UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata

Advanced Robotics Laboratory

Corso di Robot industriali e di servizio / Robotica mobile

(Prof. Riccardo Cassinis)

Robot Speedy aspirapolvere

Elaborato di esame di:

Jacopo Beschi, Carlo Svanera, Francesco Falanga

Consegnato il: **29 agosto 2012**

Sommario

Il lavoro svolto ha avuto come scopo lo studio e lo sviluppo di un algoritmo che regoli il comportamento di un robot aspirapolvere, in particolare del robot Speedy presente nel Laboratorio di Robotica Avanzata della facoltà. Questo elaborato descrive l'implementazione software adottata per gestire la navigazione del robot nel contesto descritto. In particolare ci si è concentrati sull'implementazione di un algoritmo che faccia compiere al robot il percorrimento completo di una stanza avente al suo interno degli ostacoli.

1. Introduzione

Questo elaborato è focalizzato sul progetto di un algoritmo di movimento per il robot *Speedy* utilizzato nel laboratorio di robotica avanzata. Sono ora esposte le principali caratteristiche del robot per cui è stato pensato l'algoritmo, e una breve panoramica del pacchetto software ARIA.

Il robot *Speedy* ha un architettura differential drive, con due ruote motrici fisse ed indipendenti ed una ruota folle e pivottante. Sulla parte superiore del robot è appoggiato un calcolatore portatile sul quale è installato il pacchetto dei software utilizzati per programmare la movimentazione del robot. Questo modello dispone di sette sonar disposti sulla parte anteriore che permettono di rilevare la presenza di ostacoli, fino ad una distanza massima di circa 4 m, su un intervallo poco più ampio da -90° a 90°. La stessa struttura del robot costituisce un primo limite per la sua programmazione di robot aspirapolvere: trattandosi di un robot anolonomo è difficile pensare alla possibilità che percorra esattamente ogni punto della superficie della stanza. Questo è dovuto ai limiti di movimento intrinseci del robot, e pertanto il risultato finale che ci si aspetta è quello di un percorrimento "il più completo possibile". Inoltre, anche la disposizione dei sonar solamente nella parte anteriore costituisce un ulteriore limitazione che deve essere presa in considerazione.

Il programma è stato scritto in C++, utilizzando le librerie Aria. Il pacchetto ARIA costituisce un'interfaccia, orientata agli oggetti, per la programmazione di applicazioni per il controllo di robot appartenenti alla linea commerciale di ActivMedia Robotics, di cui *Speedy* fa parte. Scritto interamente in C++ e la tipologia di architettura scelta per interfacciare l'applicativo di movimentazione al robot fisico è di tipo client-server: il programma dell'utente costituisce la parte client mentre il robot costituisce la parte server.

Per muovere il robot, ARIA offre un meccanismo "per azioni". La classe ArAction mette a disposizione numerose utili azioni già pronte per essere utilizzate mentre con la classe ArActionDesired è possibile definire azioni con il comportamento preferito dall'utente. La parte principale nella definizione di un'azione è la funzione denominata 'fire', che contiene l'algoritmo del comportamento dell'azione. Per poter essere eseguite, le azioni devono essere inserite in una lista tramite l'istruzione addAction. Una volta inserita, un'azione può venire rimossa o disattivata. Ad ogni azione viene associato un valore, compreso tra 0 e 100, indicato come priorità. Questo parametro è usato dal Resolver per stabilire l'ordine di esecuzione delle azioni inserite nella lista. Viene eseguita per prima l'azione con priorità più elevata, seguita da tutte le altre fino a quella con la priorità più bassa.

2. Il problema affrontato

Il problema che è stato affrontato è la creazione di un algoritmo che permetta al robot *Speedy* di emulare il comportamento di un robot aspirapolvere. Tale algoritmo deve avere le seguenti caratteristiche fondamentali:

1. Capacità di percorrere tutta la superficie raggiungibile all'interno di una stanza
2. Capacità di rilevare eventuali ostacoli presenti e di gestire le procedure per evitarli, in modo da poter continuare il proprio cammino

Per soddisfare il primo punto è necessario che il robot mantenga in memoria in qualche modo lo "stato" della stanza, cioè i punti dell'ambiente già puliti e quelli ancora da visitare. Ciò ha la duplice funzione di permettere al robot di capire quando il lavoro è concluso (cioè quando tutti i punti della stanza sono stati visitati), e di ottimizzare in parte il tragitto da compiere. Infatti un metodo efficace, ma per nulla efficiente, per un robot aspirapolvere sarebbe quello di vagare casualmente all'interno della stanza, dato che prima o poi essa sarebbe completamente pulita. La presenza di una mappa, anche approssimata, aiuterebbe invece il robot (l'algoritmo) a decidere il percorso da compiere, ed eventualmente ad ottimizzarlo in modo che sia il più breve possibile.

