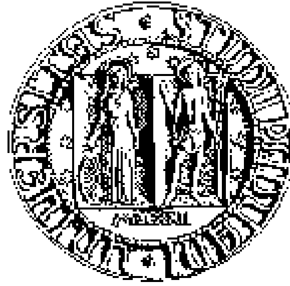
UNIVERSITÀ DEGLI STUDI DI PADOVA



Facoltà di Ingegneria
Corso di Laurea in Ingegneria dell'Automazione L.S.

Corso di Progettazione dei sistemi di Controllo a.a. 2008/09

Navigazione autonoma di robot in ambiente sconosciuto

Pietro Salvagnini, Francesco Simmini, Michele Stoppa
Matricola: 586929 586206 586204
pietro.salvagnini@gmail.com; francesco.simmini@gmail.com; stoppa.michele@gmail.com

Padova, 16 Febbraio 2009

Navigazione autonoma di robot in ambiente sconosciuto

Francesco Simmini, *586206;* Pietro Salvagnini, *586929;* Michele Stoppa, *586204*

***Abstract*—Il problema affrontato è la navigazione autonoma di robot in ambiente sconosciuto. Ci si propone di utilizzare N robot esploratori dotati di una piccola telecamera, ed un attore cieco che conosce solo la sua posizione. Si deve individuare un percorso dalla posizione iniziale dell'attore ad un obbiettivo noto, attraverso la costruzione di una mappa dell'ambiente. A tal scopo si è ideata una soluzione effettivamente implementabile, e la si è testata sperimentalmente. É stato predisposto un originale sistema di controllo che gestisce in tempo reale le complicate operazioni richieste dal problema. Si propone una soluzione al problema del *map-building*, legata al particolare sensore di visione; quindi si presenta un nuovo algoritmo di *esplorazione*, ottimo secondo un ragionevole criterio di selezione delle zone da esplorare. Si è adottato un algoritmo di ricerca dei cammini minimi per il problema del *path-planning* offerto dalla letteratura. Si è utilizzato un sistema GPS virtuale ottenuto applicando un algoritmo di visione alle immagini tratte da una telecamera fissata sopra al piano di lavoro. I test sperimentali hanno verificato la validità del sistema proposto per due robot esploratori ed un attore cieco.**

INDICE

I Introduzione 1

II Panoramica Generale 2

III Il Sistema di Controllo 3

IV Mapping 4
- IV-A Stato dell'arte 4
- IV-B Tipologia e dimensione della mappa . . 5
- IV-C Rilevamento degli ostacoli 6
- IV-D Costruzione della mappa 6
- IV-E Aggiornamento della mappa 7

V Esplorazione 8
- V-A Stato dell'arte 8
- V-B Algoritmo di esplorazione 9
- V-C Estensione Multirobot 10
- V-D Terminazione dell'algoritmo 11

VI L'algoritmo di ricerca dei cammini 11
- VI-A L'algoritmo D* (D-star) 11

VII Controllo della traiettoria e rilocalizzazione 13
- VII-A Controllo della traiettoria 13
- VII-B Rotazione del robot 14
- VII-C Rilocalizzazione 14

VIII Risultati e conclusioni 15

Appendice A: Aspetti tecnici ed implementativi 16
- A-A Impostazione dell'area di ripresa della telecamera 16
- A-B Mapping 16
- A-C Path-Planning 17
- A-D Rilocalizzazione 17
- A-E Tecniche per la simulazione del Mapping 17

Appendice B: Apparato Strumentale 17

Riferimenti bibliografici 19

I. INTRODUZIONE

Il problema è la navigazione autonoma di robot in ambiente sconosciuto. Si hanno a disposizione N robot dotati di sensori dai quali si può trarre informazione dell'ambiente circostante, ovvero essenzialmente la presenza o meno di ostacoli, e un robot privo di sensori (cieco) che si vuole far viaggiare fino ad un punto obbiettivo di coordinate note. L'idea è quella di costruire tramite i sensori una mappa del territorio, e di pianificare su questa il percorso per l'attore cieco.

Per ottenere lo scopo si è adottata una filosofia progettuale che si può riassumere in tre punti fondamentali.

- **Realismo:** utilizzare tutti gli strumenti a disposizione per mantenere il progetto il più generale possibile, limitando l'inserimento di sensori simulati;
- **Funzionalità:** creare un sistema effettivamente funzionante;
- **Semplicità:** cominciare da tecniche semplici, implementabili praticamente, da raffinare solo dopo averne verificato il funzionamento per ottenere risultati migliori.

Questi criteri hanno indirizzato ogni scelta, in particolare l'utilizzo della telecamera montata sui robot come sensore di visione, piuttosto che simulare altri sensori non presenti.

Il problema affrontato risulta piuttosto particolare, con problematiche relative alla particolare strumentazione utilizzata. Il sistema globale di controllo e l'architettura generale sono state quindi ideate *ad hoc*, mentre per quanto riguarda la soluzioni delle singole tematiche ci si è potuti confrontare con una vasta letteratura, per maggiori riferimenti a riguardo si rimanda alle singole sezioni.

Il raggiungimento dello scopo richiede la soluzione dei seguenti problemi fondamentali.

Il problema del *mapping* consiste sostanzialmente nel trovare una trasformazione dall'immagine presa dalla telecamera del robot al piano di lavoro visto dall'alto, cioè nella costruzione di un'usuale mappa del territorio. Per la rappresentazione della mappa ci si è affidati a tecniche ben conosciute in letteratura mentre l'utilizzo della telecamera ha richiesto un contributo originale per una precisa e veloce ricostruzione dell'ambiente.

L'*exploration* si propone di trovare un metodo per decidere qual è la zona più interessante da esplorare, cercando una soluzione possibilmente ottima secondo qualche criterio, e considerando lo scopo di trovare una strada che congiunga la posizione dell'attore cieco all'obbiettivo. L'algoritmo di esplorazione è stato elaborato a partire da alcune proposte già implementate, adattandolo alla particolare tecnica di costruzione della mappa ed alla focalizzazione verso la zona dell'obiettivo a cui condurre l'attore cieco.

Il *path-planning* vuole identificare una traiettoria tra due punti in modo da evitare gli ostacoli che possono trovarsi in mezzo, e generare un segnale di riferimento di posizione da far inseguire al robot per spostarsi. Per fare ciò si è sfruttato un noto algoritmo di ricerca di cammini minimi presente in letteratura.

Il problema della *rilocalizzazione* tramite GPS virtuale è sostanzialmente un problema di visione computazionale: individuare in un'immagine il robot, e trasformarne le cordinate in metri per stimarne la posizione sulla mappa.

Infine, un problema noto nei controlli automatici di sistemi meccanici che è il *trajectory-tracking*: controllare in retroazione il robot in modo da inseguire un riferimento dato, nel nostro caso in posizione.

Si presenta ora la soluzione nella sua globalità, nelle sezioni successive si presentano invece le soluzione specifiche per i problemi elencati.

II. Panoramica Generale

Si devono coordinare $N+1$ robot, N nella loro azione di esplorazione, e uno la cui azione principale è il moto verso l'obbiettivo. Gli N robot esploratori verranno denominati *Explorer*, l'altro *Actor*.

Tutti i robot sono dotati di sensori di posizione, cioè di encoder posti sugli alberi dei due motori dai quali si può trarre facilmente informazione sulla posizione e sulla direzione dei robot a partire da una condizione iniziale nota. Questi sono gli unici sensori a disposizione dell'*Actor*, mentre gli *Explorer* hanno un sensore visivo, una piccola telecamera, dalla quale si traggono le informazioni per ricostruire l'ambiente circostante. I robot non hanno la capacità di comunicare direttamente tra loro, ma comunicano con un'unità di calcolo centralizzata che raccoglie le informazioni provenienti dai sensori, le elaborara, e trasmette nuovi ordini ai robot. Il sistema di controllo è quindi di tipo *centralizzato*.

Si suppone tutti i robot siano posizionati inizialmente in un'area priva di ostacoli (ovviamente). L'idea è semplice. Ogni singolo robot *Explorer*, dalla posizione di partenza in cui si trova dovrebbe:

1) fare una foto e aggiornare la mappa (*Mapping*);
2) eseguire l'algoritmo di esplorazione, che porgerà una nuova destinazione e orientamento verso cui esplorare (*Exploration*);
3) eseguire l'algoritmo di *Path-Planning*;
4) eseguire il moto controllato in retroazione sugli encoder (*Moto*);
5) correggere la propria posizione grazie al GPS virtuale (*Rilocalizzare*);
6) girare nella direzione individuata al punto 3) (*Girare*);
7) ricominciare dal punto 1).

e proseguire così finchè non viene avvistato l'obbiettivo e ci si è accertati che è raggiungibile. L'actor intanto dovrebbe

1) aspettare aggiornamenti della mappa (*Wait*);
2) calcolare la strada più breve con l'informazione nota a questa a punto, supponendo lo spazio ancora ignoto privo di ostacoli, e decidere se partire (*Path-Planning*)
3) se si parte, muoversi fino a dove la mappa è conosciuta e tornare al punto 1) (*Moto*)

e proseguire finchè non si giunge all'obbiettivo.

Per *Mapping* si intende la costruzione della mappa attraverso le successive misure raccolte dai robot. Si è scelto di rappresentare la mappa mediante le *occupancy grid cell*, si veda [1], cioè la mappa è costituita da una matrice in cui ogni cella corrisponde ad un'area di 1 cm^2 ed il valore corrisponde alla probabilità che una cella sia occupata. Le misure sono ottenute elaborando l'informazione della telecamera dei robot: nell'immagine si individuano gli spigoli orizzontali degli ostacoli tramite un opportuno algoritmo di visione. Dalla loro posizione nel piano immagine si risale alla loro posizione nel piano reale attraverso alcune nozioni di geometria proiettiva. Intuitivamente più il bordo si trova in alto nell'immagine, più l'ostacolo sarà distante dal robot. Dalla colonna dell'immagine in cui compare l'ostacolo si possono ricavare informazioni sulla sua direzione rispetto al robot. Banalmente, un ostacolo che si trova sulla destra dell'immagine sarà sulla parte destra del cono di visione del robot.

La fase di *Exploration* consiste nell'assegnare ad ogni robot la successiva posizione da cui effettuare la fotografia. In primo luogo si individuano sulla mappa le *frontier cells*, cioè quelle celle esplorate che si trovano vicine alla regione inesplorata. Queste celle vengono ordinate in base ad un indice che tiene conto della loro vicinanza all'obiettivo e della distanza dalla posizione attuale. Inoltre nel caso di più robot si va ad aggiungere un termine che mantenga i robot ad una distanza minima tra di loro. Ad ogni robot viene quindi assegnato come punto di esplorazione successivo il punto che minimizza quest'indice.

Il *Path-Planning* serve ad individuare la traiettoria che ogni robot deve percorrere per andare da un punto ad un altro, evitando gli ostacoli e passando solo per le regioni esplorate. Si è implementato l'algoritmo *D**, un algoritmo di ricerca di cammini minimi su un grafo. Esso risulta molto efficace anche per l'individuazione della traiettoria che porti l'*Actor* all'obiettivo, in quando aggiorna di volta in volta il percorso in base all nuove osservazioni degli *Explorer*.

Infine la parte di *Rilocalizzazione* serve a correggere la posizione del robot. Tramite un algoritmo di *blob-detection* si va ad individuare il robot cercato sull'immagine ottenuta dal GPS virtuale. Quindi con una veloce normalizzazione se ne ottengono le coordinate nel sistema di riferimento assoluto.

Nella prossima sezione sarà illustrato il sistema di controllo globale che gestisce i vari robot ed assegna loro le operazioni da compiere. In quelle successive saranno affrontati nel dettaglio i problemi relativi alle singole operazione sopra presentate.

III. Il Sistema di Controllo

Le azioni computazionalmente onerose che gli *Explorer* devono compiere sono sostanzialmente quattro: la costruzione della mappa, la scelta del punto di esplorazione successivo (e della direzione verso cui guardare), la pianificazione del percorso dalla posizione attuale a quella nuova, l'aggiornamento della posizione dei robot tramite GPS virtuale. Questo si rivela necessario perchè l'odometria ricavata dagli encoder dei motori soffre del problema della deriva: piccoli errori nei sensori portano dopo tempi lunghi (ma non tanto) ad un inevitabile errore di posizione. Nel caso dei nostri robot, gli *e-puck*, tale errore si aggira intorno ai 5 cm su un moto di 40-50 cm. A questo si potrebbe rimediare sfruttando le informazioni sull'ambiente esterno date dai sensori visivi, ma questo è un problema noto e complicato, che si trova in letteratura sotto il nome di *Slam* (Simulataneous Localization And Mapping). Qui invece si suppone la posizione dei robot essere nota, cioè si finge un sistema tipo GPS attraverso una telecamera montata sopra l'ambiente di lavoro, dalla quale si ricava la posizione dei robot. Oltre a queste quattro azioni, ogni robot deve ovviamente potersi muovere, ma questa azione non richiede tempi di calcolo impegnativi. Inoltre, è comodo introdurre una distinzione tra l'azione di moto vero e proprio e quella del girarsi soltanto attorno al proprio asse verticale, mantenendo la stessa posizione, perchè questi due tipi di moto verranno gestiti in maniera diversa.

L'unica azione computazionalmente onerosa che deve compiere l'*actor* è quella della ricerca del cammino minimo dalla sua posizione attuale a quella dell'obbiettivo, oltre naturalmente all'aggiornamento anche della sua posizone tramite GPS virtuale.

Il computer centrale è dotato del software MATLAB, che è il responsabile di ogni tipo di calcolo, e attraverso il quale si sono implementati i diversi algoritmi. Grazie al pacchetto *ePic2*, disponibile sul sito ufficiale degli *e-puck*[1], è possibile comunicare con i robot attraverso MATLAB, acquisendo le informazioni dai sensori, dando i segnali di comando ai motori e via dicendo. Il problema serio che si pone subito sono i tempi di calcolo e di trasmissione tra i robot e il computer. Infatti, MATLAB, seppur molto comodo e intuitivo, non è certo famoso per la sua rapidità di esecuzione, e altrettanto lento è il protocollo *bluetooth* utilizzato per la trasmissione. Queste lentezze comportano inevitabilmente un tempo di campionamento molto alto.

Per questo motivo, si è organizzato il sistema in modo che ad ogni passo ogni robot compia una sola tra le azioni descritte pocanzi. Si assegna uno *stato* ai robot. Ad ogni stato corrisponde un'azione, e ad ogni passo si controllano gli stati dei singoli robot e si fa eseguire loro la rispettiva azione. In questo modo, ad ogni passo si dovranno compiere N azioni definite dagli stati degli N *Explorer*, più una definita dallo stato dell'*Actor*.

