cui "pixel" possono essere liberi oppure occupati. Lo scopo che ci prefiggiamo è quello di trovare "un percorso ottimo in un tempo di calcolo minimo", il ché sarà possibile, come vedremo, grazie a una particolare struttura dati che, usata per rappresentare lo spazio, permetterà di trovare un grafo di percorso semplice ma accurato. Per iniziare, facciamo le ipotesi

A. di avere a disposizione l'intera mappa dell'ambiente;

B. che il robot si muova in un ambiente statico, cioé gli ostacoli siano fissi;

C. che il robot non sia soggetto ai vincoli anolonomi;

D. che gli oggetti nell'ambiente non abbiano necessariamente forme standard.

Molti approcci per il *path planning* sono stati studiati in letteratura. Essi possono essere raggruppati in metodi *locali* e metodi *globali*. I metodi locali non cercano di risolvere il problema nella sua completa generalità, ma usano soltanto le informazioni (limitate) disponibili al robot per determinare il successivo comando da applicare per il suo movimento. I metodi globali invece sono tutti quelli che assumono una conoscenza completa circa l'ambiente in cui si svolge il moto.

In ogni caso è essenziale che la pianificazione sia efficiente dal punto di vista del tempo di calcolo, sopratutto per problemi *online*, dove i percorsi sono pianificati ed eseguiti in tempo reale, per pianificazioni fatte *on board*, dove le risorse di calcolo sono limitate, e per pianificazioni in ambienti dinamici, dove sono richieste frequenti ripianificazioni a motivo del fatto che un'unico percorso statico, anche se dettagliato, ha basse probabilità di essere eseguito senza complicazioni poiché gli ostacoli si muovono inaspettatamente man mano che il robot avanza lungo il pecorso.

Si assuma che il robot sia in qualche punto dell'area libera della mappa e che si voglia pianificare un percorso fino ad un qualche altro punto della mappa. Si assuma inoltre che il robot abbia una sezione trasversale circolare e che sia rappresentato da un punto corrispondente al centro della sua sezione trasversale. Una volta digitalizzata la mappa, è nostro desiderio sapere se il percorso di navigazione può essere eseguito prima che la mappa diventi obsoleta; infatti essa può cambiare nel tempo a causa della presenza di altri agenti mobili. Inoltre, non ha senso trovare un percorso a distanza minima se il tempo speso in tale ricerca risulta essere eccessivamente lungo.

Per i motivi sopra elencati, si cercano algoritmi che portino ad ottenere una rappresentazione dell'ambiente che lasci, sì, inalterata la ricchezza di dettagli della mappa originale ma che al contempo permetta di risolvere in modo più veloce il problema della ricerca di un percorso abilitato che porti il robot dalla posizione iniziale a quella desiderata.

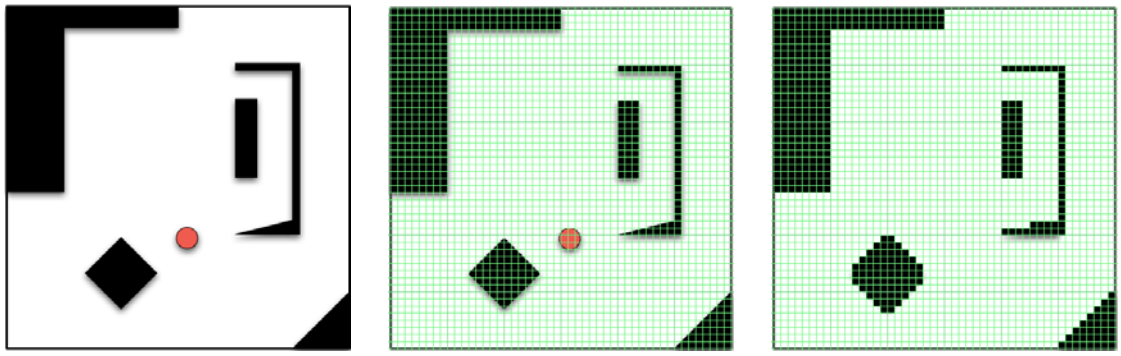
Diciamo quindi che il problema della navigazione può essere scomposto principalmente in tre sottoproblemi:

1. Mappatura e modellizzazione dell'ambiente.
2. Pianificazione del percorso.
3. Attraversamento del percorso pianificato senza collidere con gli ostacoli.

Descriviamo separatamente qui di seguito le prime due fasi, essendo la terza ovvia conseguenza delle prime due.

2.4 Mappatura e modellizzazione dell'ambiente

Prima d'ora, rappresentazioni digitali dello spazio non erano molto usate poiché richiedevano grandi quantitativi di memoria. A motivo però della drastica diminuzione dei costi delle memorie e dei notevoli sviluppi nel campo delle immagini rasterizzate, l'uso di immagini digitali comporta grandi vantaggi. Grazie ad esse, infatti, è possibile rappresentare in modo molto acccurato oggetti di forma qualsiasi. Inoltre, ogni pixel della mappa può essere visto come libero o occupato (naturalmente con appropriati fattori di confidenza). Inoltre, aree sconosciute possono essere "etichettate" come occupate fintanto che esplorazioni successive non indichino il contrario.



(a) (b) (c)

Figura 5: Esempio di digitalizzazione di una mappa: (a) Mappa originale. (b) Griglia usata per la digitalizzazione, in cui ogni cella corrisponde a un pixel. (c) Immagine digitale della mappa originale (pixel neri indicano spazio occupato, pixel bianchi indicano spazio libero).

Rifacendoci all'esempio di fig. 5, la mappa iniziale viene inizialmente suddivisa in tante piccole celle quadrate, a seconda della risoluzione desiderata, per ottenere

una griglia che è tanto più fitta quanto più piccola è la dimensione delle sue celle. Il risultato di questa operazione è mostrato in fig. 5(c). Si noti che questa immagine può essere ora memorizzata facilmente attraverso un array binario in cui ogni elemento può avere come valore 1 se il "pixel" è occupato (regioni dello spazio nere), 0 altrimenti (regioni dello spazio libere).

Puntualizziamo il fatto che la dimensione della singola cella deve essere ben valutata sulla base di due fattori principali. La sua dimensione è limitata inferiormente dalla sezione del robot, dato che celle molto più piccole di tale valore potrebbero portare il robot a urtare contro un ostacolo nel caso in cui il percorso pianificato passi abbastanza vicino al bordo. D'altra parte la dimensione delle celle è limitata superiormente dall'accuratezza di rappresentazione dell'ambiente, dato che risoluzioni troppo basse potrebbero portare alla scomparsa di percorsi (stretti) altrimenti abilitati.

2.5 Path Planning basato su *Quadtree*

Un *quadtree* è una scomposizione ricorsiva di un'immagine bidimensionale in $2^i \times 2^i$ blocchi. Ogni nodo dell'albero rappresenta una regione quadrata dell'immagine di dimensioni $2^j \times 2^j$. Un nodo bianco rappresenta una regione libera da ostacoli, un nodo nero una contenente interamente ostacoli, un nodo grigio, invece, una regione che contiene sia spazio libero che ostacoli. Il vantaggio dei quadtree è quello di riuscire a rappresentare grandi aree (libere o di ostacoli) con pochi grandi blocchi, permettendo così di diminuire il tempo di calcolo riservato alla ricerca del percorso nel grafo.

Data quindi una rappresentazione rasterizzata dell'ambiente in cui si muove il robot, ottenuta ad esempio dalla procedura descritta nella sezione 2.4, essa viene preprocessata in due fasi:

1. l'immagine rasterizzata viene dapprima convertita in un *quadtree* tramite un algoritmo efficiente come quello descritto in [7] con complessità computazionale $O(n)$, dove n è il numero di pixel dell'immagine convertita;

2. si calcola, poi, la distanza, secondo un'opportuna norma[1], tra il centro di ogni blocco di spazio libero e il bordo del blocco di ostacoli più vicino, attraverso un algoritmo trattato in dettaglio da [8].

Partendo ad esempio dalla mappa di fig. 6 e processando ricorsivamente i blocchi secondo le direzioni (NW,NE,SW,SE), si ottiene il risultato mostrato in fig. 7. In particolare, dopo il passo 1), si ottiene l'albero mostrato in fig. 7(a) che rappresenta in forma compatta la mappa di partenza. Il risultato del passo 2) è invece mostrato in fig. 7(b).

[1]Si dimostra che la norma più adatta per il problema in esame è la norma infinito.

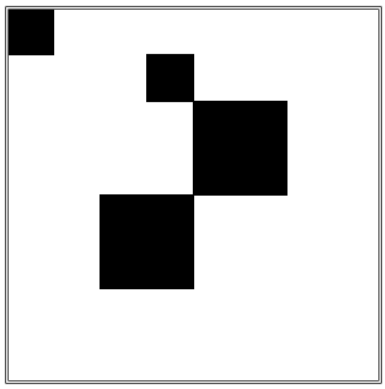
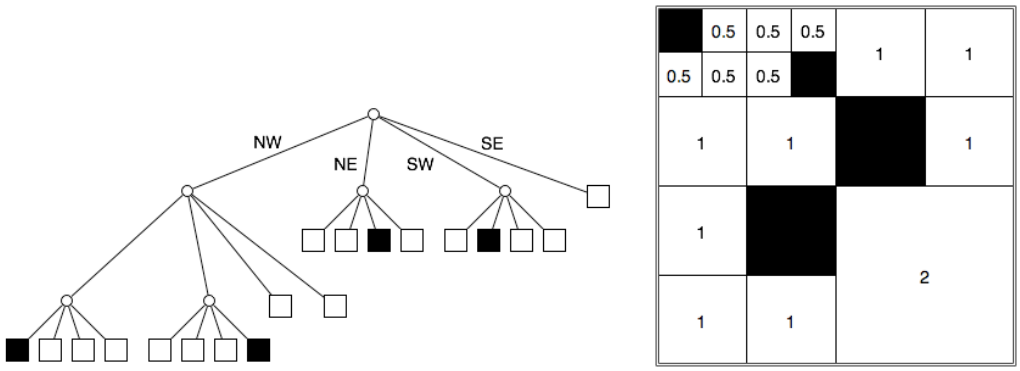


Figura 6: Un esempio di mappa.



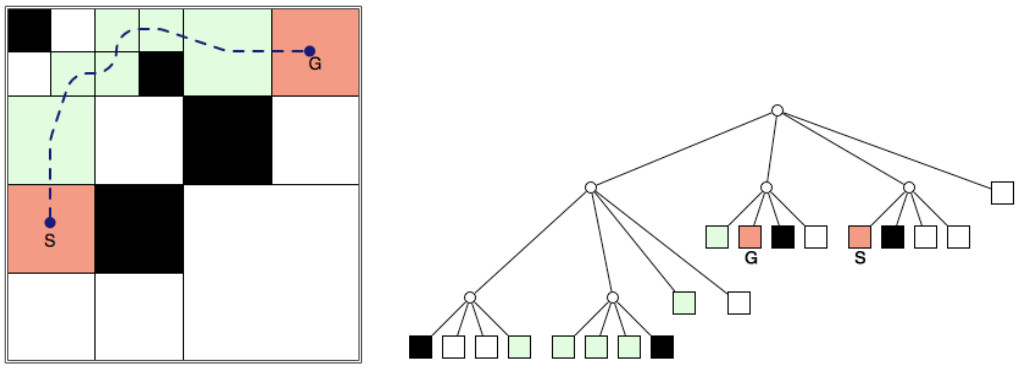
(a) (b)

Figura 7: (a) *Quadtree* rappresentante la mappa di fig. 6. (b) Distanza dei blocchi bianchi dai bordi dei più vicini blocchi neri.