A questo proposito è cruciale l'aspetto della localizzazione del robot nell'ambiente di lavoro. Esso dovrà conoscere la sua posizione attuale per poi decidere dove dirigersi. Per semplicità si è utilizzato un sistema di localizzazione relativa, per cui il robot conosce la sua posizione in coordinate relative, determinate rispetto alla sua posizione di partenza. Quest'ultima deve essere quindi specificata come precondizione per il corretto funzionamento del programma.

Per quanto riguarda il secondo punto, le procedure per evitare gli ostacoli dovranno tener conto che il robot ha solamente sensori frontali. Pertanto il movimento dovrà il più possibile essere "in avanti".

3. La soluzione adottata

L'idea dell'algoritmo è stata inizialmente orientata a delle procedure che facessero muovere il robot seguendo una traiettoria a spirale. Quest'idea non ha però avuto successo (come spiegato nel prossimo paragrafo) ed ha lasciato spazio a un nuovo progetto basato sulla tecnica "occupancy grid mapping", che ha permesso di risolvere i problemi esposti nel paragrafo precedente.

3.1. Idea 1: spirale

La prima soluzione che abbiamo preso in considerazione per ottenere la totale copertura della stanza da parte del robot si basa su una metodologia di percorrenza "a spirale". Inizialmente posizionato ai bordi della stanza, il robot avrebbe percorso tutta la stanza dalle pareti verso il centro, possibilmente senza passare più volte per uno stesso punto. L'algoritmo prevedeva le seguenti assunzioni, necessarie al suo funzionamento:

1. La stanza in cui avviene l'azione del robot è di forma rettangolare, ed è assimilabile a una griglia con celle quadrate di dimensione costante (la dimensione della stanza non è conosciuta a priori).
2. Gli ostacoli presenti nella griglia possono essere in qualunque posizione e sono di forma rettangolare (multiplo di una cella). Gli ostacoli non cambiano posizione nel tempo.
3. Le celle della griglia hanno dimensioni leggermente maggiori a quelle del robot, per consentirgli le operazioni di rotazione.
4. La posizione iniziale del robot è conosciuta a priori ed è in un angolo della stanza.

L'idea per l'algoritmo era la seguente. Supponendo una stanza priva di ostacoli, il robot si muove costeggiando la parete delle stanza finché non trova la parete successiva. Una volta raggiunta la parete, cambia direzione costeggiando il nuovo lato della stanza, e così via finché non torna al punto di partenza. A questo punto viene diminuita la misura del raggio della spirale (di una misura pari alla

dimensione della cella) e poi si ripete iterativamente la percorrenza della spirale. L'algoritmo termina quando il robot si posiziona al centro della stanza: il robot riconosce di essere giunto al centro quando percorre più volte la stessa traiettoria chiusa.

Purtroppo questo approccio portava troppi problemi implementativi: come rilevare che il robot era arrivato al bordo della stanza? E, nel caso di stanza con ostacoli: come discriminare tra la presenza di un ostacolo e la presenza di una parete?

Nelle figure seguenti viene mostrata una mappa di esempio (Figura 1) ed un immagine che rappresenta come il robot si dovrebbe comportare secondo il l'algoritmo a spirale(Figura 2).