Di seguito, le definizione degli stati degli *Explrer*, con una descrizione dettagliata delle operazioni da eseguire in ognuno di essi.

[1] `http://www.e-puck.org`

1. MAPPING: il robot deve fare una foto, elaborarla, mandarla al pc, il pc costruisce la mappa.
2. EXPLORATION: il pc deve scegliere un nuovo punto da esplorare.
3. PATH-PLANNING: il pc deve individuare una strada dal punto dove si trova il robot, al punto di osservazione definito da 2.EXPLORATION.
4. GIRARE: il pc calcola i segnali da mandare al robot per farlo girare e li trasmette, il robot gira.
5. RILOCALIZZARE: si fa una foto dell'ambiente, il pc la elabora e trasmette la nuova posizione al robot.
6. MOTO: il pc calcola i segnali da mandare al robot per fargli seguire la traiettoria definita in 3.PATH-PLANNING e li trasmette.

A questi è utile aggiungere

0. WAIT: stato di attesa.

L' azione principale che l'*Actor* deve compiere è solo quella di pianificare il proprio percorso dalla posizione attuale a quella del *goal* e decidere quando partire. Tutto ciò può essere inserito in uno stato simile al 3 dell'elenco sopra, con la differenza che la traiettoria va pianificata fino al punto obbiettivo, e l'aggiunta di una decisione su quando partire. Non essendo chiaramente possibile definire una condizione ottima di partenza, in quanto non può essere nota a priori tutta la strada se non quando l'esplorazione si è compiuta, si è deciso di far partire l'*Actor* quando la strada nota supera il doppio della strada minima mancante per raggiungere il *goal*. Questa informazione è facilmente ricavabile dalla natura della soluzione al problema di *Path-Planning* utilizzata. L'*Actor* deve poi saper rilocalizzare la propria posizione, girarsi, e muoversi. Ne consegue che i suoi stati possono essere ridotti agli ultimi quattro dell'elenco, con lo stato 3 modificato come detto.

La suddivisione degli stati è stata in fatta in modo da avere un tempo di esecuzione di ogni stato più o meno uniforme, e la scelta del tempo di campionamento T deve soddisfare $T > T_{ex}/(N+1)$, dove T_{ex} è il tempo massimo di esecuzione dello stato più lento. É importante mantenere una suddivisione uniforme del tempo nel caso ci sia almeno un robot che si sta muovendo, in modo da poter acquisire e comandare il robot a intervalli regolari. É chiaro invece che da un punto di vista logico gli stati 1, 2 e 3 potrebbero essere inglobati in un unico stato, ma il suo tempo di esecuzione risulterebbe troppo alto. Un tempo di campionamento piccolo implica migliori prestazioni in termini di velocità di movimento dei robot, in quanto si ha un maggiore controllo, e dunque si vorrebbe mantenerlo il più basso possibile. La suddivisione dei calcoli in più stati è una soluzione a questa problematica. Per l'utilizzo di due *Explorer* ed un *Actor* il tempo di campionamento minimo è pari a 2 secondi.

Ogni stato comporta ovviamente delle problematiche da risolvere, che sono oggetto di studio delle sezioni successive. Supponiamo ora di saper risolvere tutte queste problematiche e presentiamo un algoritmo di controllo che esegua lo scopo introdotto precedentemente: esplorare il territorio fino a trovare un percorso privo di ostacoli dalla posizione dell'*Actor* ad una

posizione obbiettivo le cui coordinate sono note a priori, e contemporaneamente farvi giungere l'*Actor*.

All'idea principale esposta in sezione II si aggiungono una serie di aspetti tecnici che complicano leggermente questo schema. Primo fra tutti: la visuale della telecamera degli *Explorer* è troppo stretta. Si rende necessario eseguire almeno una sequenza di 3 foto per coprire un area ragionevole di territorio.

Ci sono poi alcune operazioni più "delicate" di altre. Come già accennato, durante il moto è importante un sincronismo del tempo al quale si accede alle informazioni degli encoder e si calcola il comando da dare al robot. Infatti, il sistema calcola tale comando supponendo un tempo di campionamento regolare, T, durante il quale il segnale di controllo viene mantenuto costante. Naturalmente, eventuali imprecisioni sulla posizione all'istante successivo verranno corrette dalla retroazione, ma rimane importante che l'intervallo sia circa quello di campionamento. Per questo motivo, si sceglie di "servire" i robot ad ogni passo a partire da quello che ha stato maggiore, che corrisponde allo stato di moto, in modo da non risentire dei ritardi causati dai calcoli necessari agli altri stati. Per semplificare il moto, inoltre, si fa in modo che il robot, prima di partire, si giri nella direzione giusta, cioè quella individuata dai primi passi del percorso.

Poi, gli stati dell'*Actor* includono azioni leggermente diverse da quelli degli *Explorer*, e il loro ordine di esecuzione è pure diverso. Li si distingue mantenendogli lo stesso numero intero, ma cambiato di segno. Ad esempio, lo stato PATH-PLANNING dell'*Actor* verrà contrassegnato con -3. Anche per l'*Actor* è necessaria una rilocalizzazione tramite GPS virtuale, affinchè la posizione presunta del robot non sia errata a tal punto da farlo scontrare contro un ostacolo (o un altro robot). Il moto viene quindi interrotto a intervalli regolari per correggere la posizione del robot, a cui segue ovviamente una nuova pianificazione. Inoltre, i tempi di calcolo del cammino minimo da parte dell'*Actor* si sono rivelati decisamente troppo lunghi. Allora, si è fatto in modo che il calcolo venga fatto soltanto in una situazione in cui tutti i robot sono fermi, introducendo un ulteriore stato di "attesa del momento opportuno" per l'*Actor*. Questo non è che cambi molto le cose, diciamo che al peggio il robot scopre leggermente in ritardo la condizione di partenza postagli, cioè, ricordiamo, che la strada nota è più di metà della strada minima mancante.

Infine, manca una gestione di una situazione di emergenza in cui due robot si scontrano. In verità, per come è fatto l'algoritmo di esplorazione, non può accadere che due *Explorer* si scontrino, perchè esso provvede a dividere l'area di ricerca tra i due robot, che si mantengono dunque distanti. Invece, in una situazione finale in cui tutti i robot possono trovarsi vicino al *goal*, è più probabile uno scontro tra *Explorer* e *Actor*. Per evitare questo, si fa in modo che una volta visto l'obbiettivo, e constatata la presenza di una effettiva traiettoria possibile per raggiungerlo, si sposta l'*Explorer* in questione in una direzione opposta a quella di partenza dell'*Actor*, cioè "dietro" l'obiettivo, in modo che l'*Actor*, dovendo necessariamente essere più indietro, non si scontri. Però, non sempre esiste un punto conosciuto tanto distante da essere certi di evitare lo scontro. A questo, si è posto rimedio "occupando" nella mappa un'area corrispondente a quella del robot, proprio come fosse un ostacolo, in modo che l'algoritmo di pianificazione lo eviti, essendo certi che questo non si sposterà più in quanto ha concluso il suo compito.

Da tutte queste considerazioni, si è tratto il seguente algoritmo di controllo, leggibile in coppia con la tabella I, che illustra uno schema di transizione tra lo stato corrente e il successivo elencando le condizioni su quale stato futuro scegliere.

Finchè l'*Actor* non è giunto a destinazione

1. ordina decrescentemente i robot secondo il loro stato (in modulo);
2. **Per ogni** robot
    3. Esegui le azioni del suo corrispondente stato;
    4. Aggiorna lo stato del robot per la successiva iterazione secondo lo schema di tabella I;

Fine

Algoritmo 1: Algoritmo di controllo

IV. MAPPING

In questa parte si tratteranno le problematiche e le soluzioni proposte riguardo il problema del mapping, cioè della realizzazione della mappa di un ambiente attraverso misure successive da parte di uno o più sensori, posizionati sui diversi robot a disposizione. Nel nostro caso la consegna iniziale del progetto prevedeva l'utilizzo di un sensore virtuale realizzato a partire dall'immagine ottenuta dalla telecamera posta sopra la pedana del NAVLAB. Dall'immagine totale si ricava un cerchio od un settore circolare centrato sull'*e-puck*, simulando così un radar. Visto che i robot a nostra disposizione sono però dotati di fotocamera digitale abbiamo deciso di utilizzare questo come sensore. Gli stessi robot dispongono anche di 8 sensori di prossimità che potrebbero essere utili allo scopo. In seguito a qualche prova sperimentale le loro misure si sono dimostrate molto rumorose e soprattutto con un raggio massimo di sensibilità inferiore ai 5 cm. Si è pertanto ritenuto di utilizzare come unico sensore la telecamera di cui ogni *e-puck* è dotato. L'utilizzo della telecamera ha comportato però una notevole mole lavoro per essere sfruttata al meglio secondo i nostri scopi. Le misure successive ricavate dai diversi robot nelle varie posizioni devono essere poi rielaborate e fuse insieme per poter andare a costruire una mappa più precisa possibile della zona esplorata.

A. Stato dell'arte

Il problema del mapping è affrontato in numerosi testi presenti in letteratura. In tutti il problema viene affrontato prima in maniera generale, ma poi ci si restringe alle particolari condizioni di lavoro ed agli strumenti utilizzati. In particolare in [2] sono presentati alcuni casi di map-building e navigazione risolti. Nessuno presenta tutte le caratteristiche del nostro problema; l'uso di una telecamera come sensore non è molto diffuso per il mapping 2D, però si sono trovati spunti utili per decidere il tipo di approccio da usare.

Tabella I: Schema di transizione tra stati

STATO ATTUALE	STATO SUCCESSIVO	Condizione di scelta stato successivo
Per gli *Explorer*		
1. MAPPING	2.EXPLORATION	*se si sono fatte le 3 foto*
	3.PATH-PLANNING	*se l'obbiettivo è in vista, per essere sicuri che esiste una strada*
	4.GIRARE	*se si devono fare ancora foto per completare la tripletta*
2.EXPLORATION	3.PATH-PLANNING	*se il nuovo punto di esplorazione è diverso dalla posizione corrente*
	4.GIRARE	*se il punto è lo stesso, cambia solo la direzione*
3.PATH-PLANNING	3.PATH-PLANNING	*se si è finita l'esplorazione, per pianificare di spostarsi*
	4.GIRARE	*se la direzione iniziale del moto è diversa da quella corrente*
	6.MOTO	*se non serve girarsi perchè si è già dritti*
4.GIRARE	1.MAPPING	*se non ho finito la tripla oppure ho finito il moto e la localizzazione*
	4.GIRARE	*se non avevo finito di girare in un passo*
	6.MOTO	*se devo partire verso un nuovo punto*
5.RILOCALIZZARE	4.GIRARE	*in generale, se non ho finito, devo fare una foto*
	0.WAIT	*se ho finito: ho visto, so che si può andare, mi sono spostato*
6.MOTO	6.MOTO	*se non sono ancora arrivato*
	5.RILOCALIZZARE	*finito il moto*
0.WAIT	0.WAIT	*stato di nullafacenza*
Per l'*Actor*		
-1.MAP WAIT	-1.MAP WAIT	*se non ho nuove informazioni sulla mappa*
	-2.PLANNING WAIT	*se ho nuove informazioni sulla mappa*
-2.PLANINNG WAIT	-2.PLANNING WAIT	*se qualche* Explorer *è in stato* 6.MOTO
	-3.PATH-PLANNING	*se nessun* Explorer *è in stato* 6.MOTO
-3.PATH-PLANINNG	-1.MAP WAIT	*se non so abbastanza percorso*
	-6.MOTO	*se so abbastanza percorso*
-4.GIRARE	-4.GIRARE	*se non avevo finito di girare in un passo*
	-6.MOTO	*se devo partire verso l'obbiettivo*
-5.RILOCALIZZARE	-2.PLANNING WAIT	*sempre*
-6.MOTO	-6.MOTO	*se non sono ancora arrivato*
	-5.RILOCALIZZARE	*finito il moto*

a) Tipologia e dimensione della mappa: La prima scelta da effettuarsi riguarda la modalità di rappresentazione della mappa. Si è riscontrato in [1] un utilissimo riepilogo delle possibili tecniche, dei loro vantaggi e svantaggi. In linea con la filosofia progettuale già presentata si è cercato innanzitutto una rappresentazione che fosse leggera da un posto di vista computazionale e facilmente implementabile in MATLAB. Le ipotesi del problema prevedono la conoscenza della posizione e dell'orientamento del robot ad ogni istante, pertanto non è richiesto l'uso di stimatori come il filtro di Kalman. Le possibilità sono essenzialmente due: o si utilizza una rappresentazioni con nodi e grafi o si utilizza una rappresentazione *occupancy grid cells* cioè una griglia in cui il valore delle singole celle indica la probabilità che essa sia occupata. La prima tecnica risulterebbe decisamente più pratica per la successiva fase di esplorazione e pianificazione del moto in quanto consente di individuare facilmente percorsi liberi e transitabili da parte del robot. Inoltre risultando più sintetica comporta una minore occupazione della memoria. D'altro canto essa risulta però complessa da un punto di vista implementativo. Si è scelto quindi di adottare la tecnica della *occupancy grid cells*, adatta al linguaggio MATLAB che useremo, visto che si tratta in sostanza di una matrice, e più intuitiva. Come detto in [1] vengono anche suggerite le diverse tecniche applicabili per l'assegnazione delle probabilità di occupazione alle singole celle. Tipicamente, considerando ambienti statici, la probabilità di occupazione di una cella x data una misura m, $p(x|m)$, viene calcolata tramite la regola di Bayes:

$$p(x|m) = \eta p(m|x)p(x)$$

dove con $p(m|x)$ si indica la probabilità di generare una certa misura m sotto l'ipotesi x, $p(x)$ costituisce la probabilità di occupazione della cella a priori, in genere pari ad $\frac{1}{2}$, mentre η è una fattore di normalizzazione che rende $p(x|m)$ effettivamente una densità di probabilità. Questo aproccio probabilistico sviluppabile poi in diverse forme serve sostanzialmente ad irrobustire la costruzione della mappa agli errori di posizione e di misura, ed è pertanto necessario problemi quali lo SLAM, in cui la posizione è incognita. Nel nostro caso però viene garantita la conoscenza della posizione con errore minimo e ci si è concentrati, con buoni risultati nel rendere il più preciso possibile il sensore utilizzato.