Terminata la fase di *preprocessing* e determinati i punti iniziale (S) e finale (G), è possibile passare all'effettiva ricerca del percorso: si individuano i nodi foglia dell'albero corrispondenti alle regioni contenenti i punti dati e si cerca un percorso di costo minimo tra S e G che usi i nodi foglia bianchi dell'albero con un algoritmo di ricerca detto A^* e descritto in [5].

Il risultato dell'algoritmo applicato all'immagine di fig. 7(a) è mostrato in fig. 8, dove sono evidenziate le regioni contenenti i punti iniziale e finale e quelle attraversate dal percorso trovato.

Come abbiamo visto, l'approccio basato su *Quadtree* è molto semplice, robusto ed efficiente. Ha comunque due grossi svantaggi. Il primo è quello di richiedere molte piccole celle per rappresentare bene gli ostacoli poichè la posizione e la dimensione delle celle non dipendono dall'oggetto (fig. 9). Oltre a questo, costruendo il quadtree



(a) (b)

Figura 8: Risultato della pianificazione di un percorso da S a G su una rappresentazione a *Quadtree*. I nodi neri corrispondono agli ostacoli, mentre le regioni colorate corrispondono a quelle attraversate dal percorso.

a partire da una griglia a bassa risoluzione, molti passaggi stretti tra gli ostacoli potrebbero non apparire. Di conseguenza, è richiesta una griglia di ingresso ad alta risoluzione, il ché può far capire subito come la ricerca del percorso minimo nel grafo diventi lenta a causa del numero elevato di nodi da considerare.

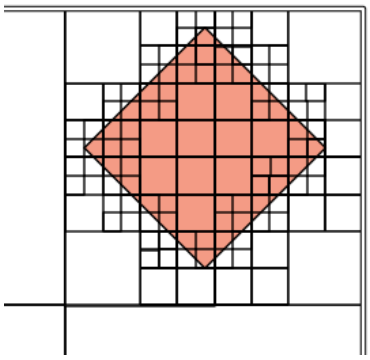
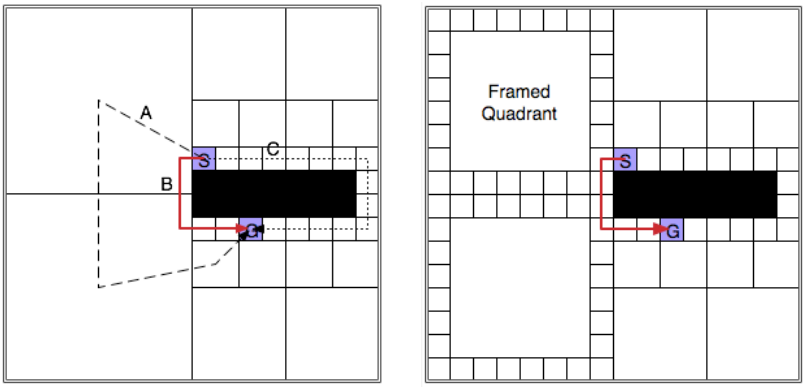


Figura 9: Per rappresentare bene la forma di alcuni ostacoli sono richieste un numero elevato di celle.

Il secondo limite sta invece nel fatto che molto spesso nello spazio libero si formano celle così grandi da portare l'algoritmo a trovare percorsi di lunghezza non minima. Poiché la posizione del nodo nel grafo corrisponde al centro della cella, il costo per attraversare la cella è lo stesso sia nel caso in cui il percorso passi per l'intera cella oppure che si possa limitare a sfiorarne un angolo. I quadtree sono perciò molto sensibili alla disposizione degli ostacoli e il fattore di errore è proporzionale

alla dimensione della cella. Un esempio di questo problema è mostrato in fig. 10(a)



(a) (b)

Figura 10: (a) Esempio di *Quadtree*. (b) Esempio di *Framed-Quadtree*. L'approccio convenzionale dei *Quadtree* potrebbero fallire nella ricerca del percorso ottimo.

che descrive la situazione in cui si vuole pianificare un percorso da S a G usando un *Quadtree*. Al posto del percorso B, che risulta essere il più corto, viene trovato il percorso C, anche perché il percorso A risulta avere costo più elevato per il fatto che attraversa blocchi di dimensione, e quindi anche di costo, maggiore. Per risolvere in parte questo problema, sono stati introdotti i cosiddetti *Framed-Quadtree*, una struttura dati in cui i bordi delle grandi celle sono racchiusi da un array di celle quadrate la cui dimensione è pari a quella più piccola ammissibile. In tal caso, come evidenziato in fig. 10(b), si riesce a trovare il percorso B come quello di costo minimo.

Per una descrizione più dettagliata dell'algoritmo e suoi possibili miglioramenti, si veda [5].

2.6 Path Planning basato su *maglie triangolari*

Visti gli svantaggi dei *Quadtree*, si è scelto di studiare un metodo di pianificazione alternativo che sia efficiente ma al tempo stesso accurato nella descrizione dell'ambiente. L'algoritmo si basa essenzialmente sulla triangolazione dello spazio sulla base di una griglia di ingresso ottenendo così una prima maglia di partenza. Tale maglia, essendo formata da molti triangli, viene esemplificata senza introdurre un elevato *errore geometrico* fino a ottenere una maglia più compatta. A questo punto viene estratto dalla maglia semplificata un grafo di percorso. Tale grafo, nonostante abbia una struttura molto semplice, riesce comunque a mantenere alto il livello di dettaglio, soprattutto dei bordi degli ostacoli. Vedremo infatti che la forma

degli ostacoli viene mantenuta anche dopo significative esemplificazioni.

L'algoritmo consiste essenzialmente di cinque passi principali che saranno discussi separatamente in dettaglio:

1. generazione della maglia di base tramite triangolazione dello spazio;
2. semplificazione della maglia di base;
3. generazione del grafo di percorso;
4. ricerca del percorso minimo usando l'algoritmo di Dijkstra;
5. *smoothing* del percorso.

Prima di addentrarci nella discussione dell'algoritmo, richiamiamo brevemente qui di seguito alcune definizioni di teoria dei grafi di cui faremo uso.

Un *grafo non orientato* è una coppia $G = (V, E)$ in cui V è l'insieme finito dei vertici ed E è una famiglia di coppie non ordinate di elementi di V chiamate lati. Indichiamo con $n := |V|$ ed $m := |E|$, rispettivamente il numero dei vertici e dei lati di G. Se indichiamo poi $V = \{1, \ldots, n\}$, i lati hanno la forma $[i, j] \equiv [j, i]$ con $i, j \in V$. Diciamo che due vertici sono adiacenti se esiste il lato che li collega, mentre diciamo che due lati sono adiacenti se hanno un vertice in comune. Il *grado* del vertice v, indicato con $d_G(v)$, è il numero di lati incidenti in quel vertice. Un cammino (*path*) di G è una sequenza di lati consecutivi $e_1, e_2, \ldots, e_k \in E$ del tipo: $e_1 = [v_1, v_2]$, $e_2 = [v_2, v_3]$, $\ldots$, $e_k = [v_k, v_{k+1}]$. Tale cammino collega quindi i vertici v_1 e v_{k+1}. Chiamiamo l'insieme di tutti i lati uscenti da un nodo v *star* del nodo v e la indichiamo con $\delta(v)$; quindi $\delta(v) = \{[i, j] \in E : i = v\}$.
Se il grafo G è *orientato*, alla famiglia dei lati viene sostituita quella degli *archi* (i, j) con $i, j \in V$ e $(i, j) \neq (j, i)$. Si dice che l'arco $(i, j) \in A$ esce dal vertice i ed entra nel vertice j.

I grafi, orientati e non, possono essere descritti da opportune matrici. Una struttura particolarmente semplice e utile per la nostra trattazione può essere quella che memorizza, per ogni vertice $v \in V$, la sua star $\delta(v)$. Si noti che in questo modo, per un grafo non orientato, ogni lato $[i, j]$ viene memorizzato due volte, dato che $[i, j] \in \delta(i) \cap \delta(j)$.

Per memorizzare i vertici della maglia è stata costruita una matrice la cui h-esima colonna è del tipo

$$[v \; x_v \; y_v \; z_v]'$$

dove in questo caso v è il numero del vertice[2], mentre le restanti componenti deno-

[2]Si noti che tale elemento del vettore colonna non è necessario ai fini della semplice memorizzazione del vertice, risulta però necessario nel nostro caso poiché non abbiamo un preciso ordinamento dei vertici

tano le coordinate spaziali del vertice, cosa che sarà spiegata meglio nella sezione successiva.

Per memorizzare i lati del grafo costruiamo due matrici. La prima è una matrice, chiamata `FROM-TO`, la cui h-esima colonna è del tipo

$$[\texttt{from} \quad \texttt{to}]'$$

dove `from` e `to` sono i numeri dei due vertici adiacenti al lato [`from`, `to`]. La seconda matrice, chiamata `FIRST`, ha le colonne del tipo

$$[v \quad \texttt{f}]'$$

e tiene conto della prima colonna `f` della matrice `FROM-TO` a partire dalla quale è memorizzata la star del vertice v.

2.6.1 Generazione della maglia di base

Durante questa fase, un ambiente 2-D viene trasformato in una maglia di base 3-D. In maniera analitica, una maglia M è descritta da una terna (V, E, F), dove V è l'insieme dei vertici, E è l'insieme dei lati, ed F è l'insieme dei triangoli. Per rappresentare i vertici e i lati facciamo uso di un grafo, memorizzato attraverso le matrici `FROM-TO` e `FIRST`, come già descritto nella sezione 2.6. Per rappresentare invece l'insieme F, utilizziamo semplicemente una matrice le cui colonne contengono i numeri dei tre vertici formanti i triangoli della maglia.

Più in specifico, la maglia viene realizzata sulla base della griglia di ingresso [3]. I nodi al centro delle celle della griglia vengono poi usati come vertici della maglia. I lati vengono aggiunti congiungendo i vertici secondo un criterio di adiacenza delle celle. Ogni vertice è quindi della forma $v = [v_v \, y_v \, z_v]'$, dove x_v e y_v sono le coordinate 2-D del vertice nello spazio 2-D, mentre la coordinata z_v vale 0 se il vertice si trova su uno spazio libero da ostacoli, vale $\alpha \in \mathbb{R}_+$ se si trova su uno spazio occupato da ostacoli.

Un esempio di ciò che si ottiene è mostrato in fig. 11(b).

