ALMA MATER STUDIORUM - UNIVERSITÁ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea in Ingegneria Informatica, Elettronica e Telecomunicazioni

ROBOT PATROLLING: ANALISI DI STRATEGIE E APPLICAZIONE AD UNO SCENARIO

ELABORATO FINALE DI LAUREA IN
Fondamenti di Informatica B

Relatore:
Prof. Ing.
ANDREA ROLI

Presentata da:
SIMONE GROTTI

Sessione II
Anno Accademico 2012-2013

Con le buone si ottiene tutto.
Devi tenerlo sempre a mente,
è questo il trucco.

Indice

Elenco delle figure

Introduzione

Con il termine patrolling si indica l'atto del pattugliare un'area, ad intervalli regolari, in modo da proteggerla o supervisionarla. La qualità di una strategia di patrolling può essere valutata secondo diversi criteri ma, in maniera del tutto informale, possiamo dire che una buona strategia è quella che minimizza il tempo trascorso tra due passaggi consecutivi per lo stesso luogo. È evidente come la principale utilità di questa attività sia quella di proteggere zone di particolare interesse, come ad esempio banche, uffici, accampamenti militari o, più in generale, un qualunque luogo dove possa essere necessario difendere delle risorse da un ipotetico intruso.

Nonostante la sua evidente utilità, solo negli ultimi tempi il patrolling è stato oggetto di studi approfonditi. Questo ha portato a galla quella che è la vera difficoltà nello studio teorico, e quindi nella successiva messa in pratica dei risultati raccolti: le casistiche da prendere in considerazione e i punti di vista dai quali può essere analizzato il problema del patrolling sono innumerevoli.

Ad esempio, è fondamentale definire quanti siano gli agenti destinati a svolgere l'operazione di ricognizione: infatti se il patrolling è compiuto da più agenti allora è necessario anche specificare come questi agenti si comportano nell'ambiente e come nei confronti gli uni degli altri.

O ancora, anche il numero degli ipotetici ladri può variare e con esso anche il loro comportamento, che può essere razionale o non razionale, ovvero l'intruso può osservare il pattugliatore e le mosse che compie ed agire in base ad esse. Si rende

quindi necessaria anche la definizione di un comportamento per il supervisore, che può essere predicibile o non predicibile.

Infine, anche l'ambiente può presentare diverse caratteristiche: possiamo immaginarlo come caratterizzato dalla presenza di ostacoli mobili oppure da una impossibilità a comunicare come in diverse simulazioni militari. O ancora, possiamo prediligere un patrolling mirato al pattugliamento di un intero perimetro di un'area oppure finalizzato alla protezione di solo alcune regioni di tale area, che possono avere anche diversa rilevanza.

È evidente quindi che gli scenari che possono essere studiati sono innumerevoli e che le casistiche e i vincoli da prendere in considerazione sono numerosi. Proprio per questo motivo durante l'analisi teorica di uno scenario di patrolling alcuni di questi vincoli vengono rilassati: ad esempio il tempo può essere discretizzato, si suppone che sia il difensore che l'invasore impieghino tempi prestabiliti per compiere determinate azioni, o che non vi possano essere malfunzionamenti che minino il corretto funzionamento degli agenti.

L'obiettivo principale di questa tesi è quello di definire una strategia di patrolling che sia affidabile e permetta ad uno o più pattugliatori di proteggere un'area da possibili violazioni. In seguito verrà descritto uno scenario valutandone le caratteristiche e modellando la strategia di patrolling precedentemente definita rendendola idonea a garantire sicurezza per quel particolare caso di studio; si passerà poi alla definizione del comportamento del robot ricognitore, realizzato tramite Lego Mindstorms, le cui caratteristiche verranno descritte al meglio nei capitoli successivi.

La tesi è strutturata come segue. Nel capitolo 1 è riassunto lo stato dell'arte attuale per quel che riguarda il problema del pattugliamento, presentando alcune delle metodologie impiegate per risolverlo. Nel capitolo 2 definiremo quella che sarà la nostra strategia di patrolling e nel capitolo 3 introdurremo il nostro caso di studio, descrivendo le caratteristiche dello scenario e i motivi per i quali lo

abbiamo scelto. Verrà poi realizzata una descrizione della realizzazione del patroller che intendiamo utilizzare per tentare di effettuare l'operazione di ricognizione. Successivamente, termineremo il nostro elaborato discutendone le conclusioni e analizzando alcuni possibili sviluppi futuri.