B. Tipologia e dimensione della mappa

Per cercare di semplificare il problema e giungere rapidamente ad una implementazione completa del sistema si è deciso di iniziare assegnando alle singole celle di occupazione solo tre valori:

- 0: nel caso una cella sia ritenuta libera
- 1: nel caso una cella sia ritenuta occupata
- ∞: nel caso nessuna misurazione abbia riguardato quella cella

A prima vista questa scelta può apparire rischiosa e poco affidabile, mentre in fase di simulazione e di sperimentazione in NAVLAB si è dimostrata efficace e sufficientemente robusta. Questo è dovuto in particolar modo alla tecnica decisamente conservativa con cui vengono analizzate le misure (cioè le foto) delle telecamere dei robot. E' chiaro che tale scelta rappresenta solamente un primo approccio al problema, essendo

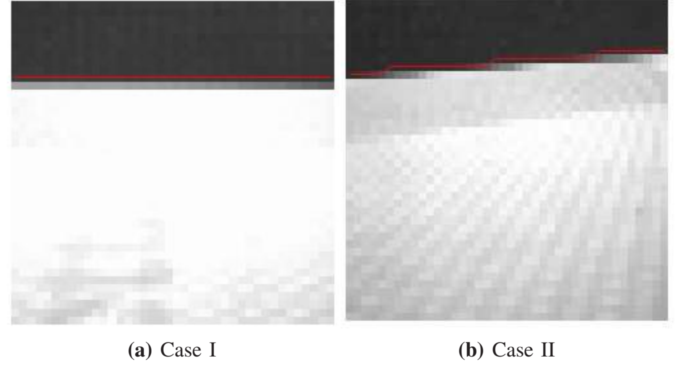
però che nel corso del lavoro essa non si è mai rivelata, al contrario di altri aspetti, un punto critico è stata mantenuta invariata, ed anzi la sua semplicità ha garantito una velocità di calcolo determinante nell'applicazione real-time con MATLAB. Si tratta ora di dimensionare adeguatamente la matrice che andrà a rappresentare la mappa. Dopo alcune verifiche sulla precisione della telecamera dei robot, del controllo del loro e delle dimensioni degli ostacoli che si intendeva utilizzare si è deciso di utilizzare una griglia con celle quadrate di lato 1 cm. In definitiva quindi la mappa che useremo sarà costituita da una matrice di dimensioni 192 $\times$ 256, in cui ogni elemento individua un'area di un centimetro quadrato.

C. Rilevamento degli ostacoli

Poichè si vuole ricostruire una mappa 2D dell'ambiente sufficiente ricavare dalle foto effettuate dalla telecamera gli spigoli orizzantali degli ostacoli adiacenti al pavimento. Gli algoritmi di *edge detection* sono numerosissimi in letteratura, si veda per esempio Canny o Sobel, le cui funzioni sono già implementate anche in MATLAB. Per ridurre il tempo di trasmissione dell'immagine abbiamo pensato di fare in modo che l'eventuale bordo venisse individuato sfruttando le capacità computazionali del microprocessore di cui è dotato l'*e-puck*, così che lo stesso trasmetta solamente un vettore che indichi per ogni colonna dell'immagine la posizione del bordo. Si è dovuto dunque implementare l'algoritmo direttamte sull'*e-puck*. Viste le sue limitate capacità di calcolo e le difficoltà implementative che si sarebbero incontrate si è deciso di individuare il bordo degli ostacoli in maniera più semplice, sfruttando qualche informazione aggiuntiva sugli ostacoli utilizzati. Si sono utilizzati ostacoli neri di modo da avere un chiaro contrasto con il pavimento bianco. Quindi si va a scorrere ogni colonna dal basso verso l'alto e non appena si individuano due pixel consecutivi di intensità inferiore ad una certa soglia si segnala quello come bordo dell'ostacolo. Con questa tecnica quindi si riduce di 1/40 la quantità di byte tasmessa dall'*e-puck* al computer in seguito ad ogni fotografia. L'implementazione dell'algoritmo in C non ha presentato particolari problemi, al di là del fatto di inserirla correttamente all'interno del codice già presente. Una taratura della soglia adeguata alla luminosità esterna garantisce il funzionamento dell'algoritmo come mostrato in figura 1.

D. Costruzione della mappa

Sostanzialmente l'uso della telecamera come sensore per ricostruire la mappa richiede di definire una trasformazione Γ che associ ai punti (pixel) dell'immagine ripresa dalla telecamera i punti del piano su cui si muove il robot: $\Gamma : \mathbb{Z}^2 \mapsto \mathbb{R}^2$, o più precisamente nel nostro caso $\Gamma : [1, X_p] \times [1, Y_p] \mapsto R^2$, avendo a che fare con un'immagine di risoluzione $X_p \times Y_p$ pixel. In una prima fase si andrà a considerare un sistema di riferimento del piano solidale alla camera, e quindi all'*e-puck* stesso, conoscendo poi la posizione e l'orientamento del robot si ottiene facilmente una corrispondenza con il piano definito nel sistema di riferimento assoluto. Se si considera il modello della fotocamera *pin hole* riportato in figura 2 si ricava la



(a) Case I **(b)** Case II

Figura 1: Funzionamento dell'algoritmo di individuazione del bordo implementato direttamente sul microprocessore dell'*e-puck*. In effetti la quantità trasmessa dal robot è il solo vettore rappresentato in rosso.

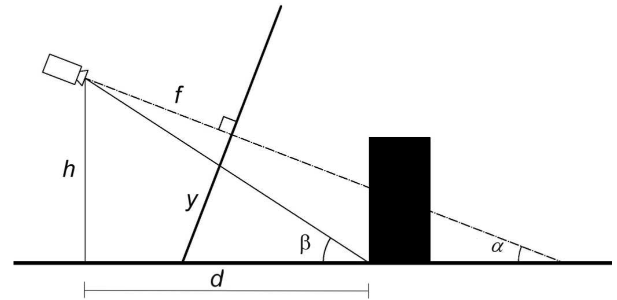


Figura 2: Geometria della camera

seguente relazione:

$$\frac{d}{h} = \frac{d - f \frac{\cos\beta}{\cos(\beta-\alpha)}}{y_p \cos\alpha} \tag{1}$$

dove si è indicato con f il fuoco della camera, con h la sua altezza rispetto al piano, con d la distanza dall'ostacolo, con y l'ordinata del punto del piano immagine in cui viene visto il bordo inferiore dell'ostacolo, con α l'angolo che l'asse ottico forma con il piano ed infine $\beta = \arctan \frac{h}{d}$. Nel caso in esame si ha $\alpha \approx 0$ [2], e dunque si può riscrivere la (1) come:

$$\frac{d}{h} = \frac{d - f}{y_p} \tag{2}$$

che esplicitando rispetto a d diventa:

$$d = \frac{fh}{h - y_p}. \tag{3}$$

Se si orientano gli assi del piano cui appartiene la pedana come mostrato in figura 3 si è dunque definita una corrispondenza tra le ordinate sul piano immagine, y_p, e le ordinate $y_r = d$ su questo piano.

Poichè ad ogni colonna dell'immagine corrisponde un particolare settore angolare sul piano reale risulta conveniente utilizzare le coordinate polari, in quanto così facendo risulterà immediato associare ad ogni colonna dell'immagine il corrispondente settore circolare e quindi la distanza a cui si trova

[2]In quanto la telecamera montata sugli *e-puck* ha l'asse ottico circa orizzontale

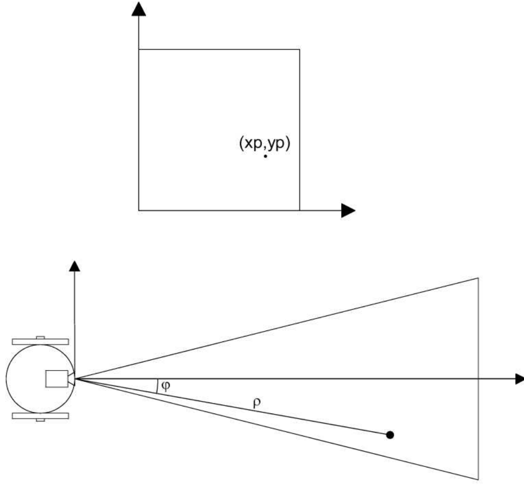


Figura 3: Geometria della camera

l'ostacolo. Detto θ l'angolo di visione dell'*e-puck*, x_p l'ascissa del piano immagine di cui si vuole ricavare la fase ϕ, si ha:

$$\phi = \arctan\left(\frac{\left(\frac{X_p}{2} - x_p\right)}{\frac{X_p}{2}} \tan\left(\frac{\theta}{2}\right)\right) \tag{4}$$

Dunque ad ogni punto del piano immagine (x_p, y_p) è possibile associare il punto (ρ, ϕ) del piano reale con sistema di riferimento solidale alla telecamera, dove ϕ è calcolato secondo la (4) mentre $\rho = \frac{d}{\cos\phi}$, dove d è ricavato dalla (3).

Chiaramente infine tale sistema di riferimento, solidale all'*e-puck* stesso, dovrà essere centrato non sulla telecamera ma sul centro del robot, visto che di questo sono note le coordinate, e la fase corrisponderà con lo sfasamento rispetto all'orientazione del robot stesso, cioè i punti di fase nulla sono quelli che si trovano lungo l'asse ottico della telecamera[3].

Si vogliono quindi calcolare le coordinate polari in seguito alla traslazione del centro del sistema di riferimento. Sia l la distanza della camera dal centro del robot, allora come si vede in figura 3, al punto (ρ, ϕ), corrisponde il punto (σ, ξ) secondo la relazione:

$$\begin{aligned} \xi &= \arctan\left(\frac{\rho\sin\phi}{\rho\cos\phi + l}\right) \\ \sigma &= \frac{\rho\cos\phi + l}{\cos\xi} \end{aligned} \tag{5}$$

Riassumendo si è così definita la trasformazione $\bar{\Gamma} : [1, X_p] \times [1, Y_p] \mapsto R^2, (x_p, y_p) \to (\sigma, \xi)$, l'ultimo passaggio consiste nel riferirsi al sistema di riferimento assoluto conoscendo la posizione e l'orientazione del robot, si tratta cioè di applicare un semplice cambio di coordinate. Questa semplice viene implementata in maniera un po' particolare nel nostro caso, per diminuire i tempi di calcolo ed adattarsi meglio alla tipologia di mappa scelta. Per prima cosa si esprimono i punti rispetto ad un sistema di riferimento centrato nello stesso punto ma orientato secondo il sistema di riferimento assoluto. Sia

[3] Più precisamente sulla proiezione dell'asse ottico sulla pedana.

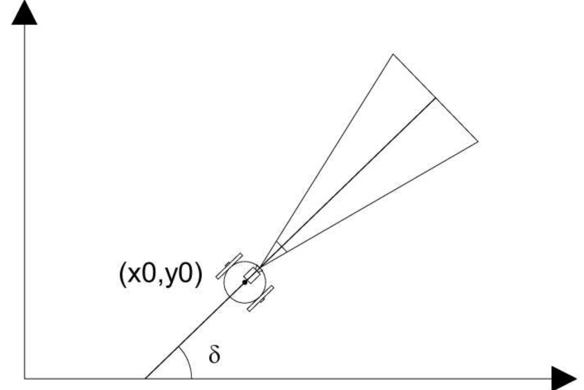


Figura 4: Passaggio al sistema di riferimento assoluto

δ l'orientazione dell'*e-puck*, i nuovi punti avranno modulo $\tilde{\sigma} = \sigma$ e fase $\tilde{\xi} = \xi + \delta$. Detta (x_0, y_0) la posizione del centro dell'*e-puck* al punto (x_p, y_p) sul piano immagine si andrà a far corrispondere il punto (x, y), sul piano solidale al pavimento e rispetto al sistema di riferimento assoluto, così definito:

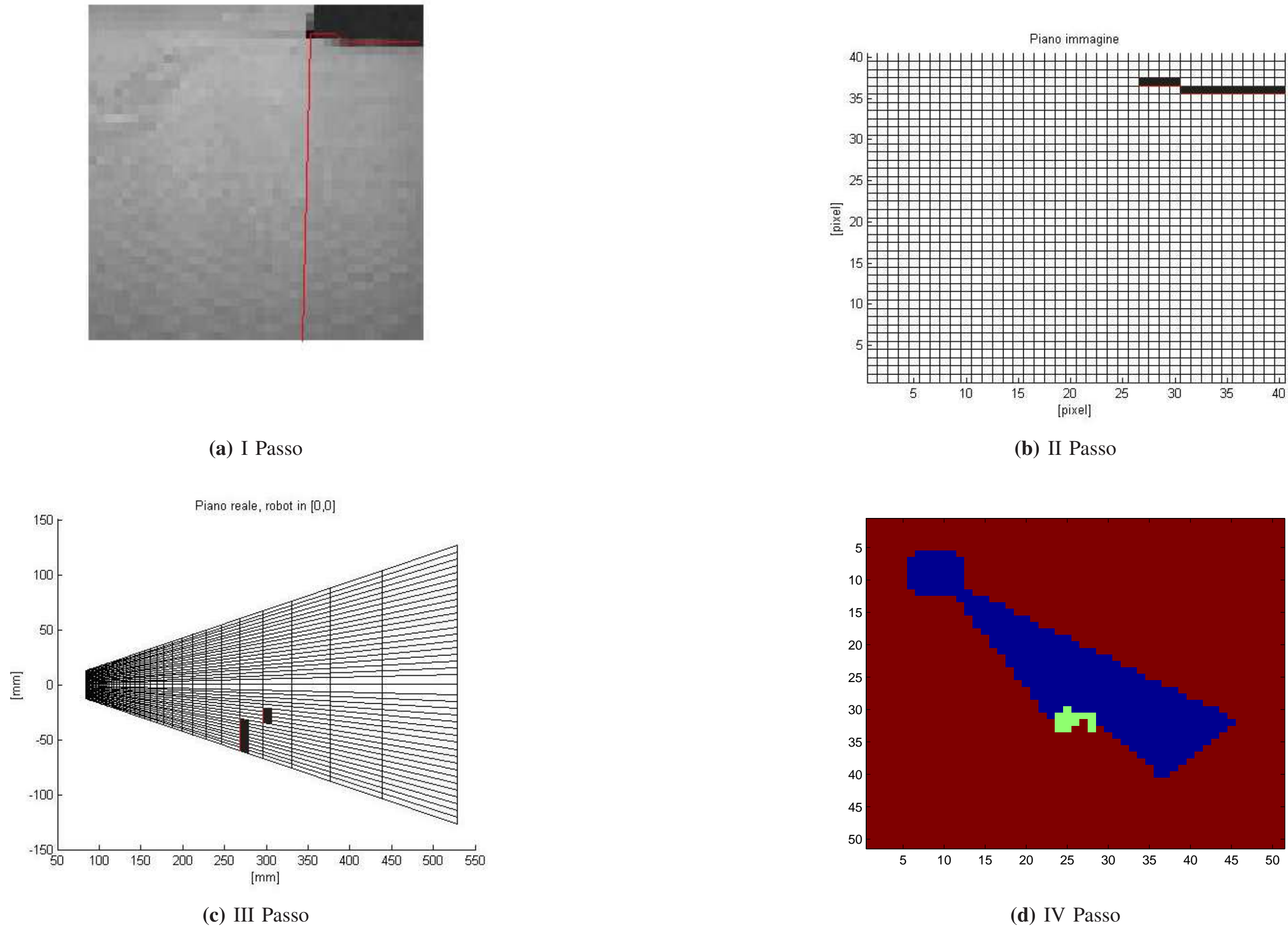
$$\begin{aligned} x &= x_0 + \sigma\cos\tilde{\xi} \\ y &= y_0 + \sigma\sin\tilde{\xi}. \end{aligned} \tag{6}$$