2.6.2 Semplificazione della maglia di base

Il problema di trovare una maglia semplificata ottima è un problema di ottimizzazione geometrica che ha un tempo di calcolo non-polinomiale (NP-*hard*); perciò per la soluzione del problema usiamo delle tecniche euristiche, come quella del collassamento dei lati mostrata in fig. 12. Il lato $[v_1, v_2]$ viene collassato a un vertice *target* v, con la conseguente rimozione dei triangoli adiacenti al lato rimosso.

[3] Si guardi la sezione 2.4 per sapere come ottenere una griglia opportuna.

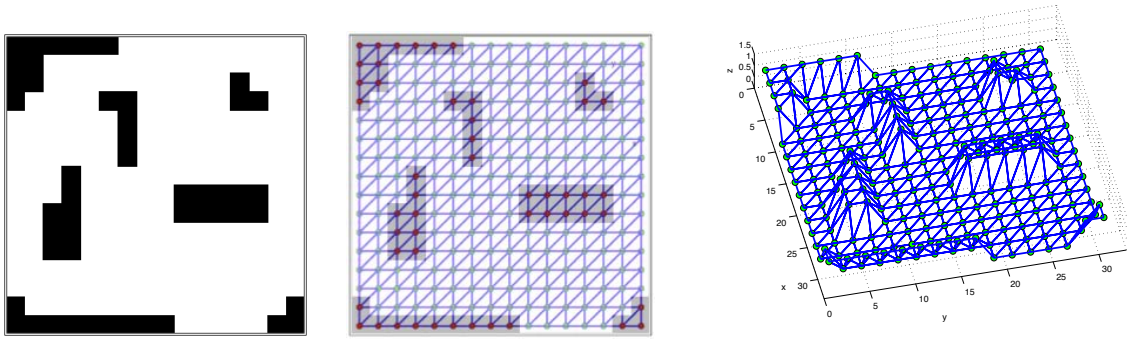


Figura 11: Generazione della maglia di base a partire dalla mappa. (a) Mappa dell'ambiente. (b) Vista 2D della maglia di base corrispondente (sono evidenziati in rosso i vertici situati sugli ostacoli). (c) Vista 3D della maglia di base.

Le operazioni di collassamento $ecol_i$ vengono realizzate iterativamente a partire dalla maglia originale M_0, fino ad ottenere, dopo n collassamenti, la maglia semplificata $\hat{M}$, come evidenziato dalla seguente

$$M_0 \xrightarrow{ecol_1} M_1 \xrightarrow{ecol_2} \cdots \xrightarrow{ecol_n} M_n = \hat{M}. \tag{1}$$

I lati della maglia M_i sono ordinati secondo il loro costo di collassamento e memorizzati in una coda prioritaria. Il costo per collassare il lato $[v_1, v_2]$ nel vertice v è definito come l'errore geometrico tra la *geometria locale* e la *geometria locale originale*. Spieghiamo la differenza tra queste due geometrie facendo un esempio. Supponiamo che la maglia a sinistra in fig. 12 sia la maglia originale. La *geometria locale originale* dei vertici v_1 e v_2 è formata rispettivamente dai triangoli f_{1-6} e $f_{3,4,7-9}$. La *geometria locale* dei vertici v_1 e v_2 è uguale alla geometria locale originale. La geometria locale del vertice v è invece formata dai triangoli $f_{1,2,5,6,7-9}$, mentre la sua *geometria locale originale* è f_{1-9}, cioé l'unione delle *geometrie locali originali* dei vertici v_1 e v_2.

L'ordine di collassamento dei lati è determinato in modo da minimizzare l'errore geometrico tra la maglia di partenza e quella di arrivo. Ad ogni passo viene rimosso dalla coda, e quindi collassato, il lato con più alta priorità (costo minore). La coda viene aggiornata e i lati vengono estratti iterativamente finchè non si raggiungono determinate condizioni di terminazione. L'errore geometrico è definito, in base a una metrica di errore quadratico, come la somma dei quadrati delle distanze dai vertici ai piani che contengono i triangoli della geometria locale originale del vertice, chiamati *piani del vertice* e indicati con P_v. L'errore quadratico del vertice v può

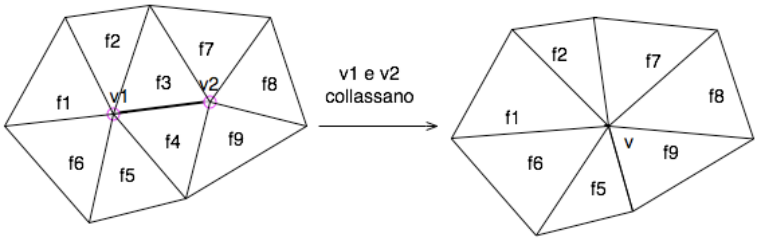


Figura 12: Il lato $[v_1, v_2]$ collassa nel vertice v; i triangoli f_3 ed f_4 vengono rimossi.

quindi essere scritto come

$$\Delta(v) = \sum_{p \in P_v} (p'v)^2 \tag{2}$$

dove $p = [a\ b\ c\ d]' \in P_v$ rappresenta il piano definito dall'equazione parametrica $ax + by + cz + d = 0$, con $a^2 + b^2 + c^2 + d^2 = 1$.

La (2) può essere riscritta anche come

$$\begin{aligned} \Delta(v) &= \sum_{p \in P_v} (p'v)(p'v) = \sum_{p \in P_v} (v'p)(p'v) \\ &= \sum_{p \in P_v} v'(pp')v = v'\Big(\sum_{p \in P_v} K_p\Big)v \\ &= v'Q_v v \end{aligned} \tag{3}$$

dove

$$K_p = pp' = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \tag{4}$$

Se i piani di v_1 e v_2 sono disgiunti, l'errore quadratico di v è uguale alla somma degli errori quadratici di v_1 e v_2. Altrimenti, almeno un piano è incluso due volte e, quindi, teoricamente dovremmo trovare Q_v come

$$\begin{aligned} Q_v &= \sum_{p \in P_{v_1} \cup P_{v_2}} K_p \\ &= \sum_{p \in P_{v_1}} K_p + \sum_{p \in P_{v_2}} K_p - \sum_{p \in P_{v_1} \cap P_{v_2}} K_p \\ &= Q_{v_1} + Q_{v_2} - Q_c \end{aligned} \tag{5}$$

Per maglie complicate, questo può portare a problemi di sovraccarico di memoria e di calcolo, perciò al posto di Q_v utilizziamo $\hat{Q}_v$, definito dalla seguente

$$\hat{Q}_v = Q_{v_1} + Q_{v_2} = \begin{cases} Q_v & \text{se } P_{v_1} \cap P_{v_2} = \varnothing \\ Q_v + Q_c, & \text{altrimenti.} \end{cases} \tag{6}$$

Per completare l'operazione di collassamento $[v_1, v_2] \longrightarrow v$, non ci resta che trovare la posizione del vertice v. Poiché l'errore geometrico è una funzione quadratica, trovarne il minimo è facile (funzione lineare). Infatti il valore di v che minimizza la (3) è quello che risolve la seguente

$$\begin{bmatrix} \partial\Delta/\partial x \\ \partial\Delta/\partial y \\ \partial\Delta/\partial z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

che è equivalente a risolvere per $\bar{v}$ la seguente

$$A\bar{v} = b \tag{8}$$

dove

$$A = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{9}$$

Se la matrice A è invertibile, allora otteniamo $\bar{v}$ semplicemente come

$$\bar{v} = A^{-1}b \tag{10}$$

Se la matrice A non è invertibile, allora si può pensare di scegliere v tra v_1, v_2 e $(v_1 + v_2)/2$ a seconda di quale fra questi produce errore più piccolo.

Concludendo, possiamo riassumere l'algoritmo nei seguenti 4 passaggi:

1. Si calcolano le matrici Q per tutti i vertici della maglia originale.

2. Si calcolano i vertici $\bar{v}$ risultanti dal collassamento dei vertici v_1 e v_2 della maglia originale. L'errore quadratico associato a questo vertice

$$\bar{v}'(Q_{v_1} + Q_{v_2})\bar{v}$$

è il costo per collassare il lato.

3. Si inseriscono tutti i lati in una coda prioritaria ordinata in base al costo di collassamento.

4. Iterativamente si rimuovono i lati con costo minimo, si collassa il lato $[v_1, v_2]$ e si aggiorna il costo di tutti i lati che coinvolgono v_1 e v_2. Questo procedimento continua finché l'errore geometrico della maglia semplificata resta sotto la soglia specificata.

È nostro desiderio che alla fine del processo di semplificazione i nodi siano distribuiti in modo bilanciato nello spazio libero. Usando però solo l'errore quadratico convezionale come condizione di terminazione, lo spazio libero viene oltremodo semplificato. La maggior parte dei lati situati nello spazio libero, infatti, hanno un costo di collassamento nullo: se i piani di due vertici v_1 e v_2 sono piani la cui equazione è $z = 0$, il vertice v viene posto sul piano $z = 0$, per cui l'errore quadratico di v è zero. Poichè collassare tali lati ha costo nullo, la maggior parte dei lati presenti nella zona libera vengono collassati e solo pochi vertici vi restano.

Per risolvere il problema, non permettiamo collassamenti di lati che comportino la nascita di altri lati la cui lunghezza sia superiore a una lunghezza massima stabilita a priori. Empiricamente si è trovato che una lunghezza soglia del 10% della dimensione della mappa è un buon compromesso per efficienza e qualità.

Per cui, il processo di semplificazione termina quando non ci sono più lati che, collassando, generano altri lati di lunghezza inferiore a quella della soglia e un errore geometrico anch'esso inferiore alla soglia prestabilita.

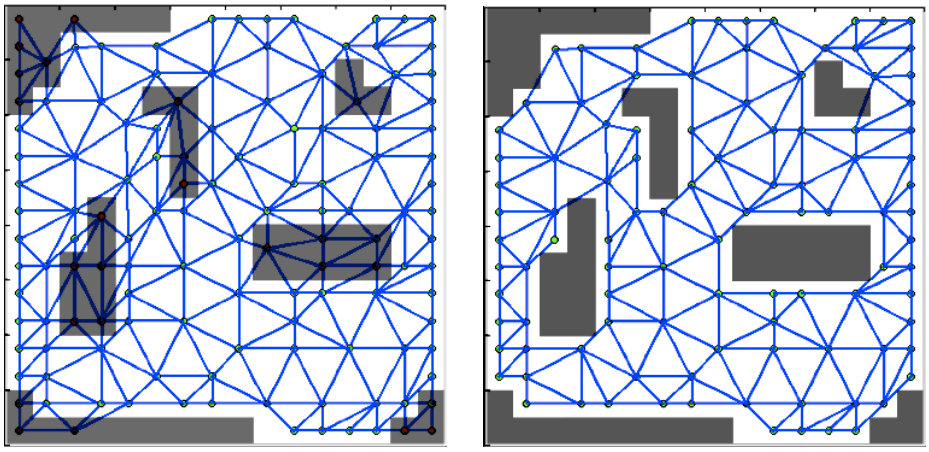
Rifacendoci alla maglia di base mostrata in fig. 11(b), il risultato che si ottiene dopo il processo di semplificazione è mostrato in fig. 13(a).