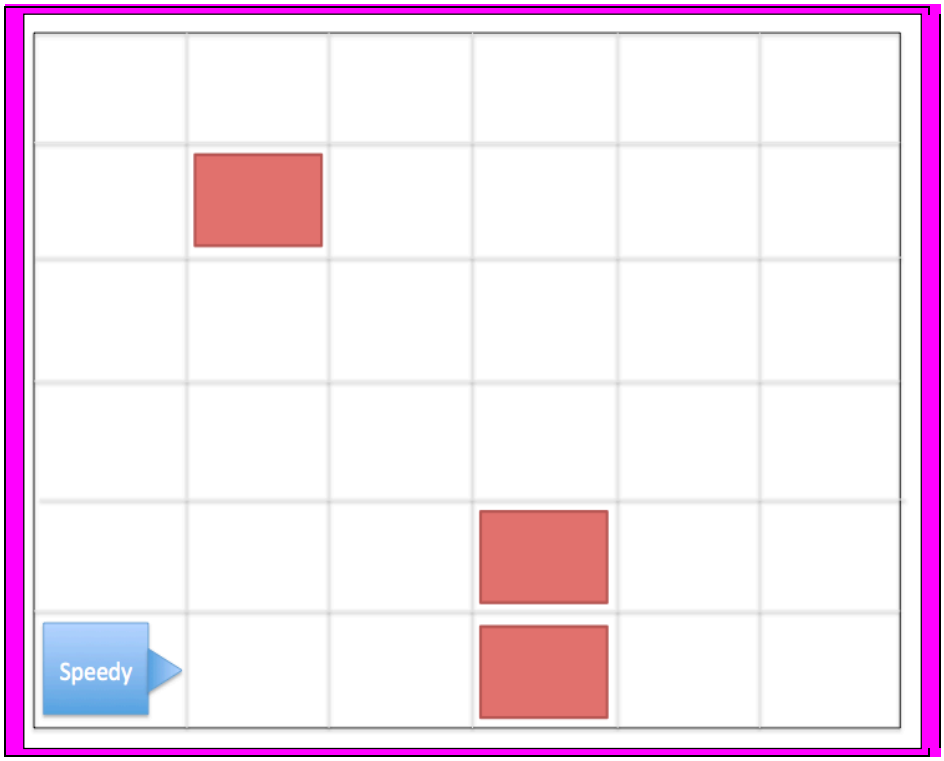

Figura 1 - Stanza con ostacoli

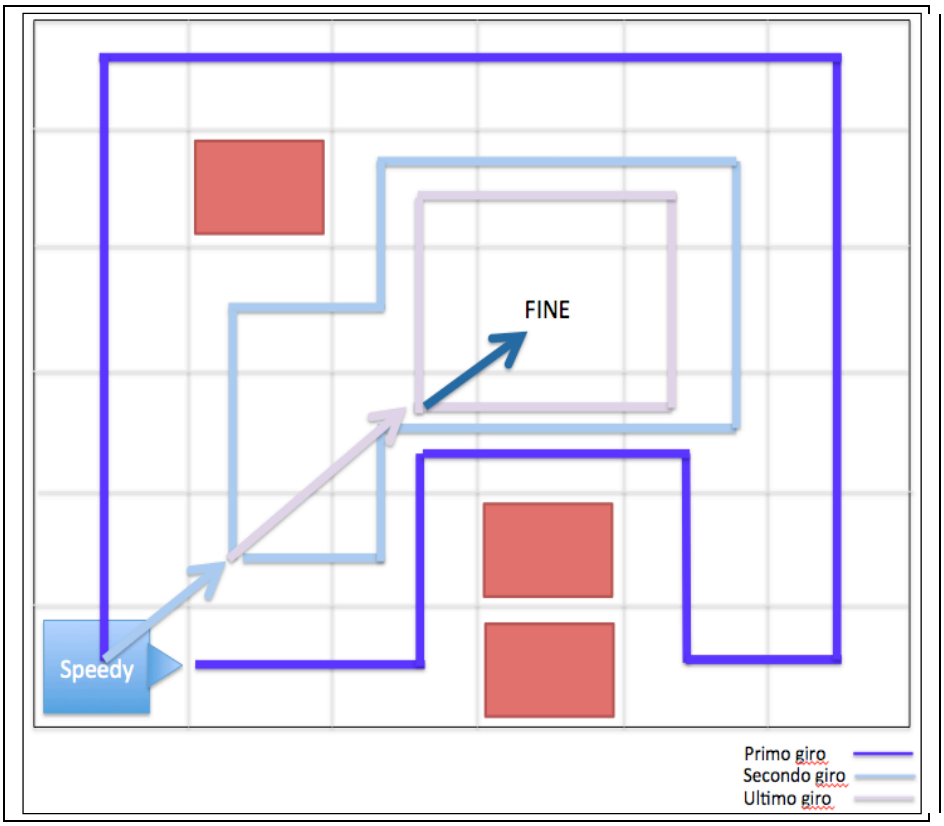

Figura 2 - Esempio di traiettoria a spirale in stanza con ostacoli

3.2. Idea 2: matrice con goal

La soluzione finale è stata trovata implementando una matrice che rispecchi la forma della stanza. La matrice creata è sostanzialmente una griglia in cui ogni punto di intersezione tra le linee è un "goal" che il robot vuole raggiungere per poi poterlo pulire. Gli ostacoli presenti nella stanza sono mappati nella matrice seguendo la tecnica "occupancy grid mapping".

L'algoritmo funziona nel seguente modo: il robot calcola inizialmente le coordinate di un goal da raggiungere (sulla base delle informazioni che ha della stanza) e tenta di raggiungerlo; se il punto è raggiungibile lo segna come "pulito", viceversa lo segna come "irraggiungibile" (inserendo un opportuno valore nella matrice). L'algoritmo procede iterativamente continuando a calcolare nuovi goal e tentando di raggiungerli fino a che non sono stati marcati completamente tutti i punti della stanza. A questo punto la matrice è di fatto una mappa molto approssimata della stanza.

Con la soluzione matriciale, utilizzando un buon algoritmo per il calcolo dei punti da raggiungere, è possibile pulire la stanza con il minore tempo possibile, poiché in molte situazioni si passa per un certo punto un'unica volta. Inoltre con l'individuazione degli ostacoli all'interno della mappa è possibile aggirarli con successo. Rispetto al problema precedente abbiamo dovuto aggiungere un parametro in più in input: la dimensione della stanza, necessario per dimensionare la matrice che la rappresenta.