Capitolo 1

Stato dell'arte

L' obiettivo di questo capitolo è quello di introdurre il problema del pattugliamento, fornendo una panoramica generale sugli aspetti principali del patrolling e, successivamente, raccogliendo ed illustrando alcuni degli approcci e delle metodologie finora impiegati per la sua risoluzione. In questo modo verrà ricavato un ampio quadro sullo stato dell'arte attuale, permettendo una più facile comprensione delle principali questioni da analizzare quando, successivamente, dovremo scegliere una strategia di patrolling per il nostro scenario.

1.1 Il problema del pattugliamento

Con il termine pattugliamento, letteralmente si intende l'atto di muoversi attorno ad un'area ad intervalli regolari per proteggerla e sorvegliarla da eventuali intrusi. Come già detto, è evidente che le applicazioni pratiche possono essere molto interessanti: difesa di banche, uffici, accampamenti militari o, più in generale, di un qualunque luogo dove possa essere necessario difendere delle risorse da un ipotetico intruso; proprio per questi motivi, negli ultimi anni l'argomento è divenuto particolarmente interessante e la ricerca ha dedicato sempre maggiori risorse ed

attenzioni a questo problema.

Andiamo quindi ad analizzare gli elementi principali che compongono uno scenario di patrolling:

- uno o più robot pattugliatori;
- un'area all'interno della quale i robot possono muoversi;
- un insieme di target interni a quest'area che devono essere protetti, con la possibilità che questi presentino diversi livelli di priorità;
- uno o più intrusi che cercano di violare uno o più target e devono essere identificati.

Inoltre possiamo distinguere due differenti tipologie di pattugliamento, a seconda di come i pattugliatori si comportano:

- **pattugliamento perimetrale**: uno (o più) robot pattugliano il perimetro dell'area di interesse impedendo a qualcuno di entrarvi;
- **pattugliamento ad obiettivi**: ad essere difesi devono essere i target posti all'interno dell'area.

Definite le caratteristiche principali dello scenario e la tipologia del pattugliamento, come già accennato in precedenza abbiamo un grande numero di fattori da tenere in considerazione. Ad esempio, una volta definito il numero di agenti in ricognizione, se il patrolling è compiuto da più pattugliatori allora è necessario anche specificare se questi agenti sono in comunicazione tra loro o meno e se agiscono simultaneamente su tutto l'ambiente oppure se questo è partizionato ed ogni zona viene assegnata ad un singolo agente; è evidente che nel caso in cui il patrolling sia eseguito da un solo supervisore queste questioni non necessitano di essere definite.

Inoltre, il comportamento degli agenti - sia pattugliatori, sia intrusi - può essere definito in diversi modi; per quanta riguarda gli intrusi esso può essere razionale

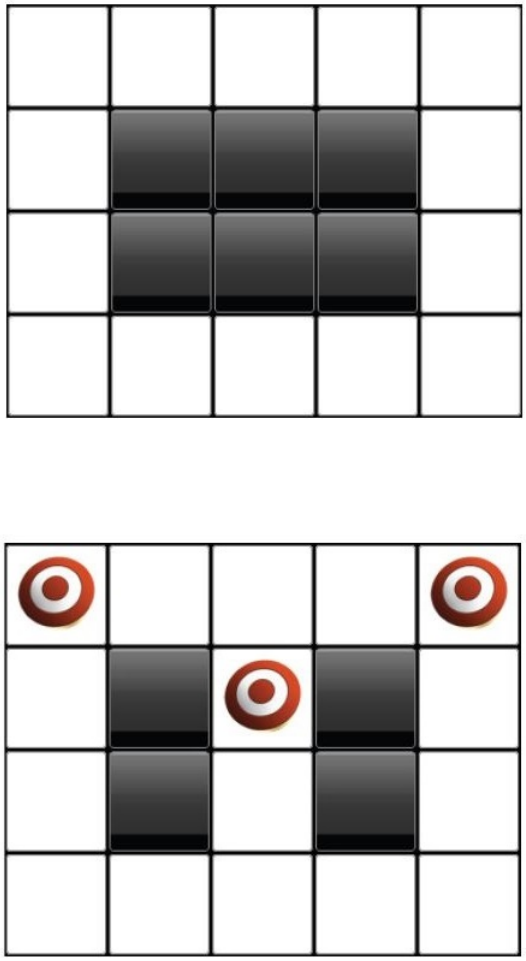

Figura 1.1: Esempio di pattugliamento perimetrale (in alto) e ad obiettivi (in basso)