2.6.3 Generazione del grafo di percorso

Come è facile capire, il grafo che si ottiene dal processo di semplificazione non è sicuramente adatto per la ricerca di un percorso da far eseguire al robot, a motivo della presenza di vertici e lati situati su spazi occupati da ostacoli. Per cui è necessario modificare la maglia per la generazione di un grafo adatto alla navigazione che chiamiamo *grafo di percorso*.

Il *grafo di percorso* è costituito da tutti e soli i vertici e lati della maglia semplificata che stanno sullo spazio libero. Per generarlo, quindi, si eliminano semplicemente tutti i vertici e tutti i lati della maglia semplificata che si trovano su spazi ostacoli. In pratica, si eliminano tutti i vertici (con i relativi lati a loro incidenti) che hanno una coordinata z superiore a una certa soglia prossima a zero. Il risultato di questa fase, per la maglia di fig. 13(a), è mostrato in fig. 13(b).

Durante il processo di esemplificazione, però, può accadere che lati che si trovavano inizialmente in regioni libere, vadano a intersecare spazi occupati da ostacoli.



(a) (b)

Figura 13: (a) Vista 2D della maglia semplificata con evidenziati in rosso i vertici situati sugli ostacoli. (b) Vista 2D del relativo grafo di percorso.

Per cui, i lati che delimitano gli ostacoli devono essere aggiustati in modo che tornino a giacere in spazi liberi per poter essere poi inclusi nel grafo finale.

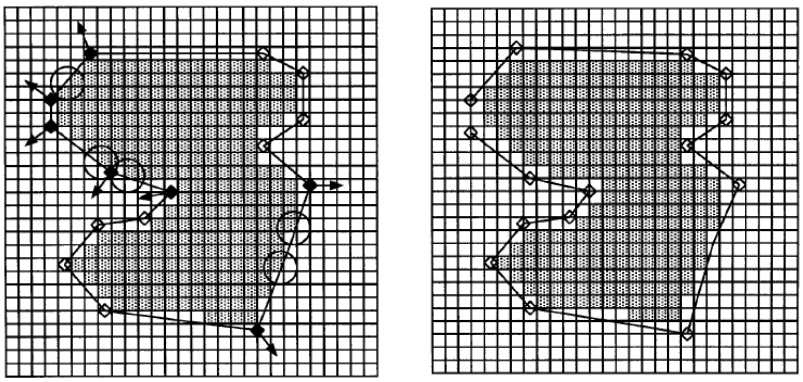
Una possibile soluzione è quella di spingere i vertici adiacenti ai lati che intersecano gli ostacoli verso lo spazio libero di una piccola quantità, finchè non ci sono più lati che intersecano spazi occupati da ostacoli. La direzione verso cui spingere i vertici è quella del vettore somma tra i due vettori ortogonali ai due lati che delimitano l'ostacolo e che si incontrano in quel vertice.

La quantità della quale devono essere spinti i vertici non è immediata da trovare. Spingere di piccole quantità permette di approssimare meglio i contorni degli ostacoli ma il numero di iterazioni necessarie aumenta. D'altra parte, spingere di una quantità troppo grande potrebbe portare a non gestire correttamente i passaggi stretti; infatti tale metodo potrebbe fallire nel momento in cui un lato che inizialmente intersecava un ostacolo, dovesse intersecarne nuovamente un'altro dopo la spinta dei relativi vertici.

Per trovare i lati che intersecano lo spazio occupato da ostacoli si può usare un analizzatore digitale discreto (DDA) che restituisce le celle di una griglia uniforme attraversate dal lato in questione. Per una trattazione dettagliata, si veda [4]. Il risultato di tale operazione è mostrato con un esempio in fig. 14.

2.6.4 Ricerca del percorso minimo e *smoothing* del percorso

Una volta ottenuto il grafo di percorso non orientato $G = (V, E)$, possiamo cercare il percorso minimo da un vertice del grafo a un altro implementando l'algoritmo



(a) (b)

Figura 14: Esempio di aggiustamento del grafo di percorso. (a) Le regioni cerchiate indicano dove i lati si intersecano con l'ostacolo. I vertici da spingere sono marcati in nero, mentre le direzioni verso cui spingere sono indicate dalle frecce. (b) Dopo l'operazione di spinta, tutti i vertici e i lati si trovano su uno spazio libero da ostacoli.

di Dijkstra che nella sua versione efficiente permette di calcolare il percorso minimo da un assegnato vertice s a tutti gli altri vertici t del grafo in un tempo $O(n^2)$. Per il codice dell'algoritmo si veda [10].

Poiché il percorso ottenuto potrebbe avere molti e improvvisi cambi di direzione, si effettua infine un rilassamento attraverso una fase di *smoothing*.

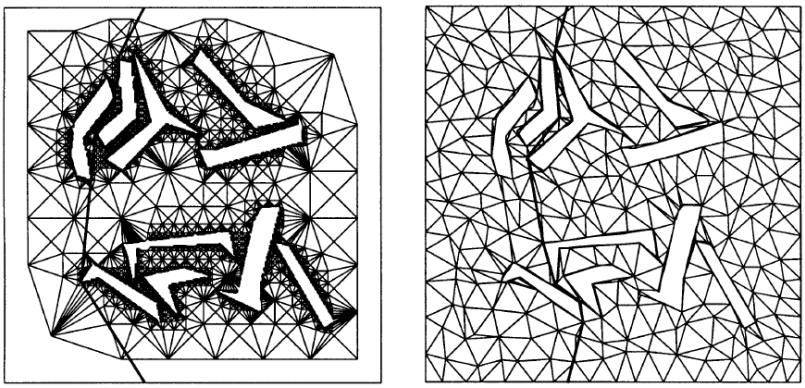
Possiamo concludere mettendo in evidenza i notevoli vantaggi di questo algoritmo rispetto all'approccio basato su *Quadtree*:

- La ricerca del percorso è notevolmente più veloce perché il grafo del percorso è di gran lunga più semplice.

- Il grafo riesce a rappresentare lo spazio in maniera accurata, riuscendo a mantenere anche un alto livello di dettaglio degli ostacoli, attraverso l'uso di celle triangolari.

- Il grafo garantisce il ritrovamento del percorso di costo minimo perché i nodi del grafo sono distribuiti in maniera bilanciata.

In fig. 15 sono messi a confronto i grafi che si ottengono usando (a) un *Quadtree* e (b) un approccio a maglie triangolari su una mappa d'esempio. In tabella 1 è invece mostrato il confronto della complessità dei due metodi. Si nota subito come con l'approccio a maglie triangolari riesca ad ottenere un grafo molto più semplice senza però peggiorare i dettagli dei bordi degli ostacoli (solo 2694 lati).



(a) (b)

Figura 15: Pianificazione del percorso dal punto (192,0) al punto (192,512) su di uno spazio di dimensioni (512x512). (a) *Quadtree*. (b) Maglie Triangolari.

Il *Quadtree* presenta invece molte piccole celle in prossimità dei bordi, portando ad un significativo aumento dei lati del grafo (26638). Inoltre, usando una maglia triangolare, la lunghezza di tutti i lati è pressoché uguale, diversamente dal *Quadtree* che presenta grosse celle negli spazi liberi.

	Quadtree (512x512)	**Maglie Triangolari** (512x512)
nodi del grafo	4225	**502**
lati del grafo	26638	**2694**
tempo (s)	0.875	**0.09**
lunghezza	554.506	**552.7**

Tabella 1: Confronto tra le complessità di pianificazione di percorso.

2.7 Sviluppi futuri

In questa considerazione si è assunto il fatto di avere una mappa 2-D completa e statica. L'algoritmo può essere però facilmente esteso ad ambienti 3-D dinamici. Infatti l'errore quadratico (3) può essere definito anche per ambienti in 3 dimensioni. L'unica accortezza sta nel ridefinire la fase di generazione della griglia (i triangoli diventano tetraedri).

Si può pensare poi di aggiungere i vincoli differenziali del robot, che possono essere visti come vincoli locali, a differenza dei vincoli globali già considerati e dovuti alla

presenza di ostacoli, in modo da assicurare che il percorso trovato sia effettivamente realizzabile dal robot. Come sappiamo, infatti, i vincoli limitano in ogni punto le velocità possibili. Come abbiamo visto, un approccio è quello di ignorare tali vincoli e sperare che in qualche modo vengano gestiti appropriatamente dall'algoritmo di controllo (*tracking* della traiettoria). Un approccio migliore e più accurato resta però quello di considerare i vincoli differenziali nella pianificazione del percorso. In questo modo, il percorso risultante è in armonia con il moto naturale del sistema meccanico.

Si può infine pensare di estendere il problema del *motion planning* al caso in cui si abbia una conoscenza solo parziale, o perfino nulla, dello spazio di lavoro.

3 Tracking di traiettorie

3.1 Premesse e finalità

Consideriamo in questa sezione due problemi strettamente legati tra loro per un WMR in uno spazio piano libero da ostacoli:

- Movimento punto a punto: partendo da un punto del piano con una certa direzione si desidera raggiungere una configurazione finale assegnata.

- Inseguimento di traiettorie: si desidera inseguire sul piano una traiettoria descritta da una legge temporale opportunamente regolare, partendo da una configurazione iniziale nota.

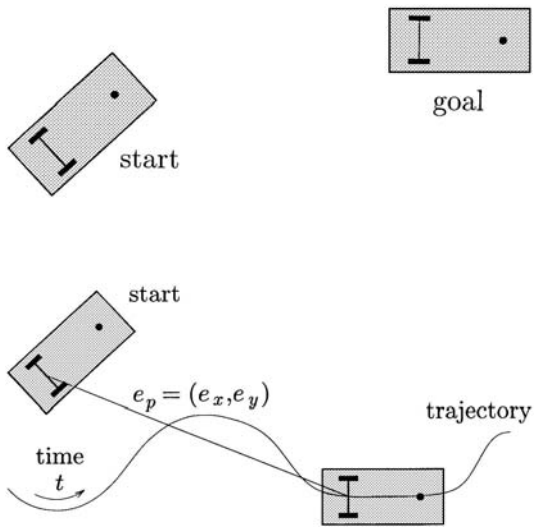


Figura 16: Movimento punto a punto e tracking

Lo schema di controllo adottato è simile nei due casi, con la sola differenza dell'utilizzo di un controllore in feedforward per il tracking di traiettorie. In realtà la catena di feedforward non è essenziale dal punto di vista teorico ma permette di aumentare notevolmente le prestazioni in transitorio. È utile evidenziare il fatto che l'algoritmo garantisce, in ipotesi di idealità (modello perfetto, assenza di saturazione dell'ingresso, assenza di disturbi, ...), l'inseguimento con errore asintotico nullo; nulla invece possiamo asserire circa la dinamica nel transitorio.