E. Aggiornamento della mappa

Il problema dell'aggiornamento della mappa data una misura si può formulare come segue: si vuole stabilire un operatore che data la k-esima misura $m(k)$ e la mappa aggiornata alla misura $(k-1)$-esima $M(k-1)$, restituisca la mappa globale aggiornata $M(k)$:

$$\Phi : (M(k-1), m(k)) \mapsto M(k)$$

Ovviamente la definizione di questo funzione non è univoca, essendo essa intrinsecamente legata al tipo di misura $m(k)$, al sensore utilizzato ed al rumore di misura, all'ambiente che si vuole mappare ed allo scopo che si intende raggiungere; per ulteriori dettagli si rimanda nuovamente a [1]. Nel nostro caso come già ricordato si è evitato l'approccio probabilistico, pertanto data la $M(k-1)$ e la $m(k)$, si tratta di decidere che scelta adottare nel caso di celle (x, y) con valori differenti sulle due mappe. Una prima scelta è quella di assegnare ad $M(k)$ i valori di $M(k-1)$, e quindi i valori di $m(k)$ per le celle che in $M(k-1)$ valevano ∞. Rimane la scelta per le celle che hanno valore non ∞ e diverso in $M(k-1)$ ed $m(k)$, celle cioè che risultano libere in una delle due mappe ed occupate nell'altra. Esse vengono impostate come libere, cioè con probabilità di occupazione pari a 0. Questa decisione, che a prima vista può apparire "rischiosa", è in realtà giustificata, ed in qualche modo richiesta, dalla tecnica di costruzione della misura. Infatti come detto in precedenza in presenza di ostacoli sulla fotografia essi vengono posti nel piano reale al minimo



(a) I Passo

(b) II Passo

(c) III Passo

(d) IV Passo

Figura 5: Esempio di costruzione di una misura a partire da una foto effettuata dal robot

della distanza a cui corrisponde ciascun pixel[4]. Poichè quando gli ostacoli visti sono lontani si ha una maggiore incertezza sulla misura, accadrà quasi sempre che essi sono posti troppo vicini al robot rispetto alla loro reale collocazione, le misure successive in cui il robot si avvicinerà ad essi andranno a correggere facilmente questo errore. Dunque a questa scelta "conservativa", in cui l'errore che si può commettere è solo quello di vedere ostacoli dove non ci sono, si fa seguire questo tipo di aggiornamento, per cui una volta che una misura rivela una cella libera essa è da ritenersi libera a tutti gli effetti, anche in seguito a misure successive che invece la ritengano occupata.

V. Esplorazione

A. Stato dell'arte

Con esplorazione si intende il problema di pianificare la navigazione dei robot al fine di acquisire più informazioni possibili dall'ambiente. Durante l'esplorazione di un ambiente ignoto l'obiettivo è cercare di ottimizzare le risorse necessarie per la sua completa esplorazione e per la creazione della mappa: tempo impiegato, spazio percorso e, nel caso di sistemi multiagente, numero di robot necessari. Gli approcci classici ([3], [4]) si basano sul concetto di Frontier-Based Exploration: guidare il robot verso le regioni di frontiera (*frontier cells*) tra l'area esplorata e quella ancora da esplorare. Ad esempio in [5] viene definita una funzione di:

- costo per raggiungere una *frontier cell*
- beneficio ottenuto dalla *frontier cell*

L'algoritmo iterativamente aggiorna la mappa del territorio e sceglie *target points* per i singoli robot in modo da raggiungere un compromesso tra costi e benefici: i costi sono legati alla possibilità di passare per celle con una probabilità di occupazione diversa da zero; i benefici sono determinati da quanta informazione si può ottenere dal raggiungimento di una particolare *frontier cell*. Un altro approccio interessante è quello adottato in [6]. In questo articolo viene implementato un algoritmo randomizzato chiamato *SRT(Sensor-based Random Tree)*; qui si assume che il robot utilizzi sensori di prossimità a 360° con un raggio R di rilevamento elevato. L'algoritmo di Stentz in [7] vuole trovare ad ogni iterazione il cammino minimo (migliore) data la mappa del territorio. Si è provato a

[4]Ad ogni pixel quadrato del piano immagine corrisponde un quadrilatero sul piano reale, come si vede in figura 5, nella costruzione della mappa relativa ad una certa misura si posiziona l'ostacolo sul lato del quadrilatero più vicino al robot

implementare quest'ultimo, ma non si è potuti proseguire, data l'onerosità dell'algoritmo. Inoltre questo faceva nascere parecchi problemi nel caso si utilizzassero diversi robot esploratori. Quest'ultima soluzione è stata scelta solo per la pianificazione della traiettoria dell'*Actor*.

B. Algoritmo di esplorazione

Rispetto alla trattazione classica l'esplorazione è mirata ad individuare un percorso libero, affinchè l'*e-puck* cieco (*Actor*) riesca a raggiungere un punto di destinazione d. Grande difficoltà è stata quella di gestire una visuale molto stretta dei robot e non particolarmente lunga (angolo di visibilità di $28°$ e raggio di circa 40 cm). Per ovviare a questo problema è stato deciso di scattare sempre tre foto prima di richiamare l'algoritmo di esplorazione. In tal modo si può lavorare con una visuale minima di $60°$. L'algoritmo di esplorazione ha come ingresso la mappa del territorio (le celle contengono il valore numerico 1, 0 o ∞ a seconda che la cella corrispondente sia rispettivamente occupata, libera o sconosciuta), la posizione del robot e il punto d di destinazione dell'*Actor*. Ha come uscita il punto di esplorazione successivo e l'angolo verso cui guardare. Per semplicità ora verrà descritto l'algoritmo di esplorazione implementato per un robot. L'estensione al caso multirobot avverrà successivamente. L'algoritmo di esplorazione è descritto in Algoritmo 2. Innanzi tutto vengono determinate le celle di frontiera; per ogni cella viene calcolato l'indice di costo J. L'indice tiene conto della distanza dalla destinazione e dalla posizione attuale. In specifico il costo J per ogni cella di frontiera f_i è:

$$J = J_1 + J_2 = c_1(\|f_i - d\| - \min_{f_j \in F} \|f_j - d\|) + c_2(\|f_i - p\| - d_{ott})$$

In parole più semplici J è la somma di un costo dovuto alla distanza dalla destinazione di f_i, J_1, e di un costo dovuto alla distanza di f_i dalla posizione attuale del robot, J_2. Il primo costo è nullo se f_i è il punto della frontiera più vicino alla destinazione. Tutti gli altri punti hanno un costo positivo come si può notare dal grafico in alto in figura 6.

Il secondo costo si annulla se la distanza tra f_i e p è uguale a d_{ott}: si vuole che il robot si muova a una certa distanza dalla posizione corrente senza allontanarsi troppo in quanto si desidera che tenga una sua traiettoria e una sua contiguità nel movimento. Simulazioni e prove sperimentali hanno fatto fissare $d_{ott} = 25$ cm. La rappresentazione di J_2 è nel grafico in basso in figura 6. I pesi c_i sono stati variati in molte simulazione, ottenendo risultati significativamente diversi. Mediamente c_1 è stato posto di due ordini di grandezza piú grande rispetto a c_2, in quanto è importante che il robot si avvicini alla destinazione d . Una volta determinato il punto c che minimizza l'indice J vengono valutati diversi casi:

- se la distanza tra c e p è maggiore di d_{min} allora il robot si muove verso c. d_{min} è la distanza minima di moto ed è stata posta uguale a 15 cm. Ora il problema è quello di determinare l'angolo verso cui guardare. Si desidera che il robot guardi verso ambienti inesplorati, cercando inoltre di puntare possibilmente verso d. Per far ciò viene calcolata la distanza tra d e tutti i punti inesplorati appartenenti ad un intorno sferico di c di raggio $R = 15$ cm (ammesso che esistano) come è indicato nell'algoritmo; ϑ_{esp} viene calcolata in modo che il robot guardi verso il punto che minimizza questa distanza. Se non esistono punti inesplorati nell'intorno, allora ϑ_{esp} punta alla destinazione.
- se la distanza tra c e p è minore di d_{min} il robot preferisce girarsi e fare un'altra foto in una diversa direzione. $\alpha(k)$ è una funzione della iterazione *k*-esima in quanto è necessario che il robot riesca a esplorare tutta la zona circostante, girandosi con angoli opportuni a sinistra e a destra.
- nel caso in cui il robot dopo aver fatto un giro su se stesso e aver esplorato la zona circostante non trovi punti ottimi abbastanza distanti, si muove verso un punto sufficientemente distante da p che permetta al robot di trovare strade alternative.

Per chiarezza vengono fatti di seguito alcuni esempi. Si osservi la figura 7. Il robot deve riuscire ad aggirare l'ostacolo e raggiungere la destinazione. In figura 8 (in alto) è rappresentata la mappa in ingresso alla funzione di esplorazione in un certo istante, mentre il robot giallo rappresenterà la soluzione dell'algoritmo 2. La zona blu è la zona libera, quella verde è quella occupata da un ostacolo, la rossa è la zona inesplorata.

Input:
p = *posizione attuale del robot;*
ϑ = *direzione attuale del robot;*
M = *mappa del territorio;*
d = *destinazione del cieco;*

Output:
p_{esp} = *punto di esplorazione successivo;*
ϑ_{esp}= *angolo verso cui scattare la foto successiva;*

1) *Determinare l'insieme delle celle di frontiera* $F \in M$
2) ***Per ogni*** *cella calcolare l'indice di costo* J
3) *Determinare* $c = \operatorname{argmin}_{f_i \in F}(J)$
4) ***Se*** $\|c - p\| > d_{min}$
   - $p_{esp} = c$;
   - $B = \{(i,j) \in \mathcal{S}(p_{esp}, R) : M(i,j) = \infty\}$

     se $B \neq \emptyset$
     - $(m,n) = \operatorname{argmin}_{(i,j)\in B} \|(i,j) - (d(1), d(2))\|$
     - ϑ_{esp} = $\arctan \frac{n - p_{esp}(2)}{m - p_{esp}(1)}$

     altrimenti
     - ϑ_{esp} = $\arctan \frac{d(2) - p_{esp}(2)}{d(1) - p_{esp}(1)}$

   altrimenti se *la zona attorno non è esplorata completamente*
   - $p_{esp} = p$;
   - $\vartheta_{esp} = \vartheta + \alpha(k)$;

   altrimenti
   - A =$\{f_i \in F : \|f_i - p\| > d_{min}\}$
   - $p_{esp} = \operatorname{argmin}_{f_i \in A}(J)$
   - $B = \{(i,j) \in \mathcal{S}(p_{esp}, R) : M(i,j) = \infty\}$

     se $B \neq \emptyset$
     - $(m,n) = \operatorname{argmin}_{(i,j)\in B} \|(i,j) - (d(1), d(2))\|$
     - ϑ_{esp} = $\arctan \frac{n - p_{esp}(2)}{m - p_{esp}(1)}$

     altrimenti
     - ϑ_{esp} = $\arctan \frac{d(2) - p_{esp}(2)}{d(1) - p_{esp}(1)}$

Fine

Algoritmo 2: Algoritmo di esplorazione

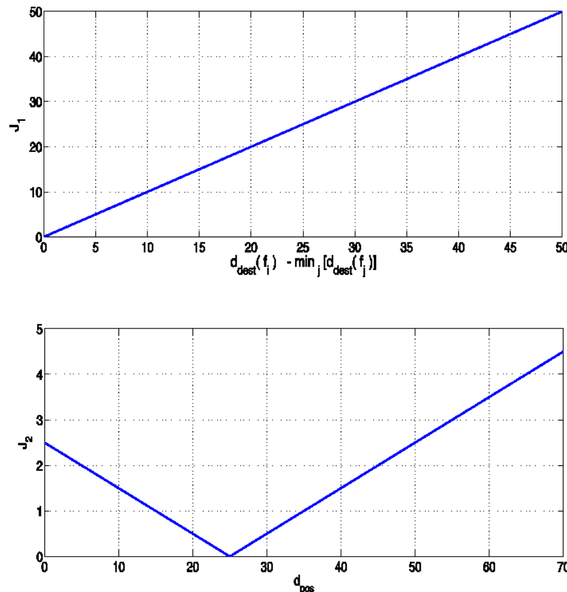


Figura 6: Indice J_1 con $c_1 = 1$ (sopra) e indice J_2 con $c_2 = 0.01$ (sotto).$d_{dest}(f_i)$ indica la distanza del punto f_i dalla destinazione; d_{pos} indica la distanza tra f_i e p

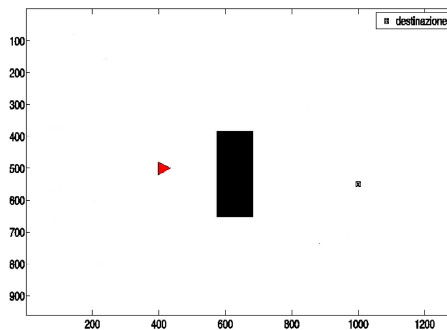
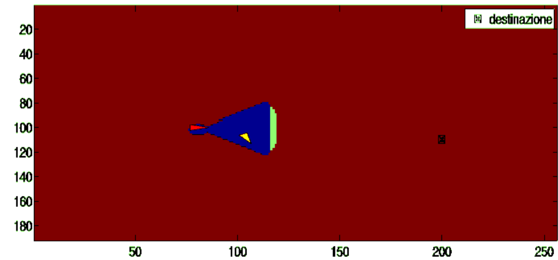


Figura 7: Mappa reale

La determinazione delle celle di frontiera avviene su una mappa “orlata”, cioè una mappa in cui l’ostacolo viene un po’ “ingrandito”. Questo tiene conto del diametro dell’*e-puck* in quanto in questo algoritmo si suppone di utilizzare robot puntiformi. Le celle di frontiera sono visibili in arancione in figura 8 (in basso) e sono state ottenute tramite un filtraggio dei dati dell’immagine. Con l’algoritmo proposto l’*e-puck* riesce ad uscire da situazioni difficili; il procedimento fornisce una soluzione anche nel caso il robot si trovi circondato da ostacoli. L’esempio successivo è significativo da questo punto di vista. Si osservi la figura 9; la soluzione che elabora l’algoritmo è la migliore possibile data l’informazione che si ha sulla mappa.