o non razionale, ovvero l'invasore può osservare il pattugliatore e le mosse che compie ed agire in base ad esse. Per quanto riguarda il supervisore, invece, esso può avere un comportamento *predicibile* o *non predicibile*; si noti però come in quest'ultimo caso si tratterebbe sostanzialmente di un comportamento casuale che non garantirebbe la visita di tutti i punti di interesse.

1.2 Lavori riguardanti il pattugliamento

Come più volte ripetuto, negli ultimi anni sono stati svolti numerosi studi e ricerche riguardo al problema del patrolling, dando sempre più rilevanza alla questione. Il patrolling è per definizione un problema multi agente: si parla di *multi-robot patrolling* per indicare un sistema multi robot che si muove continuamente all'interno di un ambiente per monitorarlo.

Come descritto in [8] le principali strategie per realizzare tecniche di patrolling sono due: quella dell'ottimizzazione, che mira ad ottimizzare il percorso scelto per gli agenti, e quella *intrusion detection-based*, che si occupa principalmente di garantire una cattura dell'intruso.

La prima tipologia di strategie, quella per noi più interessante visto il tipo di lavoro che andremo a svolgere, deve la sua fortuna principalmente al lavoro di Machado et. al. [4], dove viene descritto il problema del patrolling dal punto di vista del percorso, che viene mappato come un insieme di nodi e permettendo di utilizzare in questo modo i risultati raccolti in decenni di studi della Teoria dei Grafi. In questo modo il problema si riduce al semplice minimizzare l'intervallo di tempo tra due passaggi successivi per lo stesso nodo; inoltre gli archi possono essere caratterizzati da un peso che, mantenendo le proporzioni tra i vari archi del grafo, può rappresentare la reale distanza tra due nodi.

In [4] inoltre vengono proposte diverse architetture che si differenziano tra loro per il variare delle caratteristiche dell'ambiente, degli agenti e del loro comportamento. Gli esperimenti da loro condotti fornirono interessanti risultati, due dei

quali influenzarono molti degli studi sugli scenari di patrolling realizzati in seguito: *in primis* essi si accorsero che agenti che si muovono casualmente raggiungono dei risultati pessimi; *in secundis* notarono che degli agenti senza alcuna abilità comunicativa e che si muovono cercando semplicemente di minimizzare l'*idleness* sono in grado di raggiungere i risultati che solitamente erano raggiunti dai loro più complessi algoritmi.

Il concetto di *idleness* di cui abbiamo appena parlato venne introdotto per la prima volta da Chevaleyre in [3]: con questo termine egli indica il quantitativo di tempo per il quale ogni nodo dell'area è rimasto 'incustodito'. Nel suo lavoro egli analizza il multi robot patrolling eseguito con due distinte tecniche: nel primo caso, presa un'area, questa veniva partizionata ed ognuna di queste parti veniva assegnata ad un agente; nel secondo caso, invece, l'intera area veniva continuamente percorsa da tutti gli agenti, i quali percorrevano lo stesso tragitto in maniera consecutiva. Dopo diversi casi di studio egli arrivò alla conclusione che, pur constatando il fatto che ogni ambiente può essere più predisposto per l'una piuttosto che per l'altra tecnica, tendenzialmente i risultati migliori si ottengono non partizionando l'area bensì facendo eseguire lo stesso percorso a tutti gli agenti in modo tale da ridurre drasticamente l'idleness dell'intero ambiente. Risultati del tutto simili furono raggiunti anche da Almeida et. al. in [5].