3.2 Modello del sistema e proprietà

Il sistema che prenderemo in esame nella trattazione seguente è un classico uniciclo (Fig. 17). Si tratta di un sistema non lineare, non olonomo, pfaffiano e senza drift.

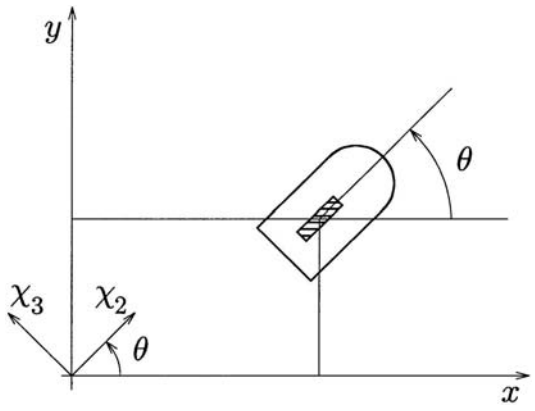


Figura 17: Descrizione del robot Khepera

Sia $q = (x, y, \theta) \in \mathbb{R}^2 \times [0, 2\pi]$ il vettore delle coordinate. I vincoli sulle velocità istantanee sono descritti dalla seguente:

$$A(q)\dot{q} = \dot{x}\sin\theta - \dot{y}\cos\theta$$

da cui si ottiene il modello di ordine uno:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{11}$$

dove v e ω rappresentano rispettivamente la velocità lineare e angolare, attorno all'asse verticale, della ruota. In realtà il sistema Khepera (Fig. 18) su cui verranno eseguiti gli esperimenti è un robot a due ruote non sterzanti, controllabile sia in posizione sia in velocità. Tuttavia il modello è equivalente a quello dell'uniciclo, nel senso che esiste una funzione biunivoca che lega i due:

$$v = \frac{r}{2}(\omega_R + \omega_L)$$
$$\omega = \frac{r}{d}(\omega_R - \omega_L)$$

dove r e d sono rispettivamente il raggio e la distanza tra le ruote e ω_R ed ω_L le velocità angolari delle ruote destra e sinistra.

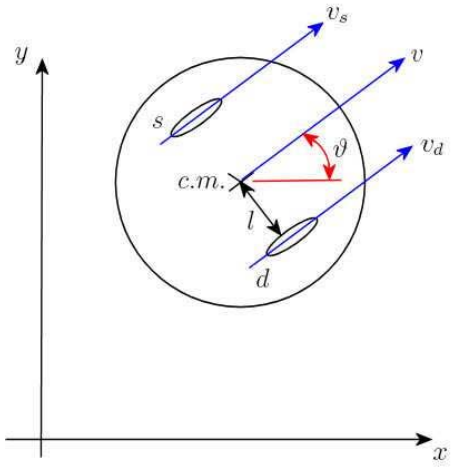


Figura 18: Descrizione dell'uniciclo

3.2.1 Controllabilità e stabilizzabilità su una traiettoria

Per un'analisi dettagliata delle proprietà di controllabilità di sistemi anolonomi rimandiamo all'articolo [2], limitandoci in questa sede a richiamare alcuni risultati notevoli.

Sia $[g_1, g_2]$ la parentesi di Lie di g_1 e g_2. Si verifica che:

$$\text{rank} \begin{bmatrix} g_1 & g_2 & [g_1, g_2] \end{bmatrix} = n = 3$$

Allora, per il teorema di Chow, il sistema è controllabile, ovvero è possibile determinare un ingresso $u(t)$ che porti il sistema da uno stato iniziale x_0 ad uno stato finale x_f arbitrari.

Per quanto riguarda invece la stabilizzabilità lungo una traiettoria chiamiamo $q_d(t) := \begin{bmatrix} x_d(t) & y_d(t) & \theta_d(t) \end{bmatrix}$ la dinamica da inseguire e $u(t) := \begin{bmatrix} v_d(t) & \omega_d(t) \end{bmatrix}$ l'ingresso di controllo richiesto per il tracking. Si può dimostrare che condizione necessaria e sufficiente per la risolubilità del problema è che il gramiano di controllabilità sia non singolare. Si verifica infine che tale condizione è soddisfatta se e solo se $v_d(t) \neq 0$ e $\omega_d(t) \neq 0$ lungo la stessa traiettoria.

3.2.2 Dynamic Feedback Linearization

La strategia di controllo adottata va sotto il nome di Dynamic Feedback Linearization e permette di risovere in modo unificato il problema del tracking e quello del movimento punto a punto in un ambiente privo di ostacoli. Si tratta in sostanza di costruire un controllore dinamico non lineare in modo che il sistema aumentato, in catena chiusa, sia equivalente ad un sistema lineare a meno di un cambiamento di

coordinate. La costruzione di tale controllore segue da un algoritmo di disaccoppiamento ingresso-uscita che qui descriviamo nel caso particolare dell'uniciclo; si faccia rifermento a [1] per la generalizzazione ad altre classi di sistemi.

Definiamo il vettore delle uscite $\eta := \begin{bmatrix} x \\ y \end{bmatrix}$. Risulta semplicemente:

$$\dot{\eta} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} = v \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$$

Inseriamo a questo punto un integratore con stato ξ e poniamo:

$$v = \xi \qquad \dot{\xi} = a \ ,$$

per cui abbiamo:

$$\dot{\eta} = \xi \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix}$$

Fisicamente a rappresenta l'accelerazione dell'uniciclo ed è un nuovo ingresso virtuale. Risulta ancora:

$$\ddot{\eta} := \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \dot{\xi} \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} + \xi\dot{\theta} \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\xi\sin\theta \\ \sin\theta & \xi\cos\theta \end{bmatrix} \begin{bmatrix} a \\ \omega \end{bmatrix}$$

La matrice dell'ultimo membro è non singolare se e solo se $\xi \neq 0$. In tal caso otteniamo:

$$\begin{bmatrix} a \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & -\xi\sin\theta \\ \sin\theta & \xi\cos\theta \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Il compensatore dinamico cercato è descritto dalle seguenti:

$$\begin{aligned} \dot{\xi} &= u_1\cos\theta + u_2\sin\theta \\ v &= \xi \\ \omega &= \frac{u_2\cos\theta - u_1\sin\theta}{\xi} \end{aligned} \tag{12}$$

Osserviamo che per $v = \xi = 0$ il sistema è non ben definito. Ci preoccuperemo di questo fatto successivamente quando tratteremo dell'algoritmo di tracking delle traiettorie. Infine con il cambio di coordinate:

$$\begin{aligned} z_1 &= x \\ z_2 &= y \\ z_3 &= \dot{x} = \xi\cos\theta \\ z_4 &= \dot{y} = \xi\sin\theta \end{aligned} \tag{13}$$

otteniamo un sistema in catena chiusa lineare descritto da:

$$\begin{aligned} \ddot{z}_1 &= u_1 \\ \ddot{z}_2 &= u_2 \end{aligned} \tag{14}$$

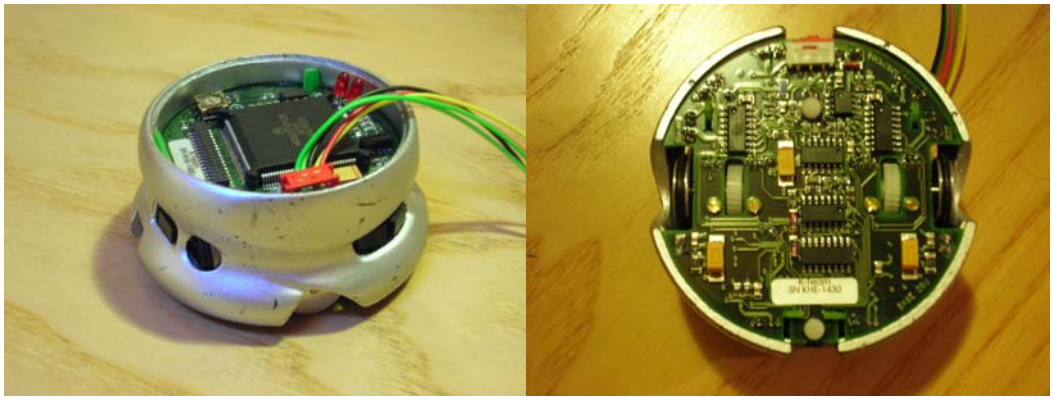


Figura 19: Foto del robot khepera utilizzato per gli esperimenti.

3.2.3 Il robot Khepera

Il robot utilizzato per gli esperimenti, chiamato *Khepera*, è un piccolo robot a due ruote, sviluppato dal K-Team (`www.k-team.com`), che rispecchia bene il modello dell'uniciclo utilizzato per la progettazione (figura 19). La base ha un diametro di 70mm, è alto 30mm ed è capace di raggiungere una velocità massima di 1m/s.

Le ruote, di raggio 7.5mm e distanti 53mm, sono collegate ai motori tramite dei motoriduttori con rapporto di riduzione 25:1, e sui motori sono presenti due encoder a 24 passi. Questo significa che per ogni giro completo delle ruote vengono misurati 600 impulsi degli encoder, ed a ogni millimetro di spostamento corrispondono 12 impulsi (0.083 mm per impulso).

Monta un microcontrollore Motorola 68331 a 25MHz, con 512Kb di RAM e 512Kb di memoria flash programmabile tramite seriale. Il software presente sul robot è in grado di controllare in velocità ed in posizione ciascuno dei due motori, ma non di calcolare lo spostamento o la velocità del robot in coordinate cartesiane.

La comunicazione col computer avviene tramite un collegamento seriale RS232, e con MATLAB si possono usare le funzioni `kopen`, `ksend` e `kclose` fornite dai produttori, e gli script matlab correlati che ne semplificano l'uso.

Le specifiche tecniche del robot vengono riportate in tabella 2.

Al contrario di quanto visto per il modello dell'uniciclo (11), l'uscita riportata dal robot non è la propria posizione, espressa come (x, y, θ), ma piuttosto la posizione delle singole ruote, letta dagli encoder. A partire da queste, non è possibile ricavare, con un cambio di coordinate, la posizione del robot. Invece, è necessario calcolare la posizione integrando il modello (11), usando le misure odometriche. Per fare questo, si stima la velocità media nell'ultimo periodo di campionamento, lineare e rotazionale, e la si assume costante nel periodo. Si ottengono così le seguenti

Elemento	Descrizione
Processore	Motorola 68331, 25MHz
RAM	512Kb
Flash	512Kb
Movimento	2 servomotori brushed in corrente continua, con encoder incrementali (circa 12 impulsi per mm di movimento)
Velocità	Max: 1 m/s, Min: 0.02 m/s
Alimentazione	Alimentatore esterno o batterie ricaricabili NiMH
Autonomia	1 ora di movimento continuo
Comunicazione	Porta seriale RS232, fino a 115kbps
Dimensione	Diametro: 70 mm, Altezza: 30 mm
Peso	Circa 80 g
Carico	Circa 250 g

Tabella 2: Specifiche tecniche del robot Khepera.

equazioni:

$$x(t+\tau) = x(t) + \begin{cases} \frac{v}{\omega}(\sin(\theta+\omega\tau) - \sin(\theta)) & \omega \neq 0 \\ v\tau\cos(\theta) & \omega = 0 \end{cases}$$

$$y(t+\tau) = y(t) + \begin{cases} -\frac{v}{\omega}(\cos(\theta+\omega\tau) - \cos(\theta)) & \omega \neq 0 \\ v\tau\sin(\theta) & \omega = 0 \end{cases}$$

3.3 Inseguimento della traiettoria

La soluzione del problema prevede l'implementazione di un controllore in feedforward che restituisce le velocità nominali per il tracking, e di un controllore in feedback per la correzione degli errori, entrambi non lineari.