C. Estensione Multirobot

La situazione diventa più articolata e più complicata quando si introducono nella mappa diversi robot esploratori. Una tecnica interessante è descritta in [5]. Ad ogni iterazione vengono determinati N punti di frontiera che saranno i prossimi punti di osservazione. In seguito ciascun punto di frontiera viene



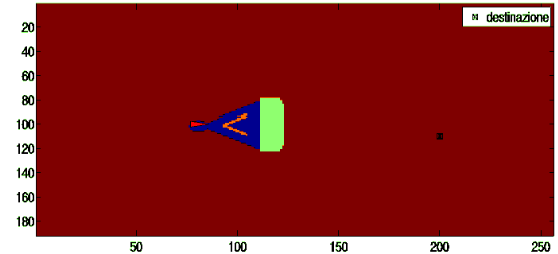


Figura 8: Mappa e soluzione dell’algoritmo

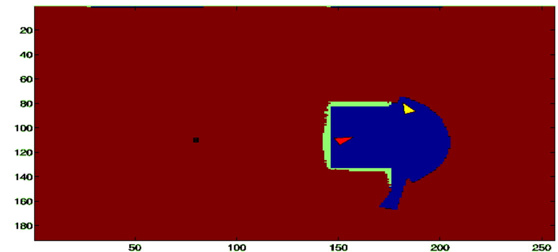


Figura 9: Robot circondato da ostacoli

assegnato ad uno degli N robot in base ad una funzione di costo: ogni robot andrà verso il punto di frontiera piu vicino.

L’algorimo implementato è simile all’Algoritmo 2; inoltre è stata tratta l’idea da [5] di dividere la zona di frontiera in N zone, ciascuna assegnata a un preciso robot, in base al criterio di minima distanza. Per semplicità viene descritto il metodo utilizzato con due robot esploratori. L’approccio è facilmente estendibile al caso di più esploratori. Ciascun robot si muove nel punto della sua frontiera minimizzante l’indice J. Ora l’indice tiene conto anche della distanza dei punti ammissibili dalla posizione di osservazione dell’altro robot: per ottenere ciò è stato introdotto un costo relativo alla vicinanza degli *e-puck*. Se i due robot sono più lontani di $d_o = 40$ cm questo costo si annulla; si vuole dunque che gli *e-puck* mantengano una certa distanza nell’esplorazione. A J viene aggiunto dunque il costo quadratico:

$$J_3 = c_3(\|f_i - p_o\| - d_o)^2 \quad \delta_{-1}(-\|f_i - p_o\| + d_o)$$

Dove p_o è la posizione dell’altro robot e δ_{-1} indica la funzione gradino. Sono stati valutati diversi valori del peso c_3. Il valore medio di simulazione è stato di un ordine di grandezza più piccolo di c_1. In figura 10 è rappresentato il grafico di J_3.

Si veda l’esempio con due robot esploratori in figura 11. In rosso è rappresentato il robot considerato. L’altro robot è quello azzurro. La zona di frontiera è rappresentata in

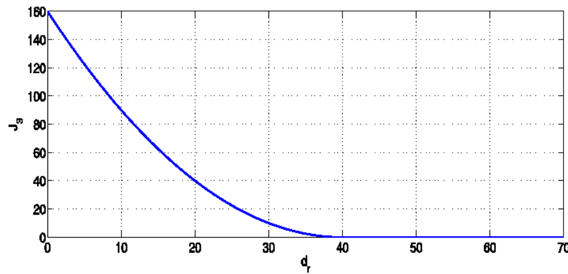


Figura 10: Indice J_3 con $c_3 = 0.1$. In ascissa c'è $d_r = \|f_i - p_o\|$

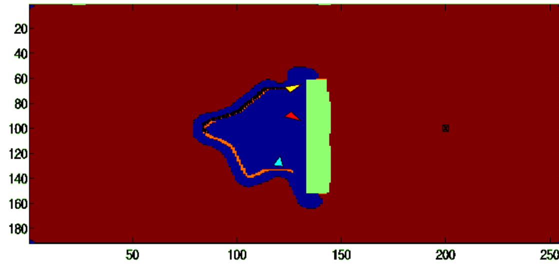


Figura 11: Due robot esploratori

arancione: quella tratteggiata viene assegnata al robot rosso; l'Algoritmo restituisce la posizione di osservazione successiva (robot giallo). Alla successiva chiamata della funzione di esplorazione sarà il robot azzurro che, in base alla posizione dell'altro robot (ora in giallo) deciderà dove andare e così via di seguito.

D. *Terminazione dell'algoritmo*

Per ragioni di ordine in algoritmo 2 si è omessa la parte di terminazione. In realtà quando viene avvistato l'obiettivo il file di esecuzione non richiama più l'algoritmo di esplorazione. L'Algoritmo 2 gestisce inoltre delle situazioni degeneri: l'obiettivo può essere irraggiungibile, per esempio a causa di passaggi troppo stretti. In tal caso l'algoritmo termina quando non ci sono più punti ammissibili, dopo l'esplorazione di tutto il territorio raggiungibile per i robot.

VI. L'ALGORITMO DI RICERCA DEI CAMMINI

La necessità di muoversi in un ambiente ostacolato comporta delle difficoltà nella fase di generazione delle traiettorie. L'argomento è stato approfonditamente trattato in letteratura. Esistono varie tecniche per fare sì che l'algoritmo di pianificazione sia in grado di evitare gli ostacoli. Un buon riassunto si può trovare in [8][5]. In genere, tali algoritmi si riducono alla ricerca di cammini minimi su un grafo, con algoritmi tipo *Dijkstra*, ben analizzati in [9], o alla minimizzazione di indici che penalizzano la distanza, l'energia spesa, il tempo di esposizione al pericolo e via dicendo [10], o ancora alla creazione di potenziali attrattivi verso l'obiettivo e repulsivi dagli ostacoli [11].

Tuttavia, tali tecniche suppongono perlopiù che l'ambiente ove si muove il robot sia noto a priori. É ovvio che tale supposizione non è valida ai nostri scopi, ove lo spazio operativo viene scoperto gradualmente dai robot esploratori. L'algoritmo che si è scelto per la ricerca di cammini minimi è dunque quello descritto in [7]:

A. *L'algoritmo D* (D-star)*

b) Un po' di storia e motivazioni della scelta: É un algoritmo di ricerca di cammini minimi su un grafo, tipo il famoso *Dijkstra* (risalente all'anno 1959, [12]). L'idea alla base di quest'ultimo algoritmo è quella di cercare un cammino su un grafo da un nodo di partenza a un nodo obbiettivo a partire dalla fine, e risalendo a ritroso scegliendo di volta in volta l'arco meno costoso fino a raggiungere il nodo di partenza. É facile intuire che questo procedimento è computazionalmente molto oneroso, basti pensare che in realtà si trova un percorso non solo dal nodo che interessa, ma da tutti i nodi che distano circa quanto esso dal target. Esiste un miglioramento in termini di efficienza di questo algoritmo, l'altrettanto noto *A** (A-star[6], [13]), che introduce una stima euristica della distanza tra la partenza e l'arrivo in modo da limitare il campo di ricerca del cammino su meno nodi. Nonostante l'euristica, tale algoritmo presenta delle bellissime proprietà di ottimalità e completezza che ora non indaghiamo, ma che si trovano tutte formalmente dimostrate, sempre in [13].

Tali proprietà vengono trasferite anche all'algoritmo in questione, suo fratello *D**, che in effetti prende il nome proprio da questo, ma sostituisce la *A* con una *D*, che sta per *Dynamic*. Infatti prevede la possibilità di aggiornare il cammino non appena si verifichino dei cambiamenti di costo negli archi, individuando una strada alternativa se questi divengono proibitivi, e migliorando la strada attuale se si apre un percorso di costo minore.

É questo l'algoritmo che cercavamo per il nostro *Actor*: dopo una fase di inizializzazione del cammino, la strada tra l'*Actor* e il *goal* viene mantenuta aggiornata man mano che gli esploratori avanzano nel loro lavoro di costruzione della mappa. Prima di passare ad una breve descrizione dell'algoritmo, è meglio introdurre il tema della costruzione del grafo a partire dalla mappa, giusto per avere in mente un esempio concreto di applicazione.

c) Costruzione del Grafo: Un grafo è costituito da un inseme di N nodi connessi tra di loro da archi pesati. Esistono svariate tecniche che permettono di costruire un grafo a partire da una mappa bidimensionale del territorio. Queste tecniche sono state ampio oggetto di studio[7].

Il criterio che è stato utilizzato per scegliere una di queste tecniche è quello della semplicità: data la mappa come griglia di occupazione, ogni nodo è collegato ai suoi otto vicini come in figura 12. Il costo dell'attraversamento di un arco libero si può scegliere pari a 1 nel caso di spostamenti laterali, alla $\sqrt{2}$ per uno spostamento in diagonale. Ma per semplicità di calcolo si sono scelti costi interi, in particolare un costo pari a 10 per vicini laterali, a 14 per quelli diagonali. Il costo di

[5]Consultabile anche in rete grazie a *Google Book*

[6]Non pervenuto il significato di tale lettera, ma la stellina sta a significare un qualche concetto di ottimalità

[7]anche nelle tesine del Corso di Progettazione dei Sistemi di Controllo degli anni precedenti, ad esempio quella di *Ricciato, Siego, Tamino, "Pianificazione del Moto e Controllo di un Uniciclo"*

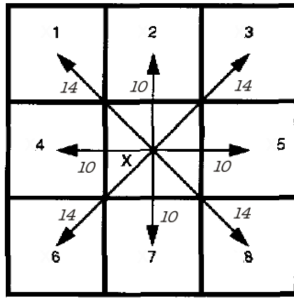


Figura 12: Dalla mappa al grafo

attraversamento per una cella occupata si può per il momento pensare infinito.

d) Descrizione dell'algoritmo: L'algoritmo utilizza puntatori che tengono traccia dei percorsi ottimi nel grafo. Si indichino con X, Y, due nodi generici del grafo, e con $b(X) = Y$ si intenda che X punta ad Y. Come *A**, *D** mantiene una *lista aperta* ($\mathcal{L}$) di nodi che devono essere elaborati. Per comodità, etichettiamo i nodi con $t(X) = NEW$ se il nodo X non è mai stato in $\mathcal{L}$, con $t(X) = OPEN$ se $X \in \mathcal{L}$, con $t(X) = CLOSED$ se X è uscito da $\mathcal{L}$. Ad ogni nodo si associano due funzioni di costo. Denotando con G il nodo obbiettivo (*goal*), la funzione $h(X)$ tiene traccia della somma dei costi di un cammino da X a G, mentre la funzione $k(X)$, detta funzione *chiave*, è definita come il minimo tra la $h(X)$ attuale e tutti i valori da essa precedentemente assunti da quando X è stato introdotto in $\mathcal{L}$. Più chiaramente, indicando con un pedice i una generica iterazione dell'algoritmo,

$$k(x) = \min_i h_i(X) \tag{7}$$

Si indichi poi con c_{XY} il costo di attraversamento dell'arco che congiunge X e Y. In Algoritmo 3 si descrive il procedimento principale, si veda anche [7].

L'algoritmo fa in modo che alla strada individuata dai puntatori corrisponda una $h(X)$ ottima, controllando passo per passo se i nodi vicini porgono un cammino migliore a partire dal *goal* e procedendo a ritroso verso il nodo di partenza, S, proprio come l'algoritmo di *Djkstra*. Ma permette un aggiornamento rapido nel caso cambino alcuni costi negli archi che compongono il grafo. L'aggiornamento avviene per mezzo di ripetuti inserimenti ed eliminazioni di nodi dalla *lista aperta*. L'idea principale dell'algoritmo, è quella di mantenere ottimi i nodi più vicini al *Goal*, e tramite questi propagare poi i cambiamenti di costo, in modo che il cammino si mantenga ottimo. I nodi presenti nella lista vengono dunque elaborati a partire da quello la cui funzione $k(X)$ è minore, cioè da quello più vicino al *goal*. Così operando, è chiaro che

$$k_{\min} = \min_{X \in \mathcal{L}} k(X) \tag{8}$$

rappresenta un valore importante, perchè esso è pari al costo minimo possibile, quello ottimo, esistente tra i nodi della *lista aperta*. La funzione $k(x)$ discrimina i nodi della lista in due tipologie:

1) nodi *RAISE*: se $k(X) < h(X)$
2) nodi *LOWER*: se $k(X) = h(X)$

Prima chiamata: si inserisca il nodo G in $\mathcal{L}$, e si proceda con i passi 1-11 finchè $t(S) = CLOSED$.
Chiamate successive: si inseriscano i nodi il cui arco è cambiato di costo in $\mathcal{L}$
Finchè $k_{\min} < h(S)$

1. $X = \arg\min_{Z \in \mathcal{L}}$, $k_{\min} = \min_{X \in \mathcal{L}} k(X)$
2. Se il passo 1. non porge X, **FINE**: non esiste un percorso
3. Eliminare X da $\mathcal{L}$; $t(X) = CLOSED$
4. **Se** $k_{\min} < h(x)$, cioè X è un nodo *RAISE*
   5. **Per ogni** Y vicino di X:
      Se $h(Y) \leq k_{\min}$ e $h(X) > h(Y) + c_{XY}$,
      allora $b(X) = Y$; $h(X) = h(Y) + c_{XY}$
   6. **Per ogni** Y vicino di X:
      7. **Se** $t(Y) = NEW$ **o**
         $\{b(Y) = X$ e $h(Y) \neq h(X) + c_{YX}\}$
         allora $b(Y) = X$; $Inserire(Y, h(X) + c_{YX})$
      8. **Altrimenti**
         9. **Se** $b(Y) \neq X$ **e** $h(Y) > h(X) + c_{YX}$
            allora $Inserire(X, h(X))$
         10. **Altrimenti se** $b(Y) \neq X$ **e** $h(X) > h(Y)+c_{XY}$
             e $t(Y) = CLOSED$ **e** $h(Y) > k_{\min}$
             allora $Inserire(Y, h(Y))$
11. **Altrimenti** ($k_{\min} = h(x)$, cioè X è un nodo *LOWER*)
    Per ogni Y vicino di X:
    se $t(Y) = NEW$ **o** $\{b(Y) = X$ e $h(Y) \neq h(X)+c_{YX}\}$
    o $\{b(y) \neq X$ **e** $h(Y) > h(X) + c_{YX}\}$
    allora $b(Y) = X$;$Inserire(Y, h(X) + c_{YX})$