Nel seguito è descritto nel dettaglio l'algoritmo finale utilizzato, delle assunzioni necessarie per il suo funzionamento all'implementazione dello stesso.

3.2.1. Assunzioni

Di seguito sono elencate le assunzioni necessarie per il funzionamento dell'algoritmo:

1. Le dimensioni della stanza sono conosciute a priori;
2. La stanza deve avere una forma rettangolare;

3. Gli ostacoli non cambiano posizione nel tempo;
4. Gli ostacoli non sono “eccessivamente” grandi (ciò verrà spiegato in seguito)
5. Il robot è inizialmente posizionato in un angolo prefissato della stanza, come in Figura 3;
6. Tutti i punti liberi da ostacoli sono fisicamente raggiungibili dal robot;

3.2.2. Algoritmo

Il seguente frammento di pseudo-codice illustra la struttura dell’algoritmo ad alto livello:

1. Lettura degli input e inizializzazione dei parametri. Si impostano le dimensioni della stanza e i parametri di inizializzazione delle action di ARIA (ad esempio distanza minima per la quale si riconosce la presenza di un ostacolo).
2. Inizializzazione della matrice con tutti i valori impostati come “punti da visitare” (Figura 3).
3. Calcola il primo goal da raggiungere.
4. **While** (ci sono ancora punti da visitare)
    4.1. **If** (l’obiettivo è raggiunto)
        4.1.1 Marca il punto raggiunto come pulito.
        4.1.2 Calcola un nuovo goal.* Se non ce ne sono, esci dal ciclo.
    4.2. Else
        4.2.1 Muoviti verso l'obiettivo.
        4.2.2 **If** (l’obiettivo non è raggiungibile)**
            4.2.2.1.1 Marca il punto come “non raggiungibile”.
            4.2.2.1.2 Calcola un nuovo goal. Se non ce ne sono, esci dal ciclo.
5. Torna alla posizione iniziale.

*Il calcolo di un nuovo punto goal è fatto tramite una semplice procedura “greedy”. L’algoritmo calcola le distanze tra la posizione attuale e tutti i punti non ancora visitati, e sceglie come nuovo goal quello più vicino.

** Questa condizione è valutata prendendo in considerazione la distanza del robot dal goal e le distanze misurate dai sonar frontali: se il robot si trova “nei paraggi” del goal e i sensori rilevano un ostacolo, si suppone che l’ostacolo sia nel punto goal, che viene marcato come “non raggiungibile”.

E' questo il motivo per cui gli ostacoli non devono essere più grandi di una certa dimensione. Infatti si supponga che sia presente uno ostacolo che occupi un'area pari a 9 celle della matrice (cioè una sottomatrice 3x3). A un certo punto dell'esecuzione il robot avrà come goal il punto corrispondente alla cella centrale della sottomatrice. Il robot, oltre non poter fisicamente raggiungere tale punto, non può nemmeno portarsi nelle sue vicinanze, il che gli impedirebbe di marcarlo come “irraggiungibile” e ciò porterebbe ad una situazione di deadlock. Tale situazione potrebbe essere rivelata tramite opportuni controlli sulla matrice, ma ciò esula dallo scopo di questo progetto.

Inoltre, questo criterio per discriminare se un punto è raggiungibile o meno non è esente da errori. Potrebbe capitare che il punto goal sia fisicamente raggiungibile ma che un ostacolo nelle vicinanze del goal inganni il robot e faccia marcare il punto come “occupato”. Nonostante ciò, dalle simulazioni si è visto che il robot giungerebbe comunque nelle vicinanze del goal, e di fatto pulirebbe in modo soddisfacente l’area circostante.