3.3.1 Controllo in feedforward

Supponiamo di conoscere a priori la dinamica $(x_d(t), y_d(t))$, $t \in [0, T]$, da inseguire. Il calcolo della velocità lineare è naturale:

$$v_d(t) = \pm\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{15}$$

La scelta del segno è legata al tipo di moto del veicolo, in avanti oppure all'indietro. Dal modello dell'uniciclo si ricava poi:

$$\theta = \arctan\frac{\dot{y}}{\dot{x}}$$

da cui, derivando

$$\omega_d(t) = \frac{\ddot{y}_d(t)\dot{x}_d(t) - \ddot{x}_d(t)\dot{y}_d(t)}{v_d(t)} \tag{16}$$

Dall'espressione della velocità angolare si nota che nasce un problema se esiste $\bar{t} \in [0, T]$ tale che $v_d(\bar{t}) = 0$. Questo potrebbe rappresentare una situazione critica in due casi importanti: la partenza con velocità nulla o la presenza di cuspidi con inversione del moto. Si può cercare risolvere il problema in modo euristico assegnando $\omega_d(\bar{t}) = \bar{\omega}$ oppure calcolando il limite in $\bar{t}$.

3.3.2 Controllo in feedback

Sia $(x_d(t), y_d(t))$, $t \in [0, T]$ la traiettoria nota da seguire, lungo la quale $v_d(t) \neq 0$ (traiettoria persistente). Consideriamo il sistema in catena chiusa ottenuto via dynamic feedback linearization:

$$\ddot{z}_1 = u_1$$
$$\ddot{z}_2 = u_2$$

Supponiamo inoltre note le derivate temporali del primo e del secondo ordine di $x_d(t)$ e $y_d(t)$ e misurabili la posizione e la velocità reali del robot. Secondo lo schema di Fig. 20 inseriamo un controllore dinamico del tipo:

$$\begin{aligned} u_1 &= \ddot{x}_d + k_{d1}(\dot{x}_d - \dot{x}) + k_{p1}(x_d - x) \\ u_2 &= \ddot{y}_d + k_{d2}(\dot{y}_d - \dot{y}) + k_{p2}(y_d - y) \end{aligned} \tag{17}$$

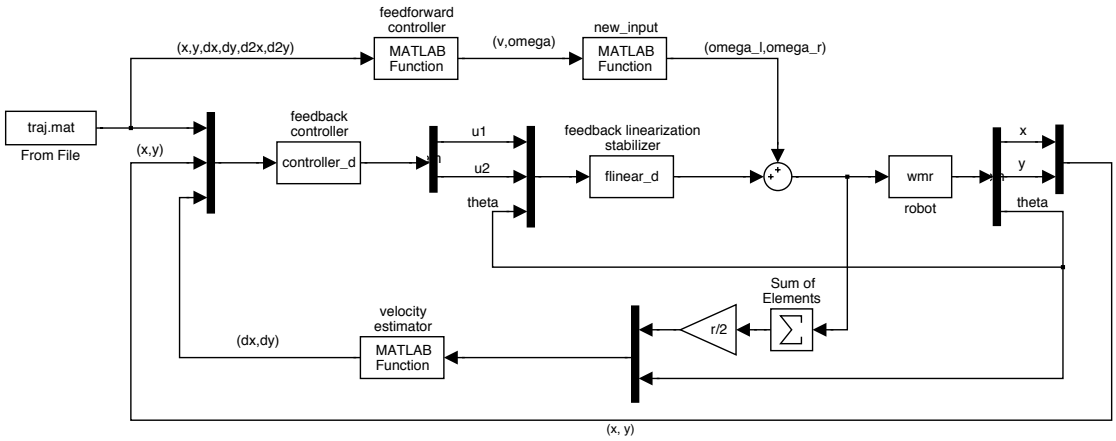


Figura 20: Schema (semplificato) per il controllo

Facciamo le posizioni:

$$\epsilon_x = x_d - x$$
$$\epsilon_y = y_d - y$$

che rappresentano gli errori di posizione del tracking. Poiché $\ddot{x}$ e $\ddot{y}$ non sono misurabili poniamo $\ddot{\epsilon}_x = \ddot{x}_d$ e $\ddot{\epsilon}_y = \ddot{y}_d$. Risulta allora immediatamente:

$$u_1 = \ddot{\epsilon}_x + k_{d1}\dot{\epsilon}_x + k_{p1}\epsilon_x$$
$$u_2 = \ddot{\epsilon}_y + k_{d2}\dot{\epsilon}_y + k_{p2}\epsilon_y$$

Come si vede il controllore non è realizzabile come sistema dinamico lineare. Tuttavia il sistema da controllare è un doppio integratore (o meglio una coppia di doppi integratori, trattandosi di un sistema MIMO). La catena diretta di controllo ha quindi matrice di trasferimento:

$$W(s) = \begin{bmatrix} W_1(s) & 0 \\ 0 & W_2(s) \end{bmatrix} \tag{18}$$

dove:

$$\begin{aligned} W_1(s) &= \frac{Z_1(s)}{\epsilon_x(s)} = 1 + \frac{k_{d1}}{s} + \frac{k_{p1}}{s^2} \\ W_2(s) &= \frac{Z_2(s)}{\epsilon_y(s)} = 1 + \frac{k_{d2}}{s} + \frac{k_{p2}}{s^2} \end{aligned} \tag{19}$$

ed il sistema retroazionato risulta stabile (lungo la traiettoria). Questo risultato rimane valido se lo stato del compensatore non raggiunge il punto singolare $\xi = 0$. Esistono condizioni sufficienti, legate alla scelta dei guadagni k_{pi} e k_{di} e dello stato iniziale ξ_0, affinché venga evitata la singolarità lungo la traiettoria; si faccia riferimento per esempio a [1]. Una soluzione alternativa consiste invece nel resettare lo stato ξ ad un opportuno valore qualora scenda sotto un livello di guardia, oppure utilizzare un fattore di rapporto $\tilde{\xi} \neq 0$ piccolo a piacere. In ogni caso si ammette che la velocità del robot possa subire discontinuità isolate (ricordiamo che $\xi = v$).

3.3.3 Risultati sperimentali

Per verificare il funzionamento del regolatore, si è provato con due diversi tipi di traiettorie. La prima è una semplice retta, alla quale il robot, partendo da una posizione arbitraria, deve prima avvicinarsi e quindi seguirla.

La seconda traiettoria usata è un cerchio di raggio 20 cm, centrato nell'origine, e la posizione di partenza del robot è proprio l'origine, con orientazione $\theta = 0$.

Infine, la terza è una traiettoria a forma di otto, in cui sono presenti curve e cambi di direzione. In questo terzo caso, il robot è stato fatto partire dal punto iniziale corretto.

Per gli esperimenti di tracking, i guadagni del controllore PD sono stati posti a

$$K_p = 2,\ K_d = 0.7,$$

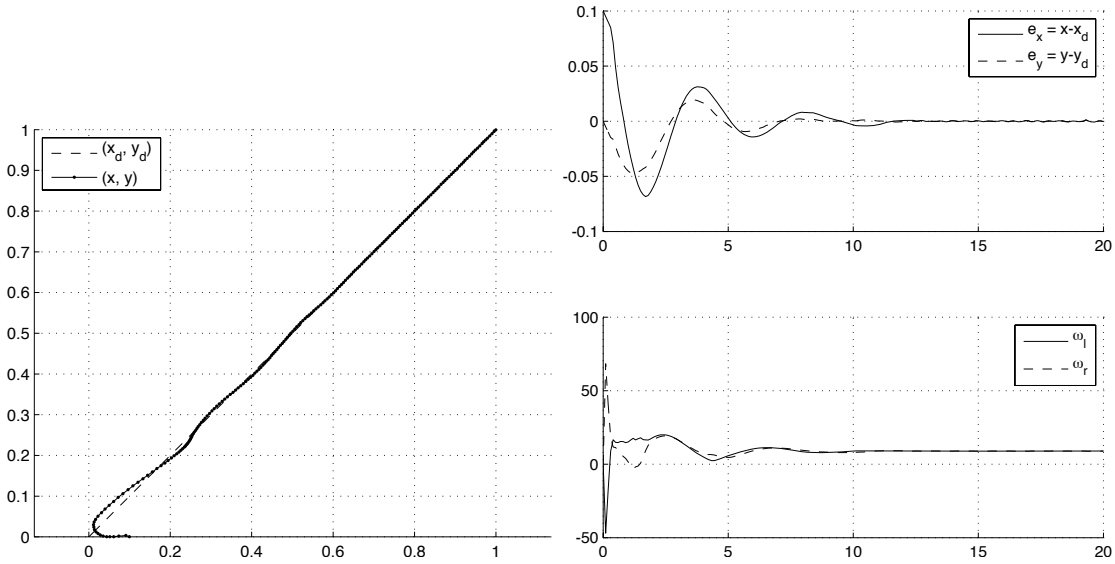


Figura 21: Tracking di una traiettoria rettilinea. Nei grafici si osserva la traiettoria sul piano, l'errore su x e y, e i comandi di velocità dati ai motori.

mentre per gli esperimenti di spostamento punto-punto (posture stabilization) questi sono

$$K_p = 0.5,\ K_d = 0.7.$$

Variando questi parametri, si possono ottenere risultati migliori con una particolare traiettoria o un particolare spostamento, ma è difficile calibrarli in modo uniforme. Probabilmente, sarebbe possibile migliorare le prestazioni tenendo in considerazione anche la dinamica del robot oltre che il modello cinematico (11).

3.3.4 Retta

Analizziamo prima i risultati ottenuti nel caso della traiettoria più semplice, un segmento di retta. Il segmento utilizzato ha per estremi $(0,0)$ e $(1,1)$, quindi di lunghezza $\sqrt{2}$ metri, con legge temporale uniforme, per la durata complessiva di 20 secondi. La posizione iniziale del robot è

$$x_0 = 0.1,\ y_0 = 0,\ \theta_0 = \frac{\pi}{2}.$$

In figura 21 si osserva la traiettoria seguita dal robot nelle condizioni sopra esposte. Il transitorio dura circa 10-12 secondi, tuttavia dopo 8 secondi l'errore è già inferiore ad 1cm. A regime, l'errore è inferiore ad un millimetro.