Funzione $Inserire(X, h_{new})$

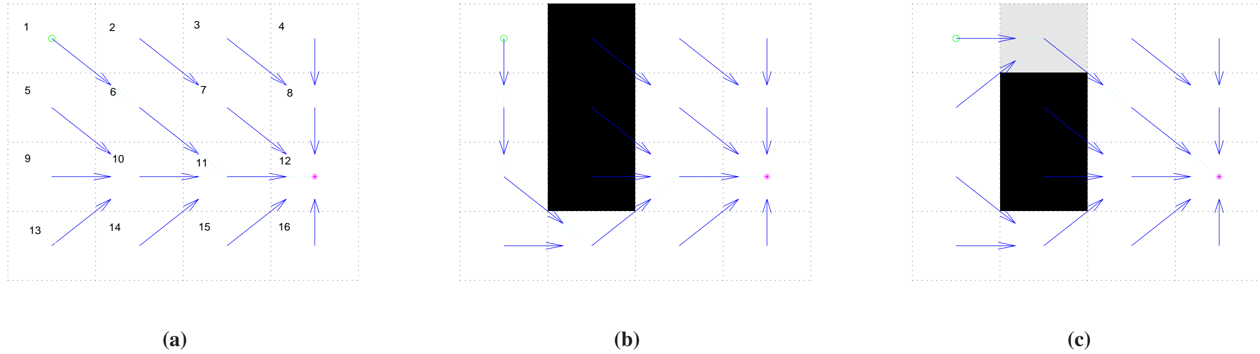
1. **Se** $t(x) = NEW$, **allora** $k(X) = h_{new}$
2. **Se** $t(x) = OPEN$, **allora** $k(X) = \min(k(X), h_{new})$
3. **Se** $t(x) = CLOSED$ **allora** $k(X) = \min(h(X), h_{new})$
4. $h(X) = h_{new}$; $t(X) = CLOSED$

Algoritmo 3: Algoritmo di ricerca dei cammini minimi *D**

Data la definizione (7), è chiaro che per i nodi *RAISE* non può essere ancora presente un cammino ottimo, mentre i nodi *LOWER* sono ottimi, perchè la loro funzione attuale di costo $h(X)$, nel momento in cui vengono elaborati, cioè quando la loro funzione $k(X)$ è la minima presente nella lista, è pari a $k_{\min}$. Questo significa che seguendo i puntatori a partire da un nodo *LOWER* fino a G, il cammino risultante è minimo. *D** usa i nodi *RAISE* per propagare i rialzamenti di costo del cammino dovuti alla scoperta di un arco di costo crescente, i nodi *LOWER* per propagare le riduzioni.

Seguendo l'esempio di figura 13, immaginiamo di partire da una situazione di piano libero come nel caso (a), e che a un certo punto venga individuato un ostacolo (che occupa i nodi 2, 6 e 10 in figura). Questi nodi, il cui cammino era ottimo, dunque sono *LOWER*, vengono inseriti nella *lista aperta*, e la loro elaborazione causerà l'inserimento di tutti i nodi che li puntavano (passo 11. dell'algoritmo, nodi 1, 5, 9, 13 in figura). Questi sono ora nodi *RAISE*. Prima di tutto, si cerca tra tutti i vicini uno che porga un cammino di costo minore (passo 5.). Poi, ad ogni vicino si esegue una delle seguenti operazioni:

- si propaga il cambiamento di costo se il vicino è



(a) (b) (c)

Figura 13: Esempio di cammini generati da $D*$

successore (passo 7.)

- si controlla se il vicino può essere ulteriormente migliorato. In questo caso è il nodo stesso che va inserito in $\mathcal{L}$ (passo 9.)
- si controlla se il vicino può migliorare la strada (passo 10.).

Il risultato di queste operazioni, attuate sui nodi in questione, porta alla situazione (b) della figura, in cui è stato individuato il nuovo percorso. Se poi uno dei nodi ostruiti torna nuovamente libero (diviene *LOWER*), ad esempio il numero 2, un solo passo dell'algoritmo (il 5.) porta alla soluzione di figura (c).

VII. Controllo della traiettoria e rilocalizzazione

A. Controllo della traiettoria

Si propone ora una soluzione semplice al problema del *trajectory-tracking*, cioè di far seguire al robot una traiettoria di riferimento. La letteratura propone diversi metodi, ma quello che va per la maggiore è il *Dynamic-Feedback Linearization*. Una trattazione approfondita del metodo, completa anche di prove sperimentali, si trova in [14]. Purtroppo tale tecnica non risulta soddisfacente per i nostri scopi. É impensabile infatti, o perlomeno alquanto insoddisfacente, controllare il robot soltanto attraverso una retroazione, in quanto si deve in ogni caso aspettare di avere un errore in posizione. Un'accurata azione di *Feedforward* è necessaria per aumentare le prestazioni del sistema. Per calcolarla, però, la tecnica menzionata pretende traiettorie sufficientemente *"smooth"*, cioè perlomeno derivabili due volte, e calcola l'azione in base ai valori di riferimento dell'accelerazione. Il nostro algoritmo di pianificazione, invece, porge traiettorie ovviamente continue, ma formate da un susseguirsi di spezzate, la cui derivata quindi è costante a tratti, e l'accelerazione risulta nulla quasi ovunque. Ci si è avvalsi allora della seguente tecnica, che risolve questo problema in maniera assai semplice ma brillante, e soprattutto perfettamente funzionante[8].

La tecnica è basata su una rimodellizzazione del sistema. Il modello del robot a due ruote è equivalente a quello dell'uniciclo, nel senso che esiste una relazione biunivoca tra i due segnali di ingresso ω_l e ω_r delle due ruote del robot con la velocità tangenziale v e quella angolare ω dell'uniciclo. Si considera allora la modellizzazione dell'uniciclo:

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{9}$$

dove con x e y si intendono le coordinate del centro del robot. In effetti, per questo sistema non è possibile calcolare v e ω tali che il percorso tracciato dal centro del robot sia non *smooth*, perchè il sistema è anolonomo, cioè non può mai avere una velocità laterale rispetto all'orientamento del veicolo[9]. Ma se si definisce un punto B fuori dall'asse delle ruote, più avanti del centro del veicolo di una auqntità b, di coordinate

$$x_B = x + b\cos(\theta), \qquad y_B = y + b\sin(\theta) \tag{10}$$

è possibile controllare il moto del robot in modo da eseguire con tale punto anche cammini con discontinuità avendo velocità sempre diversa da zero. Con tale cambio di coordinate, il sistema (9) diventa:

$$\begin{aligned} \dot{x}_B &= v\cos(\theta) - \omega\, b\sin(\theta) \stackrel{d}{=} v_{dx} \\ \dot{y}_B &= v\sin(\theta) + \omega\, b\cos(\theta) \stackrel{d}{=} v_{dy} \\ \dot{\theta} &= \omega \end{aligned} \tag{11}$$

dove con v_{dx} e v_{dy} si indicano le velocità desiderate, e la d sopra l'uguale questo concetto di desiderio. Nelle prime due equazioni la dipendenza dagli ingressi è invertibile, e si può definire la legge di controllo statica in funzione dei due nuovi ingressi $[v_{dx}\ v_{dy}]^\top$:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -b\sin(\theta) \\ \sin(\theta) & b\cos(\theta) \end{bmatrix}^{-1} \begin{bmatrix} v_{dx} \\ v_{dy} \end{bmatrix} = \begin{bmatrix} v_{dx}\cos(\theta) + v_{dy}\sin(\theta) \\ \frac{1}{b}(v_y\cos(\theta) - v_{dx}\sin(\theta)) \end{bmatrix} \tag{12}$$

dove la matrice inversa esiste perchè il suo determinante vale $b > 0$. In questo modo, si è ottenuto un sistema lineare e disaccopiato con ingressi v_{dx} e v_{dy}.

In sostanza, il riferimento si prende non più per il centro del robot, ma per il punto B, che si è scelto più avanti di 5 cm, si calcolano le velocità desiderate, *anche non continue*, derivando il riferimento, e si calcolano gli ingressi v e ω

[8]tratta da un'idea del Prof. A. De Luca, *Università di Roma Sapienza, [15]*

[9]si veda ancora [14] per maggiori dettagli

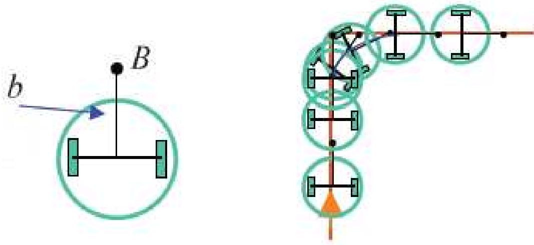


Figura 14: Il punto più avanti compie anche angoli retti

corrispondenti tramite la (12). Naturalmente, è opportuno introdurre un termine di correzione in retroazione. A tal fine è sufficiente progettare un banale regolatore, per esempio PID, per il sistema (9). Si è visto che è sufficiente un banale controllore proporzionale all'errore di posizione, ossia sostituire gli ingressi desiderati con

$$\begin{bmatrix} v_{dx} \\ v_{dy} \end{bmatrix} \rightarrow \begin{bmatrix} v_{dx} + k_x(r_x(t) - x_B) \\ v_{dy} + k_y(r_y(t) - y_B) \end{bmatrix} \tag{13}$$

La figura 14 illustra quanto detto.

B. Rotazione del robot

Per girare i robot su sè stessi al di fuori del moto non si è attuato un vero e proprio controllo, ma si è calcoato semplicemente un comando in *feedforward*. Dato che $\dot{\theta} = \omega$, è facile calcolare l'ingresso che serve. Infatti, detto T_g il tempo in cui si vuole che il robot giri, si ha (esattamente)

$$\theta(k+1) = \theta(k) + \omega T_g, \tag{14}$$

da cui si calcola la ω necessaria per compiere una rotazione di $\Delta\theta = \theta(k+1) - \theta(k)$. Verrebbe naturale prendere T_g pari al tempo di campionamento, ma siccome non si è precisi nel momento in cui il comando viene effettivamente dato al robot all'interno di un passo, si è scelto di creare un piccolo sotto-ciclo di periodo T_g che si usa per la fase di giramento. Cioè, si dà al robot il comando ω, si aspetta senza fare niente per T_g secondi, e poi si ferma il robot. Il tempo T_g è stato scelto pari a 0.35 secondi perchè così in un tempo massimo di $2T_g$ si compie la rotazione massima richiesta, ovvero π radianti.

C. Rilocalizzazione

Come detto, durante il movimento la posizione dei robot viene fornita dagli encoder posti sui due motori degli *e-puck*. L'errore di una singola misura è molto piccolo ed è dovuto essenzialmente a piccoli slittamenti delle ruote, molto sensibili alle minime imperfezioni del pavimento della pedana. Durante il moto si effettuano più misure successive e, poichè l'errore ad ogni istante non è scorrelato da quello all'istante precedente, per tempi lunghi si ottengono errori di posizionamento non trascurabili. Per rimediare a questo si è scelto di utilizzare una telecamera fissata sopra la pedana, a simulare un GPS virtuale. Poichè l'acquisizione e l'elaborazione dell'immagine richiedono un tempo significativo rispetto al tempo di campionamento, ed inoltre si ha un rumore di misura molto maggiore che nel caso degli encoder, sebbene scorrelato ad ogni istante, si è ritenuto di adottare questa tecnica di localizzazione solo quando è necessario conoscere più precisamente possibile la posizione del robot. Infine per poter ricostruire la mappa è necessario conoscere l'orientazione del robot, informazione che non è possibile ricavare con molta precisione da una singola immagine della telecamera[10] GPS. Come illustreremo invece questa informazione può essere ricavata dall'analisi del moto del robot su un certo intervallo di tempo, ed anche questa operazione viene effettuata in questa fase di rilocalizzazione.

Siano $q_{i,old}(t) = (x_{i,old}(t), y_{i,old}(t), \theta_{i,old}(t))$ le coordinate, espresse in pixel, del robot i-esimo al tempo t, I_0 l'immagine iniziale della pedana priva di robot, I_t, l'immagine della pedana al tempo t, si vuole definire un operatore $\Lambda(q_{i,old}(t), I_t)$:

$$\Lambda : (q_{i,old}(t), I_t \mapsto q_{i,new}(t) \tag{15}$$

dove $q_{i,new}(t)$ sono le coordinate *corrette* del robot i-esimo. L'algoritmo che si è adottato si basa fondamentalmente su un algoritmo di *blob-detection*, tramite filtraggio lineare. Disponendo già dell'effettivo background I_0, si ottiene un'immagine ΔI_t in cui sono presenti i soli robot: $\Delta I_t = I_0 - I_t$, dove l'ordine è dovuto al fatto che lo sfondo è più chiaro dei robot. Essendo la posizione del robot circa nota si è considerata solo una regione parziale di ΔI_t, all'interno della quale cercare il robot:

$$\overline{\Delta I_t}(h,k) = \Delta I_t(h,k) \quad \begin{aligned} h &= x_{i,old}(t) - X \dots x_{i,old}(t) + X, \\ k &= y_{i,old}(t) - Y \dots y_{i,old}(t) + Y \end{aligned} \tag{16}$$

con X ed Y sufficienti da garantire che il robot i-esimo sia contenuto in $\overline{\Delta I_t}$. Poichè gli oggetti da identificare sono solo gli *e-puck*si è costruito potuto costruire un modello K adeguato da utilizzare per il filtraggio. Esso costituisce l'oggetto che si andrà ad individuare nell'immagine $\overline{\Delta I_t}$ tramite correlazione:

$$F_t(u,v) = \sum_{u,v} \overline{\Delta I_t(h,k)} K(h-u, k-v) \tag{17}$$

Infine si vanno a cercare il massimi di $F_t(u,v)$:

$$(u_M, v_M) = \arg\max_{(u,v)} F_t(u,v). \tag{18}$$

A questo punto i $(x_{i,new}(t), y_{i,new}(t))$ si ricavano facilmente dagli (u_M, v_M) ponendo:

$$\begin{aligned} x_{i,new}(t) &= u_M + x_{i,old}(t) - X; \\ y_{i,new}(t) &= v_M + y_{i,old}(t) - Y. \end{aligned} \tag{19}$$

Un caso particolare è quello in cui l'immagine $\overline{\Delta I_t}(h,k)$ contiene più robot e quindi la $F_t(u,v)$ presenta più massimi. Per risolvere questo caso si va ogni volta ad azzerare la zona di $F_t(u,v)$ in cui è stato trovato l'*e-puck*ed a verificare se vi siano altri massimi dello stesso ordine di grandezza. Alla fine si sceglie il massimo più vicino alla posizione originaria.