Un lavoro molto interessante è sicuramente [7]; solitamente, le tecniche di patrolling vengono realizzate in maniera non deterministica così da garantire imprevedibilità e impedire agli intrusi di studiare i comportamenti dei patroller. In [7] viene cambiato il punto di vista: si introduce il concetto di tempo di penetrazione, ovvero il tempo che un possibile intruso impiega per accedere ad un qualsiasi target. Si vuole quindi 'forzare' gli intrusi a non entrare, e questo è reso possibile creando una strategia deterministica che minimizza l'idleness al punto tale che questa sia troppo piccola per permettere ad un intruder di violare uno dei target.

Prendendo come riferimento i risultati enunciati fino ad ora, possiamo prosegui-

re definendo quello che sarà lo scenario da noi realizzato per garantire una buona strategia di patrolling.

Capitolo 2

La strategia di patrolling

Arrivati a questo punto, formalizzeremo uno scenario di pattugliamento partendo dalle considerazioni fatte nel capitolo precedente e dalle conclusioni raccolte. Verrà quindi data una breve descrizione delle caratteristiche che dovrà possedere il nostro caso di studio, e giustificheremo anche le scelte fatte. Saranno poi toccati alcuni argomenti strettamente correlati alle decisioni prese ed infine verrà data una descrizione di cosa utilizzeremo per realizzare il nostro scenario.

2.1 Scenario di pattugliamento

Come già spiegato in precedenza, l'opportunità di rappresentare un determinato ambiente su cui eseguire il patrolling tramite un grafo è per noi una possibilità di grande aiuto. Infatti descrivere un ambiente tramite un grafo ci permette di prescindere dalle caratteristiche dell'ambiente stesso: l'importante è che il valore numerico associato ad ogni arco identifichi l'effettiva distanza tra i due nodi che esso collega. Immaginiamo inoltre una situazione in cui i vari nodi non sono caratterizzati da pesi diversi e decidiamo che essi siano tutti importanti in egual modo, perché - come vedremo in seguito - questo dettaglio perde rilevanza nel contesto

che andremo ad analizzare.

Decidiamo, per semplicità, di avere un singolo pattugliatore e per lo stesso motivo un solo intruso; questi due 'protagonisti' sono contraddistinti dalle seguenti caratteristiche:

- il patroller segue un comportamento deterministico, e quindi predicibile;
- il 'ladro' si assume essere razionale, e quindi che possa apprendere osservando.

Il pattugliatore quindi esegue ciclicamente lo stesso cammino; è evidente che in questa circostanza il valore di idleness sia semplicemente dato dalla somma di tutte le singole distanze tra i vari nodi del grafo, ovvero - per quanto detto in precedenza - dei pesi degli archi.

Prendendo spunto da quanto realizzato in [7], si vuole prescindere da quello che è il comportamento dell'intruso e questo è possibile minimizzando l'idleness a tal punto da renderla minore del minor tempo di penetrazione, così che il ladro non potrà con certezza violare alcun target.

2.2 Lavori affini

Come più volte detto in precedenza, il fatto di poter ricondurre un generico percorso di patrolling ad un grafo è per noi molto utile per la semplicità con la quale esso permette di raggiungere una rappresentazione del nostro percorso che sia fedele ed attendibile. Non solo, un altro indiscutibile vantaggio è quello di poter mettere in pratica i risultati raccolti in decenni di studi sulla Teoria dei Grafi; nel campo della ricerca operativa troviamo un importante riscontro a quanto appena detto, permettendo di calcolare il cammino più breve che visiti tutti i nodi di un grafo tramite il cosiddetto *problema del commesso viaggiatore*.

Il problema del commesso viaggiatore (Traveling Salesman Problem - TSP) è un problema di ottimizzazione che richiede di individuare, dato un insieme di città su

una mappa, il più corto percorso ciclico in cui ogni città è visitata esattamente una volta. È intuibile come questo problema sia direttamente applicabile agli scenari di patrolling, ma va subito fatto notare come esso in un certo senso introduca delle limitazioni a quelle strategie di patrolling che decidessero di basarsi su di esso: infatti il TSP richiede che ogni nodo sia visitato *esattamente* una volta all'interno di un singolo ciclo, mentre questo nel contesto del patrolling, oltre a non essere necessario, a volte parrebbe essere limitativo. Conseguentemente a ciò, il TSP fu riformulato e venne esteso in modo da poter essere applicato al contesto dei grafi metrici: dato un grafo i cui archi siano pesati, trovare il più breve percorso chiuso in cui ciascuno dei nodi sia visitato almeno una volta.