Bisogna osservare, tuttavia, che questo è l'errore misurato in base alle sole misure odometriche, che vengono integrate in base al modello. A causa della propagazione

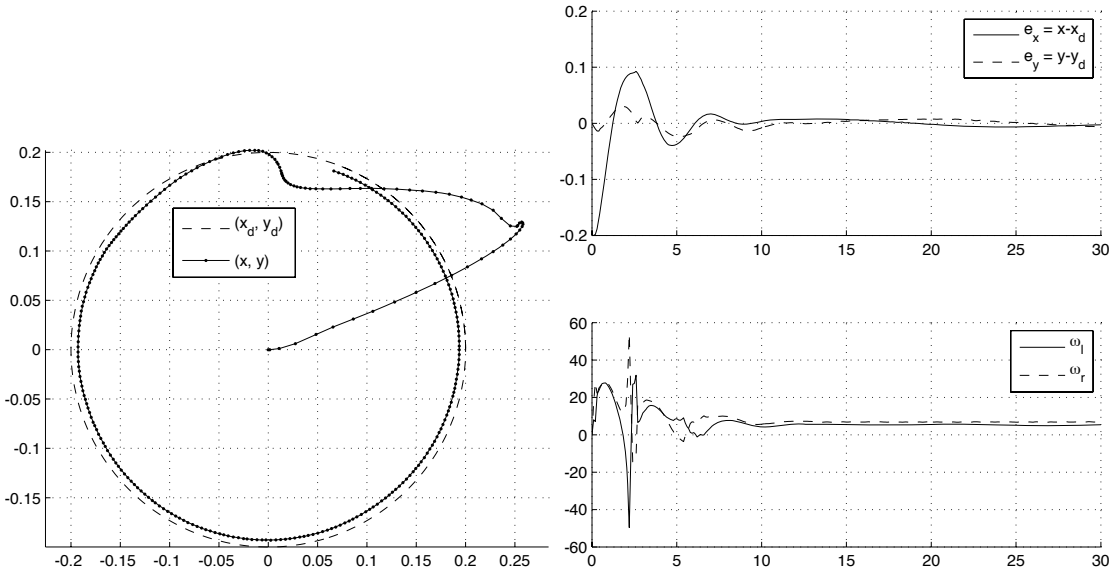


Figura 22: Tracking di una traiettoria circolare. Nei grafici si osserva la traiettoria sul piano, l'errore su x e y, e i comandi di velocità dati ai motori.

degli errori, qualunque imprecisione numerica o dovuta alla dinamica del robot può portare la posizione reale a distanziarsi da quella misurata. Per far fronte a questo problema, è necessario disporre di un metodo più preciso per localizzare il robot.

3.3.5 Traiettoria circolare

In questo caso, la traiettoria ha un raggio di curvatura costante, pari a 20 cm, e la posizione iniziale del robot è il centro del cerchio.

Si osserva in figura 22 che il robot inizialmente si move molto velocemente lungo il raggio del cerchio, quindi presenta una grossa sovraelongazione, ma dopo 6-7 secondi l'errore scende e si mantiene sotto 1 cm. Per migliorare le prestazioni, in questo caso, bisogna agire sul guadagni del controllore PD, in base alle esigenze.

3.3.6 Traiettoria a otto

In questa traiettoria si osserva il comportamento del robot lungo una traiettoria curvilinea, con curvatura non costante, al contrario del caso precedente. Questa si

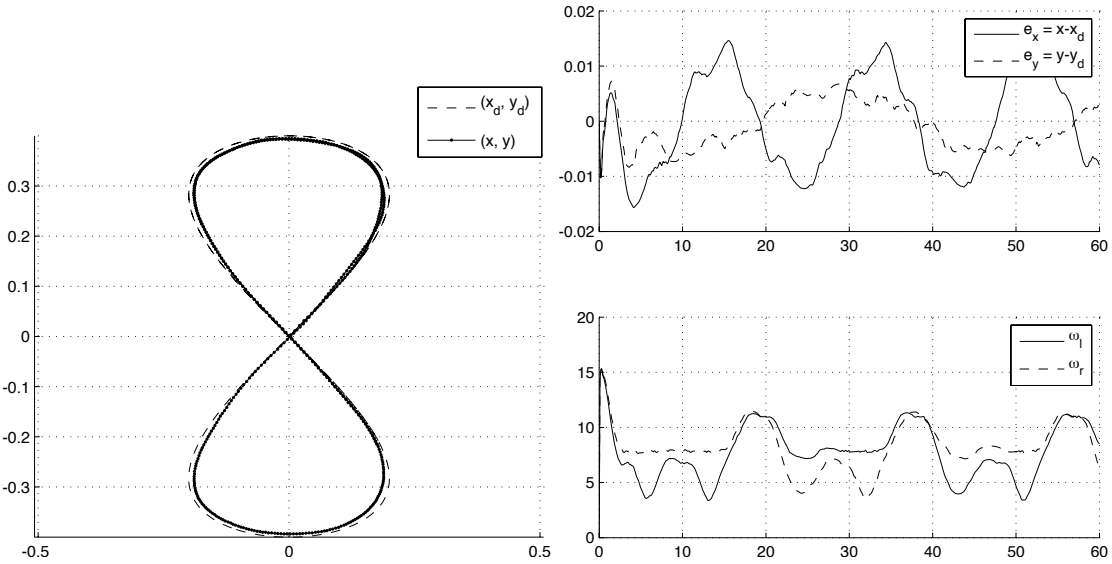


Figura 23: Tracking di una traiettoria a forma di otto. Nei grafici si osserva la traiettoria sul piano, l'errore su x e y, e i comandi di velocità dati ai motori.

può esprimere in forma parametrica come:

$$x = a_x \sin\left(\frac{s}{k_x}\right)$$
$$y = a_y \sin\left(\frac{s}{k_y}\right)$$

e le costanti a_x, a_y, k_x e k_y sono state scelte pari a

$$a_x = 20 \text{ cm}, a_y = 40 \text{ cm}, k_x = 3 \text{ cm}, k_y = 6 \text{ cm},$$

e la legge temporale $s(t)$ scelta è semplicemente $s(t) = t$. La posizione iniziale del robot è quella per $s = 0$, ovvero

$$x_0 = 0,\ y_0 = 0,\ \theta_0 = \frac{\pi}{4}$$

e la durata dell'esperimento è di 60 secondi, tempo sufficiente a percorrere il circuito completo due volte. Partendo dal punto iniziale della traiettoria, il regolatore è già a regime, senza nessun transitorio (se non quello, dovuto alla dinamica e trascurato nel nostro modello, per raggiungere la velocità iniziale).

Come si osserva in figura 23, la traiettoria viene inseguita abbastanza bene, ma l'errore è molto più marcato che nei casi precedenti, e arriva fino a 1.5cm, in corrispondenza delle curve più strette.

Anche in questo caso, mentre i grafici riportano un buon inseguimento, è facile osservare guardando il robot che l'errore reale è molto più grande, a causa degli errori di localizzazione dovuti all'integrazione delle misure odometriche. Infatti, il secondo giro non ripassa esattamente per l'origine, ovvero il punto iniziale, ma passa a una distanza di circa 10 cm. Tuttavia, con le sole misure della posizione degli encoder, questo è il meglio che si riesce a fare.

3.4 Stabilizzazione di una configurazione

Il punto di forza della strategia adottata sta nella sua versatilità. E' possibile infatti adattare lo schema del tracking di traiettorie per risolvere il problema del movimento punto a punto. Naturalmente i guadagni k_{pi} e k_{di} vanno scelti indipendentemente dal caso precedente; lo stesso vale per lo stato iniziale ξ_0. Senza perdita di generalità consideriamo come configurazione finale l'origine: $x = y = 0$, $\theta = 0$. La configurazione iniziale è invece arbitraria. Per la proprietà di convergenza asintotica con errore nullo dell'algoritmo di tracking basterà porre come traiettoria di riferimento: $x_d(t) = y_d(t) \equiv 0$. E' necessario in ogni caso evitare la singolarità $\xi = 0$ durante il transitorio e asintoticamente. La seguente proposizione, per la cui dimostrazione rimandiamo sempre a [1], risponde al quesito:

Proposizione 1. *Sia $Q = \mathbb{R}^2 \times [0, 2\pi]$ lo spazio delle configurazioni ammissibili. Indichiamo con $Q^* = \{q \in Q : (x = 0, \cos\theta \geq 0) \vee (y = 0, \cos\theta = -1) \vee (x = y = 0)\}$ un suo sottoinsieme. Imponendo $x_d = y_d \equiv 0$ in (17), il sistema (14) in catena chiusa, retroazionato con (17), converge all'origine per ogni configurazione iniziale $q_0 \in Q - Q^*$ se valgono le seguenti:*

1. *condizione sui guadagni:*

$$\begin{aligned} k_{d1}^2 - 4k_{p1} = k_{d2}^2 - 4k_{p2} > 0 \\ k_{d2} - k_{d1} > 2\sqrt{k_{d2}^2 - 4k_{p2}} \\ k_{di}, k_{pi} > 0 \quad , \quad i = 1, 2 \end{aligned} \tag{20}$$

2. *condizione sullo stato iniziale ξ_0:*

$$\begin{aligned} &\xi_0 < 0 \qquad , \text{ se } q_0 \in Q^r := \{q \in Q - Q^* : x \geq 0\} \\ &\xi_0 > 0 \qquad , \text{ se } q_0 \in Q^l := \{q \in Q - Q^* : x < 0\} \\ &\xi_0 \neq 2\frac{k_{p1} x_0 \sin\theta_0 - k_{p2} y_0 \cos\theta_0}{k_{d2} - k_{d1}} \end{aligned} \tag{21}$$

Se la configurazione iniziale q_0 appartiene a Q^* il controllo porta l'uniciclo nell'origine con l'orientamento non desiderato. Si potrebbe risolvere il problema in modo

elegante introducendo un punto $q_v \notin Q^*$ di passaggio obbligato e ottenere convergenza all'origine in due fasi successive. Tuttavia il risultato della proposizione è valido solo in condizioni ideali. Alcune simulazioni hanno mostrato come, in presenza di saturazione dell'ingresso di controllo, il sistema si porti ancora a $x = y = 0$ ma con orientamento non ben precisato. Una soluzione euristica al problema è la seguente:

1. applicare l'algoritmo di controllo per la convergenza al set-point q_f;

2. se la distanza attuale dal set-point (si considerano solo le coordinate cartesiane x ed y) è inferiore ad un livello assegnato, interrompere l'algoritmo e far ruotare su se stesso l'uniciclo (a velocità angolare costante) fino a raggiungere l'orientamento desiderato.

3.4.1 Risultati sperimentali

Per testare il funzionamento del controllore per la stabilizzazione della postura, si sono fatte due prove. Nella prima, il punto finale è posto a 45° rispetto alla direzione del robot, ad una distanza di circa 56.6cm, mentre nella seconda, il robot deve spostarsi di lato di 10 cm.