[10]Si è stimata un'approssimazione di circa 6° gradi, almeno con i metodi di elaborazione dell'immagine da noi adottati

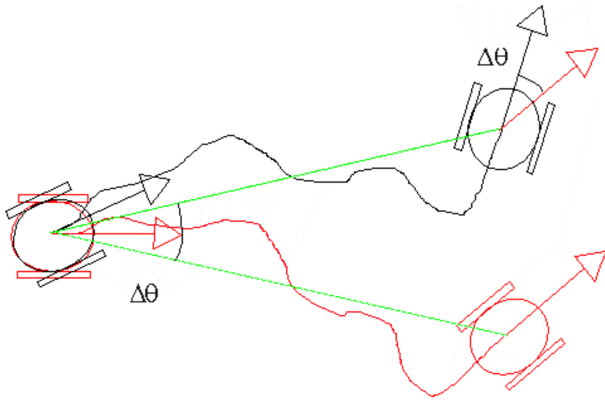


Figura 15: Prima del moto si conosce la posizione del robot, grazie al GPS virtuale, mentre è possibile che vi sia un errore sulla misura della posizione del robot: in *nero* è indicato l'*e-puck*, mentre in *rosso* è segnata la sua posizione misurata. Dopo il movimento l'errore sull'orientazione iniziale causa anche un errore sulla posizione del robot (sempre in *rosso*), che viene corretto tramite una nuova rilocalizzazione (in *nero*). L'angolo tra le due posizioni finali, segnato in *verde* coincide con l'errore sulla misura dell'orientazione del robot, che pertanto potrà essere corretta.

Nel nostro caso il blob K stato ottenuto come: $K = \frac{\bar{K}}{\max K}$ dove $\bar{K} = \sum_{i=1} N k_i$. I k_i rappresentano singole immagini quadrate di dimensione adeguate, che contengono l'*e-puck* in posizione centrale. Essi sono stati ricavati manualmente effettuando più foto del robot in diverse posizioni della pedana e con diverse orientazioni, tramite il GPS virtuale.

Effettuando due rilocalizzazioni successive, intervallate da una fase di spostamento del robot è possibile anche correggerne l'orientazione con un semplice procedimento, illustrato in figura 15. Sia $q_i(t_0) = (x_i(t_0), y_i(t_0), \theta_i(t_0))$ la posizione del robot i prima del moto. Tramite una prima rilocalizzazione si ottiene una misura precisa della sua posizione, mentre l'orientazione non può essere corretta: $\hat{q}_i(t_0) = (\hat{x}_i(t_0), \hat{y}_i(t_0), \hat{\theta}_i(t_0)) \approx (x_i(t_0), y_i(t_0), \hat{\theta}_i(t_0))$. Dopo il moto, al tempo t_1, l'errore sull'orientazione comporta un errore anche sulla posizione per cui si ha il robot in posizione $q_i(t_1) = (x_i(t_1), y_i(t_1), \theta_i(t_1))$, mentre la sua misura data dall'odometria del robot risulta $\tilde{q}_i(t_1) = (\hat{x}_i(t_1), \hat{y}_i(t_1), \hat{\theta}_i(t_1))$. Si effettua nuovamente la rilocalizzazione del robot ottenendo una misura, supponiamo esatta, della nuova posizione del robot:$\hat{q}_i(t_1) = (x_i(t_1), y_i(t_1), \hat{\theta}_i(t_1))$. Lo sfasamento $\theta_i - \hat{\theta}_i = \Delta\theta = \Delta\theta(t_0) = \Delta\theta(t_1)$ tra l'orientazione del robot e la sua misura si può perciò ottenere come:

$$\Delta\theta = \arctan\left(\frac{y_i(t_1) - y_i(t_0)}{x_i(t_1) - x_i(t_0)}\right) - \arctan\left(\frac{\hat{y}_i(t_1) - y_i(t_0)}{\hat{x}_i(t_1) - x_i(t_0)}\right).$$

Le considerazioni sopra ipotizzano che l'errore di orientazione del robot sia esclusivamente dato una misura iniziale non corretta e che poi si mantenga costante durante il moto. Questo evidentemente non accade nella realtà, dove l'errore sull'orientazione si genera durante il moto del robot. Si può facilmente verificare che in tal caso la tecnica adottata non consente di annullare, nemmeno teoricamente, l'errore sull'orientazione ma contribuisce comunque ad una correzione almeno parziale.

VIII. Risultati e conclusioni

Per testare gli algoritmi e lo schema di controllo generale si è sviluppato parallelamente alle esperienze pratiche un fedele ambiente simulativo in MATLAB. In particolare si è raffinato e calibrato l'algoritmo di esplorazione, scegliendo opportunamente i pesi dell'indice definito in Algoritmo 2.

I test in laboratorio si sono rivelati ovviamente più critici, nel senso che i problemi del mondo reale sono molti e spesso imprevedibili.

Il primo di questi riguarda la connessione *bluetooth* tra il computer centrale e i vari robot *e-puck*. Essa si è rivelata alquanto lenta ed instabile. Si pensi che il tempo medio per connettere un robot si aggira intorno ai due minuti. Purtroppo, una soluzione a questo problema non esiste, e l'unica cosa da fare è armarsi di pazienza e dedicarci molto tempo.

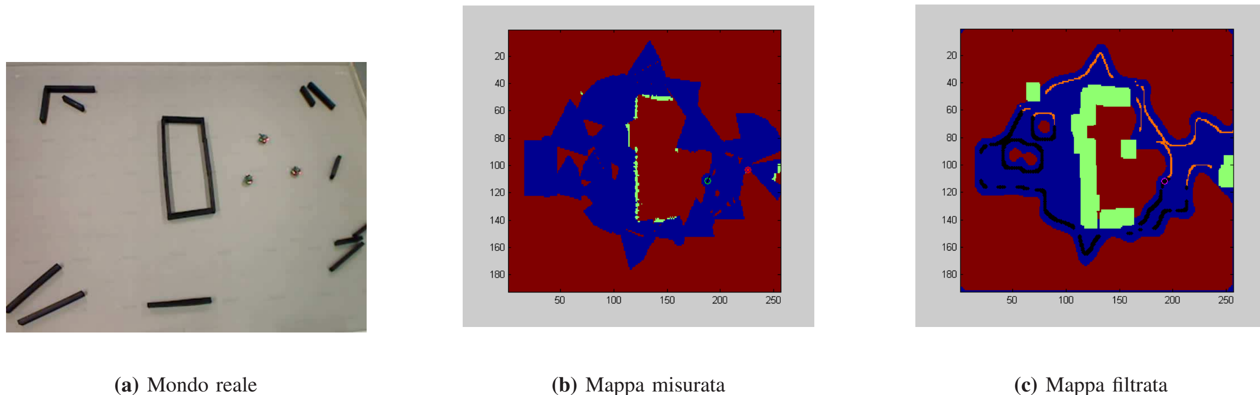
Altro problema che non era stato previsto in fase di progettazione è la calibrazione delle telecamere dei robot. I produttori svizzeri degli *e-puck*, infatti, non hanno pensato di fissare in alcun modo la telecamera al telaio, attaccata solamente tramite il cavo di trasmissione dei dati. Si è dovuto costruire un supporto artigianale, composto da frammenti di plastica uniti da scotch di carta. Ovviamente, non si è riuscito a creare un sostegno standardizzato, dunque si è dovuta compiere una calibrazione[11] della telecamera su misura per ogni robot, con conseguenza che il cono visivo non è uguale per tutti.

Un altro problema legato all'instabilità della connessione *bluetooth* è la difficolta di definire il tempo richiesto dalla comunicazione, fondamentale per la precisione nell'operazione di rotazione del robot. In effetti nella soluzione da noi proposta questa avviene in catena aperta, e la precisione è dovuta esclusivamente ad una corretta temporizzazione, resa impossibile dal protocollo *bluetooth*. Questo non comporta un errore sulla misura dell'orientazione del robot ma fa si che essa non sempre corrisponda a quanto calcolato nella fase di *Exploration*. Se ad esempio il tempo di rotazione è eccessivo può capitare che si creino alcune zone inesplorate molto sottili tra due coni visivi, così che le celle esplorate ad esse vicine sono ritenute erroneamente *frontiers cells* portando a peggioramenti nell'efficienza dell'algoritmo di esplorazione e quindi dell'intero sistema. Per diminuire quest'effetto si è deciso di applicare filtraggio a nucleo costante[12] sulla mappa, applicando poi una soglia adeguata, così che poche celle inesplorate all'interno di una zona esplorata vengano in effetti considerate esplorate. Questa operazione computazionalmente onerosa per mappe di grandi dimensioni in realtà sfrutta il filtraggio compiuto per inspessire gli ostacoli, con un costo praticamente nullo.

Una volta risolte tutte le complicazioni del caso, i test sperimentali hanno mostrato il buon funzionamento del sistema progettato. In primo luogo si sono effettuate alcuni test di prova con solamente uno o due *explorer*, per verificare la costruzione della mappa, quindi si è introdotto l'*actor* per alcuni test definitivi. I risultati sono stati positivi, per diverse mappe, e sono riportati nella figura 16. In particolare la figura

[11] che consiste nel determinare i valori di h ed f

[12] Il nucleo è cioè costituito da una matrice quadrata $p \times p$ in cui ogni elemento ha valore $\frac{1}{p^2}$



(a) Mondo reale **(b)** Mappa misurata **(c)** Mappa filtrata

Figura 16: Risultati: ricostruzione ambiente

16c mostra la mappa utilizzata per la fase di *Exploration* nella quale sono state eliminate le *frontiers cells* isolate come spiegato sopra.

APPENDICE A
ASPETTI TECNICI ED IMPLEMENTATIVI

A. Impostazione dell'area di ripresa della telecamera

Il DSP di cui è provvisto l'*e-puck* non è in grado di memorizzare ed elaborare l'immagine che viene prodotta dalla telecamera VGA di cui è dotato, essa pertanto viene ridimensionata prima di passare al DSP. L'utente, tramite la libreria *ePicKernel*, può gestire parzialmente il ridimensionamento impostando i parametri di campionamento dell'immagine o le sue dimensioni, ottenendo però sempre una porzione centrale dell'immagine di partenza. Per la nostra applicazione risulta utile allo scopo solo la parte dell'immagine sotto l'orizzonte, visto che si vuole andare a costruire una mappa bidimensionale, ed anzi, risulta indispensabile ottenere un'immagine che comprenda il più possibile la parte prossima all'*e-puck*. Per fare ciò e quindi poter utilizzare effettivamente la telecamera è stato necessario andare a modificare il codice eseguibile caricato sul microprocessore di ogni *e-puck*. si è dovuto modificare il codice C da cui era stato assemblato il file presente sull'*e-puck* e quindi compilarlo e ricavarne un eseguibile tramite il software MPLAB ed il compilatore C30 della GNU.

B. Mapping

L'implementazione della trasformazione descritta in precedenza in MATLAB si è costituita fondamentalmente di due parti. La prima ha implementato in uno script le equazioni (4) e (5) in modo da ricavare per ogni pixel del piano immagine il corrispondente raggio $\sigma(x_p, y_p)$ e la fase $\xi(x_p, y_p)$. Questi valori sono raccolti in due matrici, che chiameremo `modulo`(x_p, y_p) e `fase`(x_p, y_p), di dimensione $X_p \times Y_p$, cioè 40×40.

La definizione di tali valori richiede ovviamente di conoscere i valori del fuoco f e dell'altezza h della telecamera. La stima di questi due parametri è risultata decisamente complicata e causa di notevole dispendio di tempo poichè le telecamere montate sugli *e-puck* sono mobili, e la struttura degli stessi rende complicato ogni tentativo di fissarle adeguatamente. Pertanto prima di ogni utilizzo delle stesse si rendeva necessaria una fase di calibrazione per stimare tali parametri, effettuando alcune misure di ostacoli a distanza nota, durante la quale si procedeva sia a ripristinare la telecamera in una posizione "standard", sia ad aggiornare i valori di questi parametri. Nel caso di utilizzo di più robot esploratori si sono adoperati parametri diversi per ognuno di essi. Valori medi riscontrati in seguito a più simulazioni sono $f = 50$ mm ed $h = 44$ espresso in pixel, cioè in unità di dimensione di un pixel del piano immagine. Realizzata questa parte si è implementata una funzione che riceve in ingresso la posizione e l'orienamento del robot ed il `bordo` e restituisce una matrice di dimensioni pari alla mappa globale, che rappresenta la `misura` fatta dal robot. Tale misura sarà costituita ovviamente da celle di valore 0 ad indicare le celle che il robot ha visto libere, di valore 1 ad indicare le celle che il robot ha visto occupate e di valore ∞ per le celle che l'area che non è stata interessata dalla misura del robot. La funzione crea una griglia quadrata di dimensioni ridotte pari al doppio della visuale massima del robot, in cui ogni cella corrisponde ad un centimetro quadrato. Si scorrono uno ad uno i valori del `bordo`(i), rendendo libero un settore circolare centrato nel centro del quadrato, di modulo `modulo`$(i$,`bordo`$(i))$ e per un intervallo fase centrato in `fase`$(i$,`bordo`$(i))+\delta$ e la cui ampiezza è anch'essa contenuta in una matrice analoga a queste due, facilmente ricavabile in base alle considerazioni sin qui esposte, che non si riporta perchè costituerebbe una mera ripetizione dei concetti già esposti[13]. Occorre ricordare che questo approccio comporta una piccola approssimazione nella rappresentazione del cono visivo del robot, come mostrato in figura 17. Così facendo infatti è come se si centrasse il cono sul centro del robot e non più sulla sua telecamera. Questa scelta, che come vedremo comporta un'approssimazione minima ci consente di semplificare la trasformazione da applicare, in quanto ci si riferisce automaticamente ai dati noti, cioè a posizione e l'orientazione del robot, ed in secondo

[13]In ogni caso nel codice MATLAB allegato alla tesi si potrà trovare anche questo calcolo

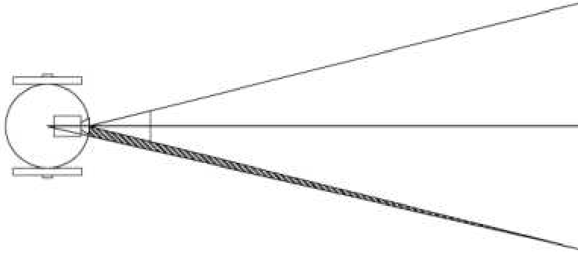


Figura 17: Approsimazione sul cono visivo

luogo consente di ritenere esplorato l'intera area circostante al robot tramite un numero sufficiente i rotazioni succesive nella stessa posizione. Come detto il valore di approssimazione è minimo poichè l'ampiezza del cono visivo è molto piccola, la distanza della camera dal centro del robot è inferiore al valore f del fuoco della stessa, ed inoltre l'area che si ritiene esplorata in più, viene spesso compresa nelle osservazioni successive. Infine poichè il robot effettua le sue misure solo da aree già esplorate in precedenza il fatto di ritenere esplorata la zona intorno alla camera non porta ad alcun rischio di mancata visione di ostacoli. Si sottolinea infine come questo non comporti approssimazioni ulteriori (oltre a quelle legate alla discretizzazione della mappa e del piano immagine) sul corretto posizionamento degli ostacoli.