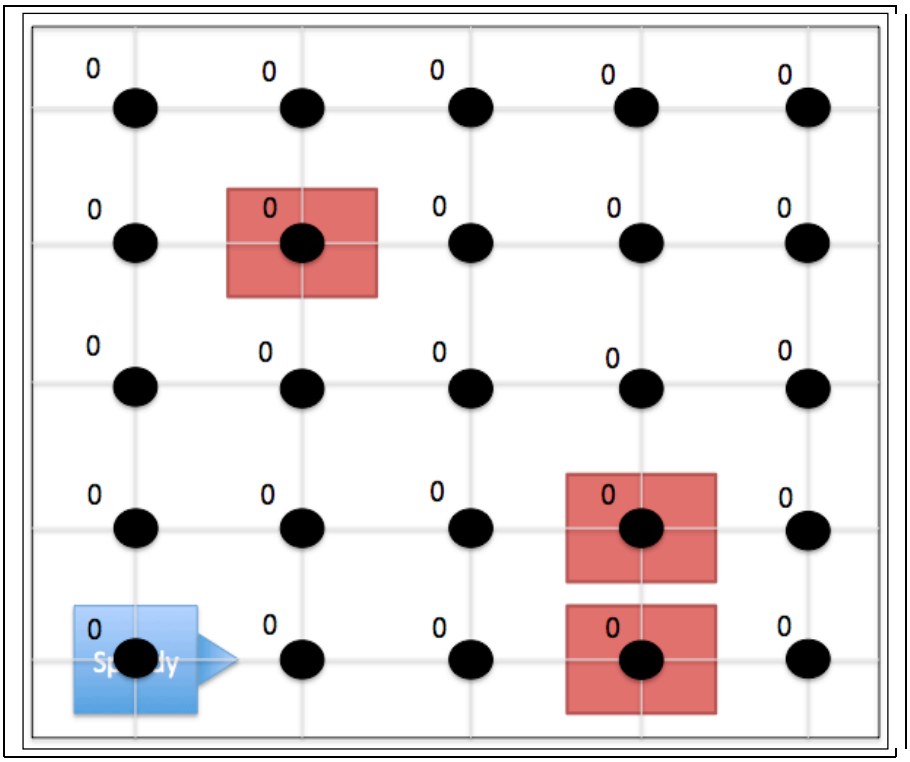

Figura 3 - Matrice (5,5) di una stanza con 3 ostacoli

3.2.3. Implementazione

L'algoritmo finale è stato implementato in un unico file C++, chiamato *aspira.cpp*. La funzione *main* gestisce interamente il comportamento del robot combinando operazioni sulla matrice, per memorizzare lo stato della stanza, e l'uso di Action preesistenti nella libreria ARIA, per regolare i movimenti del robot. Inoltre, per l'implementazione della matrice è stata definita una classe "Matrice" e una classe "Punto" definite negli header *matrice.h* e *punto.h*, i quali contengono i prototipi delle funzioni che implementano le operazioni di base su queste strutture dati.

Dopo avere creato una variabile che rappresenta il robot e una che rappresenta i suoi sensori, sono create varie azioni, che, con parametri opportuni, permettono al robot di recuperare da eventuali stalli dei motori e di evitare ostacoli frontali, e di raggiungere un determinato punto della stanza; le azioni sono, in seguito, aggiunte al robot, ciascuna con una certa priorità. Le action della libreria ARIA utilizzate sono:

```
ArActionStallRecover sbloccati("sbloccati");
ArActionAvoidFront avoidFrontNear("Avoid Front Near", distanza_max_ostacolo, 60, 15, true);
ArActionGoto vaiVerso ("goto");
        vaiVerso.setSpeed(velocita); //velocità di movimento
        vaiVerso.setCloseDist(distanza_max_punto); //distanza per cui l'obiettivo viene considerato raggiunto
```

E' stato necessario includere la action "sbloccati", in quanto dalle simulazioni capitava che il robot si bloccasse contro gli ostacoli mentre tentava di raggiungere il goal, nonostante la action "avoidFrontNear" fosse già impostata. I parametri con cui sono state inizializzate le action sono stati impostati in modo empirico dopo numerose simulazioni, per permettere al robot di eseguire un movimento il più regolare possibile.

Le priorità con cui tali action sono valutate in ambiente di esecuzione sono:

```
robot.addAction(&sbloccati, 90);
```

```
robot.addAction(&avoidFrontNear, 80);
robot.addAction(&vaiVerso, 70);
```

Le celle della matrice hanno dimensioni leggermente maggiori di quelle del robot. Nonostante questa assunzione si può pensare che il robot che raggiunge un goal pulisca un'area pari a quella di una cella della matrice. Il fatto che debbano essere "leggermente maggiori" è stato necessario per rimediare a blocchi del robot che si verificavano durante le simulazioni, e per dare al robot maggiore margine nelle operazioni di rotazione. Nelle simulazioni riportate in seguito si è utilizzato con risultati soddisfacenti un coefficiente di 1.2, in questo modo:

```
int lungh_rob = robot.getRobotLength();
int lato = lungh_rob*1.2;
```

Il raggiungimento di un goal viene valutato grazie alla funzione *haveAchievedGoal* della classe *ArActionGoto*, in questo modo:

```
if (vaiVerso.haveAchievedGoal())
        {…
        }
```

Si è utilizzata la classe *ArPose* per la lettura delle posizioni assunte dal robot, nel seguente modo:

```
ArPose posiz = robot.getPose();
        int posX = posiz.getX();
        int posY = posiz.getY();
```

dove il punto (posX,posY) rappresenta le coordinate in cui si trova il robot. Il sistema ha come centro la posizione iniziale del robot, per cui il robot calcola la sua posizione in modo "relativo", avendo come riferimento il punto di partenza. Questo è l'unico metodo utilizzato per stabilire la posizione del robot: è importante quindi che il esso abbia una buona correttezza cinematica e la sua posizione non venga modificata manualmente, altrimenti ciò porterebbe a una traiettoria errata.