3.4.2 Spostamento in diagonale

La posizione iniziale del robot è

$$x_0 = 0,\ y_0 = 0,\ \theta_0 = 0,$$

e la posizione di arrivo è posta a

$$x_f = 40\text{cm},\ y_f = 40\text{cm},\ \theta_f = -\frac{\pi}{2}.$$

In figura 24 si osserva la traiettoria compiuta dal robot per raggiungere la posizione. Come si vede, inizialmente questo si gira nella direzione corretta, tuttavia una volta giunto a destinazione, la supera ed è costretto a tornare indietro, presentando una forte sovraelongazione. In simulazione, questo non si verificava, perché come abbiamo già detto, il modello considerato è solo cinematico, e non tiene conto della dinamica (inerzia del robot, in questo caso).

Inoltre, a causa della propagazione degli errori nell'integrazione delle misure odometriche, l'errore di localizzazione del robot cresce col tempo. In questo caso, è facile osservarlo segnando il suolo nella posizione di arrivo prevista del robot e valutando lo scostamento, come si vede nella foto di figura 25.

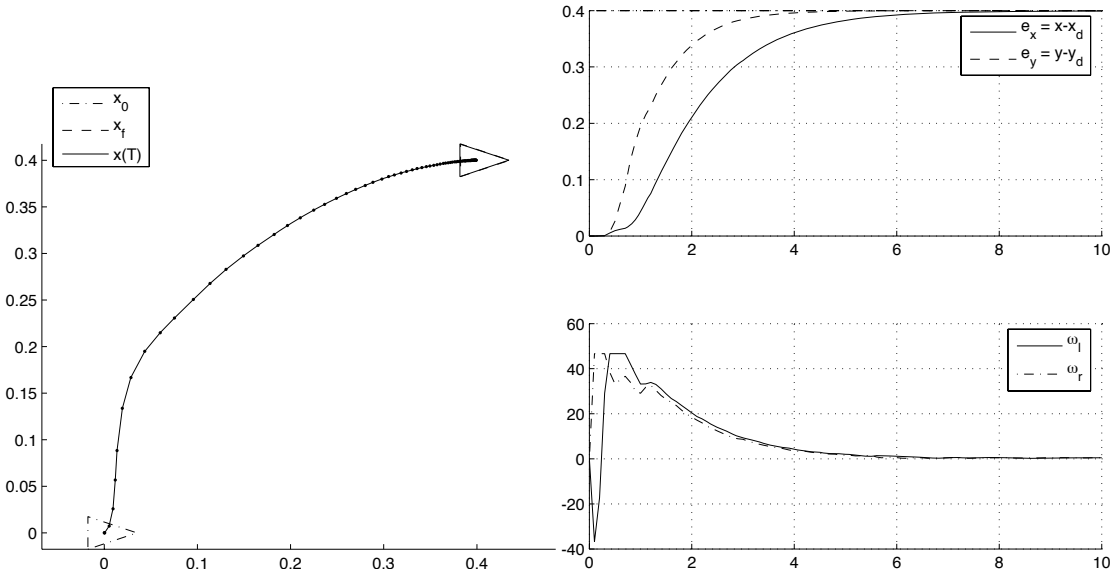


Figura 24: Spostamento lungo la diagonale di un quadrato di lato 40cm. Nei grafici si osserva la traiettoria sul piano, la posizione x e y e i comandi di velocità dati ai motori.

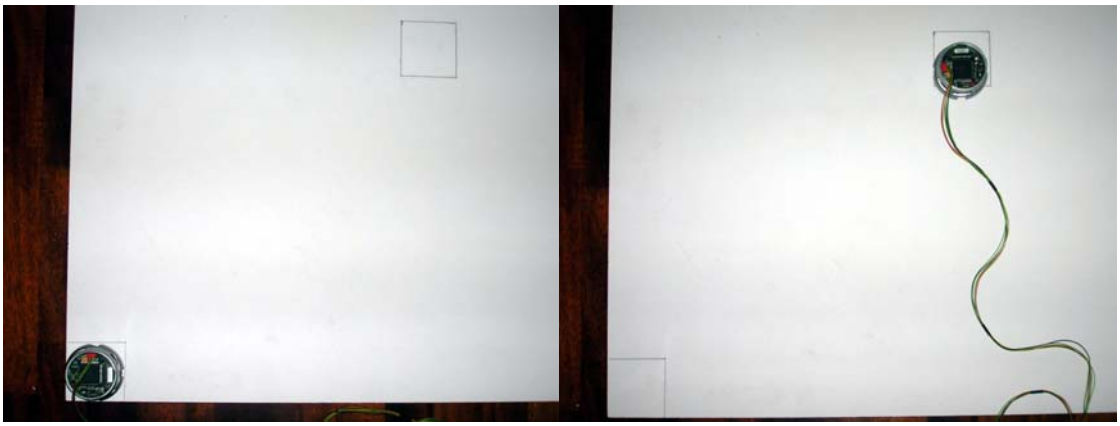


Figura 25: Foto della posizione finale del robot dopo lo spostamento in diagonale. I quadrati disegnati sul suolo indicano la posizione iniziale e finale (corretta). Il robot ha commesso un errore di localizzazione di circa 2 cm.

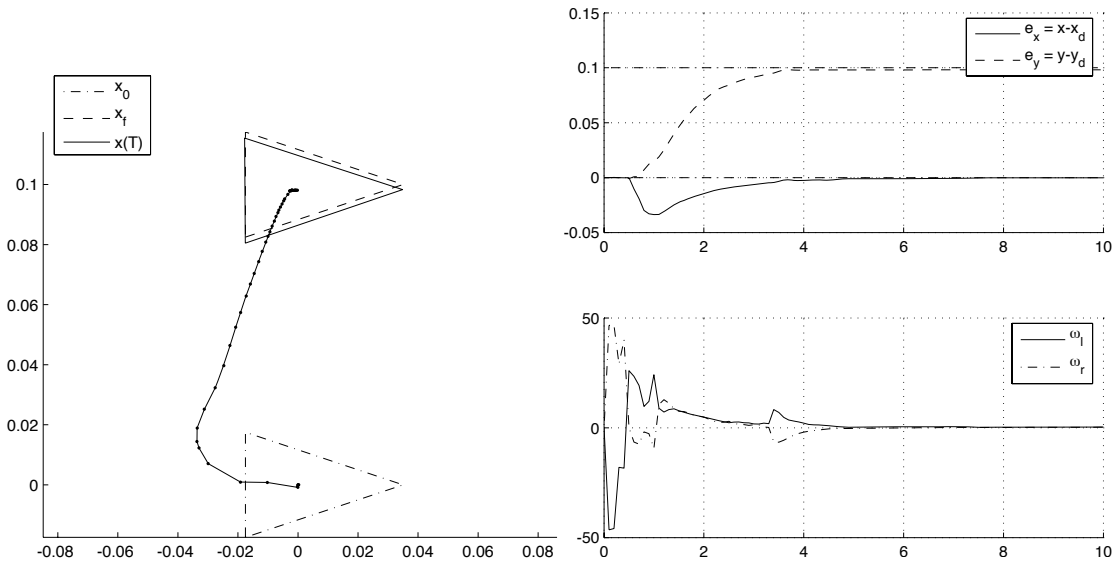


Figura 26: Spostamento in parallelo. Nei grafici si osserva la traiettoria sul piano, la posizione x e y e i comandi di velocità dati ai motori.

3.4.3 Spostamento parallelo

In questa seconda prova, la posizione di partenza è sempre l'origine,

$$x_0 = 0,\ y_0 = 0,\ \theta_0 = 0,$$

mentre la posizione di arrivo è spostata di 10 cm lungo l'asse y:

$$x_f = 0,\ y_f = 10\text{cm},\ \theta_f = 0.$$

In figura 26 si osservano i risultati. Ci si potrebbe aspettare che il robot faccia una traiettoria simile a quella per il parcheggio di un'automobile, ovvero muovendosi prima in avanti e poi all'indietro, sterzando opportunamente. Invece, il regolatore da noi usato decide di far girare il robot nella direzione del punto di arrivo e, una volta li, ruota fino alla posizione corretta.

3.5 Sviluppi futuri

Allo stato attuale delle cose esistono numerose aree di sviluppo per quanto riguarda il controllo di WMR e, in particolare, la tecnica di Dynamic Feedback Linearization.

Introduzione della dinamica Quanto fin qui sviluppato è relativo ad un modello cinematico di ordine uno dell'uniciclo. Naturalmente questa scelta ha portato ad

una notevole semplificazione del problema. L'utilizzo di modelli che descrivono la dinamica del robot è utile qualora tali robot possano essere comandati in accelerazione e/o in coppia motrice.

Utilizzo di sensori esterni In presenza di slittamento delle ruote, l'utilizzo degli encoder diventa impraticabile. Una soluzione possibile è chiudere l'anello di regolazione con sensori capaci di restituire la posizione assoluta del robot (controllo con telecamere).

Controllo robusto Nel presente contesto, per la sintesi del controllore si è supposto il sistema ideale. Si potrebbe pensare di studiare gli effetti dell'introduzione di alcune non idealità comuni e di errori di modello e progettare leggi di controllo che tengano conto di queste incertezze.

WMR complessi L'uniciclo è il più semplice sistema WMR da studiare. Esistono numerosi esempi di sistemi che non ammettono una trasformazione in catena (veicoli con rimorchio) e per i quali bisogna valutare se e come la tecnica DFL è ancora applicabile.

Riferimenti bibliografici

[1] G. Oriolo, A. De Luca, M. Vendittelli. *WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation.* IEEE Transactions on Control Systems Technology, Vol. 10, No 6, November 2006.

[2] R.M. Murray, S. Sastry. *Nonholonomic motion planning: steering using sinusoids.*

[3] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press.

[4] J. Revelles, C. Ureña, M. Lastra. *An Efficient Parametric Algorithm for Octree Traversal.*

[5] S. Kambhampati, L.S. Davis. *Multiresolution Path Planning for Mobile Robots.* IEEE Journal of Robotics and Automation, Vol. RA-2, NO. 3, September 1986.

[6] D.S. Tan, J.T. Herro, R.J. Szczerba. *Simulation of Euclidean Shortest Path Planning Algorithms Based on the Framed-Quadtree Data Structure.* Technical Report: #95-26, October 1995.

[7] H. Samet. *An algorithm for converting rasters to quadtrees.* Trans. Patt. Analy. Mach. Intell., vol. 3, 1981, pp. 93-95.

[8] C.A. Shaffer, Q.F. Stout. *Linear Time Distance Transforms for Quadtrees.* IEEE Transaction on Systems, Man, and Cybernetics Part A: SYSTEMS AND HUMANS, Vol. 33, No. 1, January 2003.

[9] J. Y. Hwang, J. S. Kim, S. S. Lim, and K. H. Park. *A Fast Path Planning by Path Graph Optimization.* IEEE Transaction on Systems, Man, and Cybernetics Part A: SYSTEMS AND HUMANS, Vol. 33, No. 1, January 2003.

[10] M. Fischetti. *Lezioni di Ricerca Operativa.* Edizioni Libreria Progetto, Padova, 1999.