In caso sia stato individuato un ostacolo esso viene segnato sulla mappa per uno spessore di circa 3 cm, mentre nel caso `bordo`(i)=0, cioè in cui la colonna i del piano immagine non visualizzi ostacoli, l'ampiezza libera sarà pari a `modulo`(i,40), e non sarà aggiunto alcun ostacolo. Essendo il tempo di calcolo un parametro critico del nostro lavoro, in realtà si è pensato di lavorare riportandosi al primo quadrante, e cioè la griglia quadrata su cui si lavora a dimensioni pari al massimo raggio visuale del robot. Dopodichè ci si riconduce alla corretta orientazione tramite alcune operazioni di traslzione o capovolgimento della matrice stessa. L'ultimo banale passaggio consiste nel collocare questa mappa *locale* così calcolata all'interno di una mappa di dimensioni pari alla mappa *globale*, secondo la posizione attuale dell'*e-puck* che ha effettuato la rilevazione.

Per quanto riguarda l'aggiornamento della mappa invece la funzione che implementa l'operatore Φ è banale: $M(k) = \min(M(k-1), m(k))$, cioè $M_{ij}(k) = \min(M_{ij}(k-1), m_{ij}(k))$.

C. Path-Planning

Una naturale implementazione dell'algoritmo *D** implicherebbe l'utilizzo di strutture dati tipo liste concatenate o altre strutture dati. Purtroppo, MATLAB offre soltanto i vettori. Si è dunque organizzato il tutto secondo questa struttura. Si sono definiti quattro vettori, t, b, h, k che rappresentano rispettivamente il *tag* (etichetta), il nodo puntato, la funzione $h(X)$ e la funzione $k(X)$, e i nodi sono rappresentati dagli indici di tali vettori. Con un'opportuna funzione si trasformano i due indici della mappa (le due cordinate), attraverso i quali verrebbe più naturale identificare i nodi, in un unico indice lineare, come si vede in figura 12. La *lista aperta* è anch'essa rappresentata da un vettore, anzi, da una matrice la cui prima colonna contiene i numeri che individuano i nodi ad essa appertenenti, la seconda il valore della rispettiva funzione k. Questo per facilitarne l'oridinamento. Con queste definizioni, è poi stato facile implementare l'algoritmo 3.

Nonostante la capacità di aggiornarsi iterativamente, l'algoritmo rimane di complessità abbastanza elevata, e naturalmente crescente con il numero dei nodi. Allora *D** non si è applicato alla mappa completa, ma ad una mappa sotto-campionata prendendo un pixel ogni due. In questo modo, si è ridotta la complessità di un fattore 4.

Inoltre, il cammino è trovato supponendo il robot puntiforme. Ma esso non è tale. Allora si è utilizzata una mappa modificata in cui le celle occupate vengono "orlate", cioè ampliate in ogni direzione della dimensione pari a (un po' più del) raggio del robot, in modo da essere certi che esso vi passi.

Ci sono poi alcune considerazioni da fare su come generare un effettivo riferimento in posizione per i robot. Dato il cammino, si devono definire gli istanti di tempo cui il robot deve attraversare i nodi. Per farlo, si è interpolata la traiettoria in modo che ad ogni istante di campionamento la strada che deve percorrere il robot sia sufficientemente corta da far sì che venga percorsa tutta.

D. Rilocalizzazione

In MATLAB l'algoritmo è stato implementato esattamente come indicato. Per la costruzione del blob K si sono utilizzate 15 immagini, 10 per costruire effettivamente l'immagine ed altre 5 per verificarne il funzionamento, ottenendo che il massimo trovato differiva dal centro del robot di al più 2-3 pixel, pari a circa 0.5 cm.

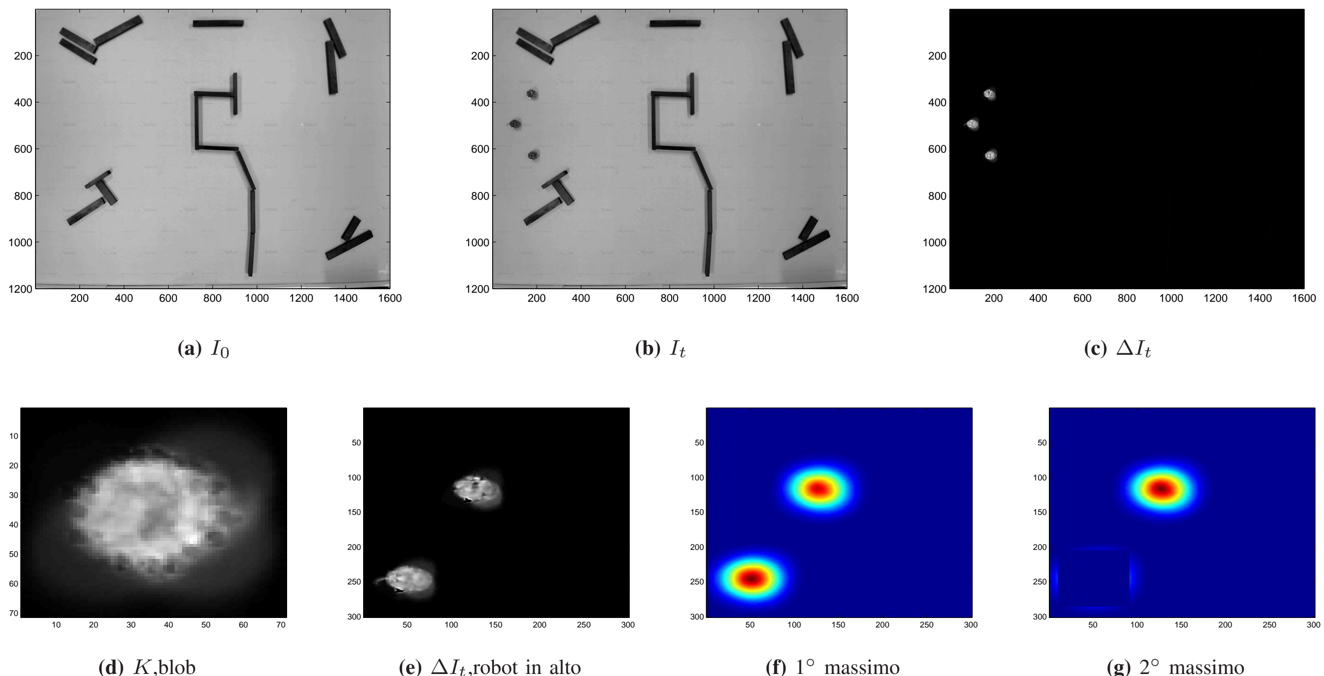
E. Tecniche per la simulazione del Mapping

Per la verifica del sistema creato in ambiente simulativo si sono utilizzate mappe virtuali realizzate tramite programmi di grafica. La simulazione del sensore è stata effettuata nella maniera più fedele possibile,imitando l'ampiezza e la lunghezza del cono visivo, come è documentato in figura 19.

APPENDICE B
APPARATO STRUMENTALE

a) Ambiente per la sperimentazione: Come area di lavoro si è utilizzata una pedana appoggiata sul pavimento, su cui si possono muovere gli *e-puck*. La pedana misura $3.20m \times 2.40m$. In un primo momento si è pensato di utilizzare il computer fisso presente in laboratorio. In seguito si è preferito utilizzare un computer portatile per maggiori capacità di calcolo. Il pc è provvisto di rete bluetooth utilizzata per la connessione con gli *e-puck*.

b) E-puck: Per sperimentare gli algoritmi ideati in questo progetto in un ambiente reale, si è utilizzato un veicolo mobile denominato *e-puck*. La meccanica di tale robot è molto semplice ed elegante: la struttura è basata su un corpo unico del diametro di 70 mm a cui sono fissati il supporto del motore, il circuito e la batteria. La batteria, allocata al di sotto del



(a) I_0 **(b)** I_t **(c)** ΔI_t

(d) K,blob **(e)** ΔI_t,robot in alto **(f)** 1° massimo **(g)** 2° massimo

Figura 18: Immagini utilizzate nella fase di rilocalizzazione dei robot. In particolare nella figura 18f si osservano i due massimi dovuti a due robot. Quello più in basso è leggermente maggiore e viene individuao per primo, poi viene cancelatoe si ottiene la figura 18g da cui si ricava il secondo massimo. Tra i due viene scelto quest'ultimo poichè più vicino alla posizione prevista per il robot, che corrisponde al centro delle figure 18e, 18f e 18g.

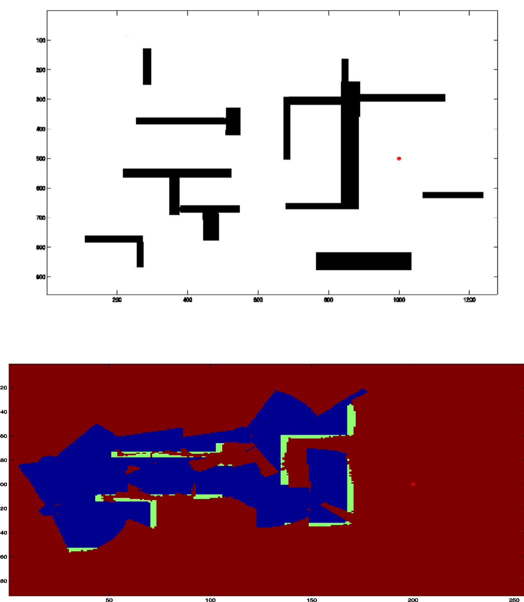


Figura 19: Costruzone della mappa in ambiente simulativo

motore, è collegata al circuito con due contatti verticali e può facilmente essere estratta per essere ricaricata.

Il motore a passo con riduttore di marcia realizza 20 passi per rivoluzione e la marcia possiede una riduzione di $50:1$. Le due ruote hanno un diametro di circa 41 mm, e possiedono un sottile pneumatico di color verde, sono distanti circa 53 mm e la massima velocità che possono raggiungere è di 100 passi al secondo, che corrisponde ad un giro completo effettuato in un secondo. La comunicazione avviene via bluetooth tramite un chip bluetooth , $LMX9820A$, potendo così comodamente accedere al robot come se fosse connesso ad una porta seriale. L'unica differenza consiste nel fatto che il $LMX9820A$ manda comandi particolari per la sua connessione e sconnessione (intervallo di comunicazione: $min: \ 15cm$, $max: \ 5m$). Il robot e-puck possiede inoltre tre microfoni (massima velocità di acquisizione: $33kHz$), un accelerometro 3D (strumento che misura la forte micro-accelerazione e micro-decelerazione che possiede il robot quando si muove), 8 sensori di prossimità IR (Infrared proximity sensors), una telecamera (risoluzione: $640(h) \times 480(l)$ pixel, a colori) e 8 led colorati (rossi) , tutti controllati utilizzando 24 segnali (Pulse Width Modulated) separati, generati da un microcontrollore posizionato internamente all' *e-puck*$(PIC18F6722)$. Ogni led può essere controllato singolarmente, per una maggiore flessibilità, oppure in maniera sincronizzata, semplicemente settando pochi parametri (come la massima luminosità,il periodo di intermittenza etc.).

c) Telecamera: Per visualizzare i robot e acquisire la loro posizione in tempo reale si è utilizzata una telecamera Logitech QuickCam (risoluzione 1200×1600) fissata al soffitto perpendicolarmente al piano di lavoro.

La telecamera utilizzata (*Logitech QuickCam*) è stata scelta poichè considerata molto efficiente: è dotata infatti di un trigger esterno asincrono, che permette di acquisire le immagini in modo istantaneo, senza ritardi significativi. Tale strumento inoltre dà la possibilità di scandire l'immagine dell'intera pedana di lavoro permettendo di sfruttare tutto lo spazio disponibile per le misurazioni. Per l'acquisizione e l'utilizzo dell'immagine nell'ambiente MATLAB si è utilizza-

Figura 20: *Epuck*

Figura 21: *Logitech QuickCam*

to il pacchetto "*imaqtool*", in modo da ottenere la matrice dell'immagine corrente tramite il comando "*getsnapshot*".

RIFERIMENTI BIBLIOGRAFICI

[1] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[2] Various, *Artificial Intelligence and Mobile Robots: Case Studies of Succesful Robot Systems*, R. M. David Kortenkamp, R. Peter Bonasso, Ed. AAAI Press, 1998.

[3] B. Yamauchi, A. C. Schultz, and W. Adams, "Mobile robot exploration and map-building with continuous localization," in *ICRA*, 1998, pp. 3715–3720.

[4] B. Yamauchi, "A frontier-based approach for autonomous exploration," Oct. 30 1997. [Online]. Available: http://citeseer.ist.psu.edu/178266.html;http://www.aic.nrl.navy.mil/~yamauchi/papers/compressed/cira97.ps.gz

[5] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, pp. 376–386, 2005.

[6] A. Franchi, L. Freda, G. Oriolo, and M. Vendittelli, "A randomized strategy for cooperative robot exploration," in *ICRA*. IEEE, 2007, pp. 768–774. [Online]. Available: http://dx.doi.org/10.1109/ROBOT.2007.363079

[7] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *In Proceedings of the IEEE International Conference on Robotics and Automation*, 1994, pp. 3310–3317.

[8] J. C. Latombe, *Robot Motion Planning*. Boston, MA: Kluver Academic Publishers, 1991.

[9] A. R. Soltani, H. Tawfik, J. Y. Goulermas, and T. Fernando, "Path planning in construction sites: performance evaluation of the dijkstra, A_*, and GA search algorithms," *Advanced Engineering Informatics*, vol. 16, no. 4, pp. 291–303, 2002. [Online]. Available: http://dx.doi.org/10.1016/S1474-0346(03)00018-1

[10] C. Lindsay, "Automatic planning of safe and efficient robot paths using octree representation of configuration space," in *IEEE Transactions on Robotics and Automation*, 1987.

[11] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 23–32, Feb. 1992.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. SSC-4(2), pp. 100–107, 1968.

[14] A. D. L. G. Oriolo and M. Vendittelli, "Wmr control via dynamic feedback linearization: Design, implementation, and experimental validation," *IEEE Transactions on Systems Technology*, vol. 10, no. 6, pp. 835–852, 2002.

[15] A. D. Luca, "Diapositive del corso di robotica 1, università di roma sapienza."