Ogni 500 millisecondi vengono eseguito ciclicamente i controlli principali dell'algoritmo, come al punto 4 del paragrafo 3.2.2. Ad ogni ciclo viene fatto eseguire un output dei valori della matrice, avente dei valori arbitrari che rappresentano lo "stato" del rispettivo punto nella stanza. In particolare abbiamo usato i valori:

- 0 : punto ancora da visitare;
- -1: punto in cui il robot è passato e segnato come "pulito";
- 8: punto in cui è stato trovato un ostacolo e quindi irraggiungibile

Nella Figura 4 è rappresentato un esempio di stato finale della matrice. Come si può intuire la condizione di fine lavoro è che la matrice non contenga alcuno '0', che equivale a dire che tutti i punti sono stati visitati. Le screenshot vere e proprie delle simulazioni sono riportate al cap. 5.

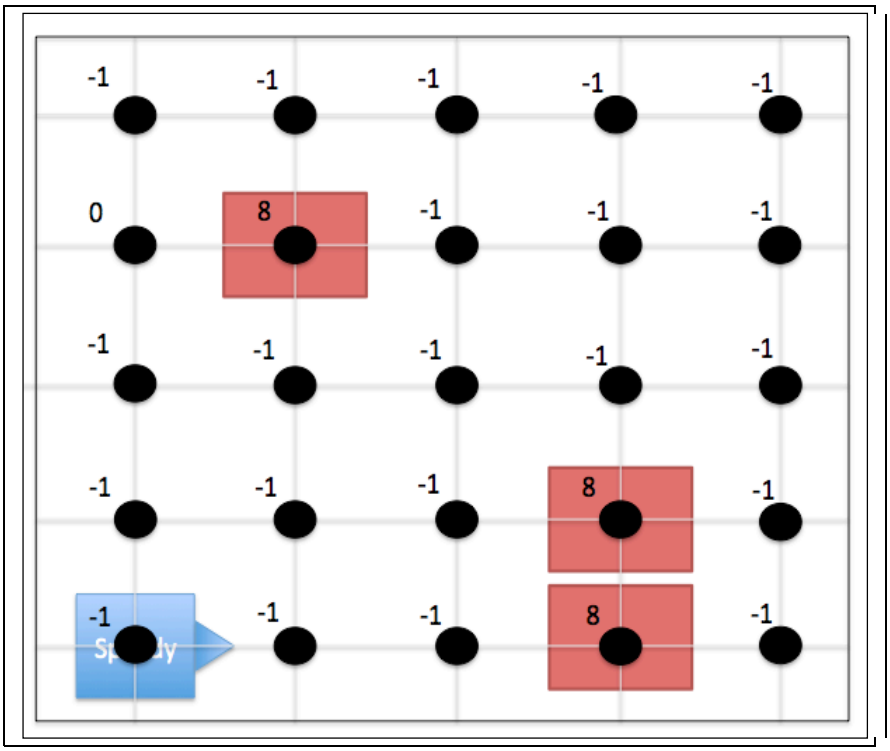

Figura 4: Matrice dei goal dopo l'esecuzione dell'algoritmo

4. Modalità operative

Il programma è stato scritto in C++, utilizzando le librerie Aria, su sistema operativo Ubuntu. È stato pensato per un robot del modello Pioneer1 della ActivMedia, nel caso specifico per il robot *Speedy* del laboratorio ARL.

In questo capitolo sono mostrati gli strumenti necessari per utilizzare il software realizzato.

4.1. Componenti necessari

Per fare funzionare il software è necessario avere un PC linux con installato la libreria LibAria e il simulatore MobileSim.

4.2. Modalità di installazione

Il programma per essere eseguito non necessita di alcuna installazione; i file aspira.cpp, matrice.h, e punto.h devono essere copiati e successivamente compilati attraverso il comando "make aspira" all'interno della directory che li contiene (meglio se nella directory "examples" del pacchetto Aria, in modo da utilizzare un makefile corretto). Successivamente, il programma viene avviato digitando il nome dell'eseguibile: ./aspira.

4.3. Modalità di utilizzo

Per utilizzare il software è necessario aprire il simulatore MobileSim e selezionare una mappa presente in /usr/lib/Aria/maps (selezionare come robot p3at). A questo punto aprire il terminale ed eseguire il file *aspira*, in questo modo partirà la simulazione.

5. Simulazione

Abbiamo svolto come dimostrazione tre simulazioni complete con mappe create ad hoc, la prima in una stanza senza ostacoli e la seconda in una stanza con un ostacolo e la terza in una stanza con più ostacoli. In seguito sono riportati gli screenshot delle situazioni, che mostrano sia la traiettoria vera e propria in MobileSim, sia lo stato della matrice di output del programma. In MobileSim, per visualizzare la traiettoria compiuta dal robot bisogna selezionare la voce “Show Trails” dal menu “View”.

5.1. Simulazione senza ostacoli

In Figura 4 è mostrato un esempio di simulazione dell’algoritmo in una stanza senza ostacoli.

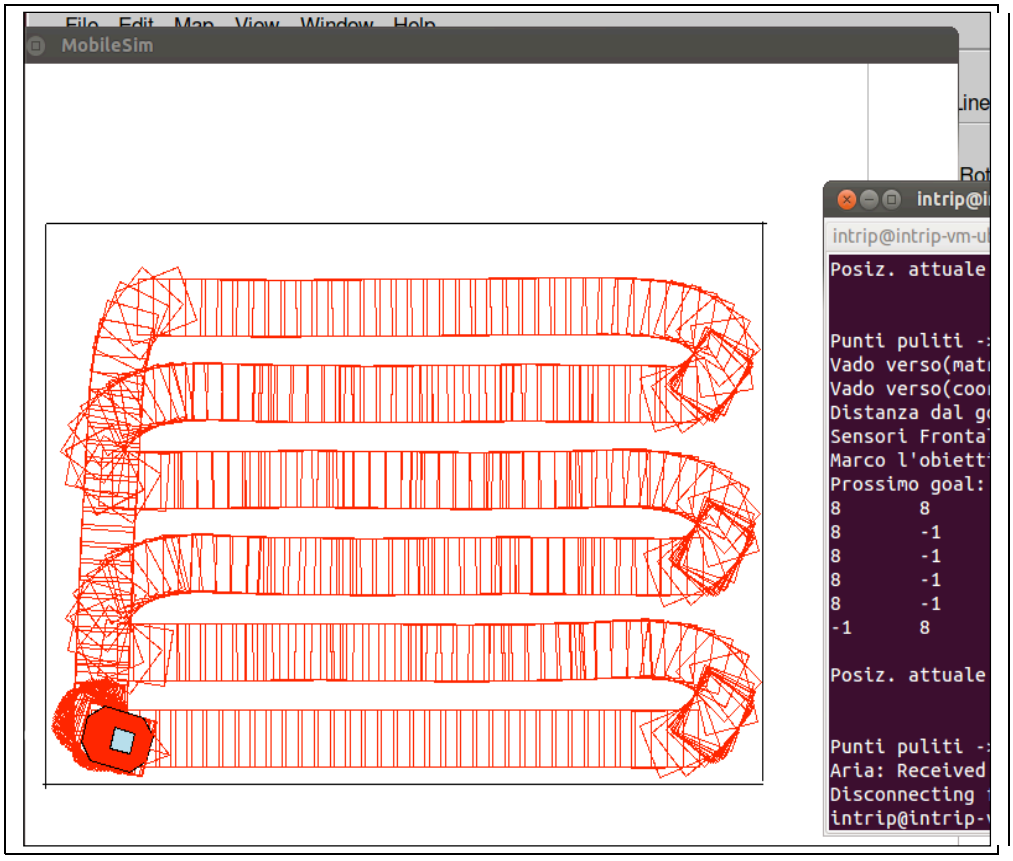

Figura 4 - Simulazione dell’algoritmo in una stanza senza ostacoli

5.2. Simulazione con ostacoli

In Figura 5 viene mostrata la simulazione dell’algoritmo in una stanza con un ostacolo.

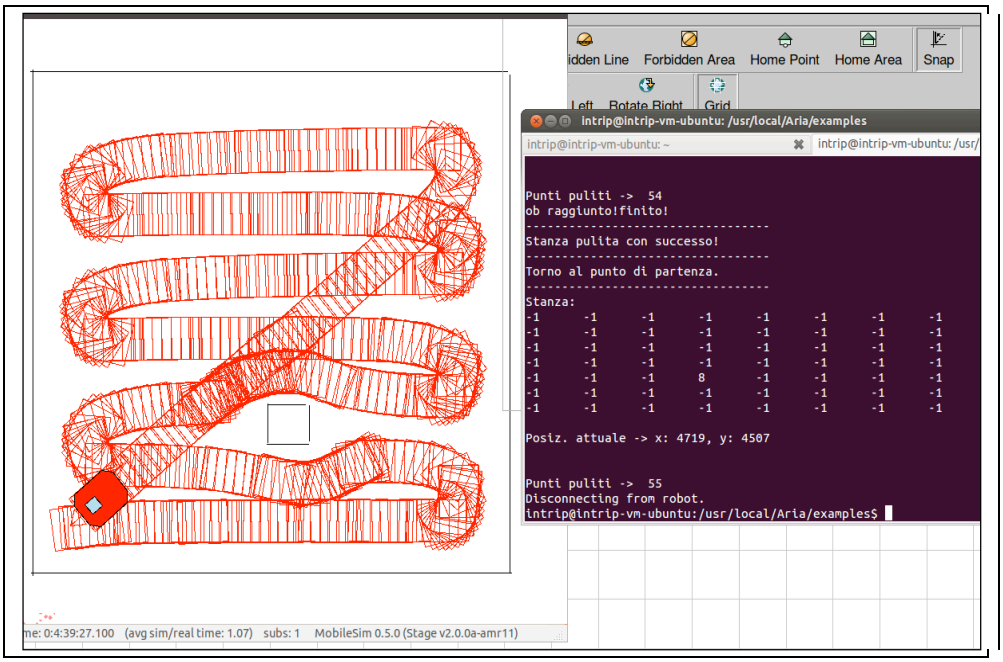

Figura 5 Simulazione dell'algoritmo in una stanza con un ostacolo

In Figura 6 viene mostrata la simulazione con una stanza con più ostacoli.

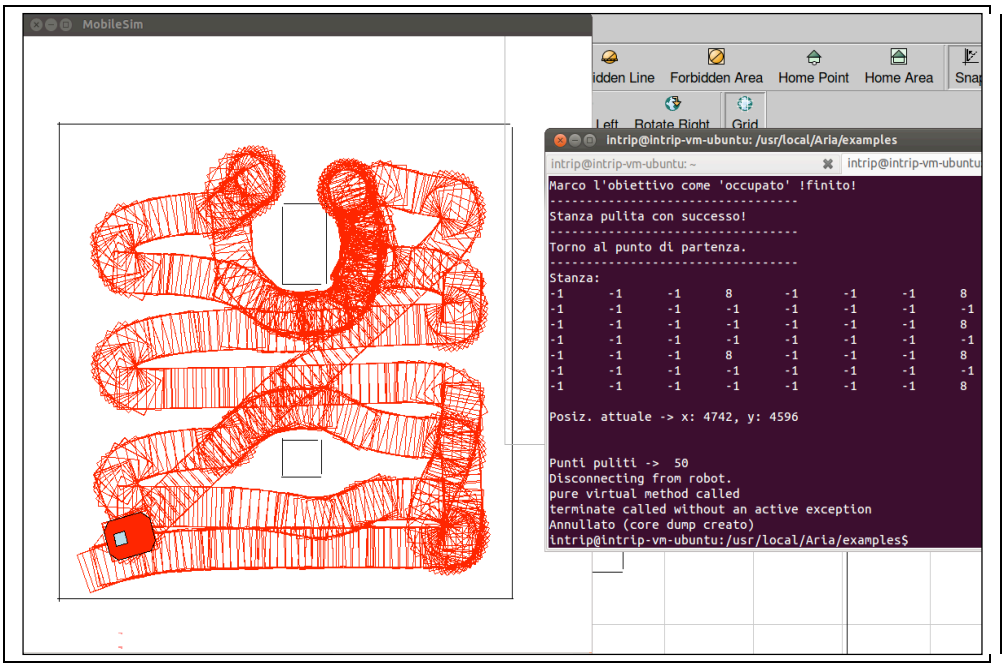

Figura 6 Simulazione dell'algoritmo in una stanza con più ostacoli

6. Conclusioni e sviluppi futuri

In conclusione è possibile affermare che l'obiettivo prefisso prima di intraprendere questa esperienza è stato raggiunto in modo soddisfacente. Infatti, a patto di rispettare le precondizioni descritte nell'introduzione, il comportamento che il programma impone al robot è quello desiderato nella maggior parte delle occasioni.

Pur con le assunzioni fatte ci sono altri fattori che possono determinare la non precisa esecuzione dell'algoritmo. Infatti l'uso di un sistema di coordinate assolute per la localizzazione del robot può essere fonte di errori che si accumulano nel tempo e che possono portare al "disorientamento" del robot, che potrebbe non compiere di fatto il percorrimento completo della stanza. Uno sviluppo futuro potrebbe consistere nell'uso di altro sistema per la localizzazione, che faccia uso di coordinate assolute. In tal modo, sarebbe possibile far partire il robot da un punto qualunque della stanza e non necessariamente da un angolo.

Ulteriori sviluppi di questo progetto potrebbero riguardare l'estensione del programma per stanze con forma non regolare, o di cui siano sconosciute a priori le dimensioni. Inoltre, all'aumentare delle dimensioni della stanza, potrebbe essere necessario ottimizzare la procedura che calcola il prossimo goal da raggiungere: un algoritmo "greedy" come quello implementato in questo progetto potrebbe risultare inefficiente in stanze di grandi dimensioni e portare a percorrere più volte punti già puliti. Un algoritmo più complesso potrebbe portare a delle soluzioni di percorrimento vicine al cosiddetto "cammino ottimo".

Bibliografia

[1] ActivMedia Robotics.: "CMPE 300 ARIA LAB MANUAL", 2002

[2] Gini, G., Caglioti, V.: "Robotica", Zanichelli, 2003.

Indice