Una soluzione al problema venne proposta da Christofides in [6], introducendo un algoritmo in grado di generare in $O(n^3)$ una soluzione che fornisca un cammino chiuso la cui lunghezza è al più pari ad 1.5 volte il più breve cammino chiuso. Chevaleyre in [2] dimostra che, nel caso di un singolo agente, la strategia ottimale che minimizza la peggior idleness è quella che adotta il cammino chiuso che è soluzione ottima del TSP.

Quest'ultima considerazione è di fondamentale importanza, soprattutto per quel che riguarda lo scenario che vogliamo studiare, e quindi il concetto verrà ribadito in modo che il suo valore sia colto a pieno. Prendiamo in considerazione una strategia basata sul cammino chiuso che è soluzione ottima del TSP: già era risaputo che tale strategia fosse la migliore possibile tra tutte le strategie cicliche per un singolo agente; Chevaleyre va oltre a questa considerazione e dimostra che questa strategia è la migliore possibile tra *tutte* le strategie single-agent. Al di là dell'indiscutibile valore assoluto di questo teorema, esso ci fornisce anche una importante indicazione su quello che sarà il percorso da seguire durante la realizzazione del nostro scenario.

Si tratterà quindi di 'tradurre' quello che sarà il nostro ambiente in un grafo metrico e individuare il cammino che è soluzione al TSP; potremmo affidarci a quello che è l'algoritmo di Christofides descritto in [6], che è in grado di fornire un

percorso *quasi ottimo* in un tempo molto breve. Invece, per valutare il percorso che seguiremo sul grafo, ci serviremo di un software open source, chiamato Concorde [9], che è in grado di fornire soluzioni più vicine a quella ottima del TSP rispetto all'algoritmo di Christofides e in un tempo più breve; una volta ottenuto un percorso chiuso, le performance degli agenti che effettueranno il patrolling su di esso verranno valutate, come già detto, calcolando l'idleness che essi sapranno garantire. Tale idleness sarà ottenuta semplicemente confrontando la velocità dei pattugliatori con la lunghezza totale del percorso, che essendo chiuso sarà data dalla somma dei pesi dei singoli archi.

Ovviamente questi calcoli si basano su considerazioni che vengono fatte tenendo conto che la situazione descritta presenta delle caratteristiche che non potranno verificarsi con certezza. Per esempio basti notare come il moto del patroller difficilmente sarà uniforme così come il tempo di penetrazione in una situazione reale non sarà definibile a priori. Quello che stiamo facendo è descrivere una situazione ideale che ci permettta di risolvere il problema, perchè senza le dovute assunzioni non è sempre possibile farlo.

Si noti come una volta ottenuto il valore dell'idleness, saremo in grado di verificare se tale valore sarà sufficientemente basso da poter garantire che non avvenga nessuna invasione. Se invece l'idleness non sarà minore rispetto al tempo di penetrazione allora basterà introdurre un secondo agente il cui comportamento sarà lo stesso del primo, ma che inizierà l'azione di patrolling dal punto opposto rispetto ad esso. In questo modo la frequenza con cui i pattugliatori visiteranno i target sarà doppia e quindi il valore dell'idleness dimezzerà. Si noti come questo modo di operare non necessiti di alcuna forma di comunicazione tra i due pattugliatori, che in linea di massima possiamo dire agiscono come due single-agent su uno stesso percorso: il lavoro svolto da uno non dipende in alcun modo da quello svolto dall'altro.

L'introduzione di un secondo pattugliatore porta la possibilità di compiere il

patrolling utilizzando due modalità differenti: una è quella che abbiamo appena descritto, mentre la seconda prevede la divisione del grafo in due sottografi ai quali vengono poi rispettivamente assegnati i singoli agenti. Come già descritto in precedenza, non vi è una regola che prestabilisce quale delle due tecniche utilizzare poiché ogni ambiente ha le proprie caratteristiche e in base ad esse si può prediligere l'una piuttosto che l'altra; ad esempio, un ambiente che presenta lunghi corridoi è più portato alla suddivisione in sotto-aree che escludono questi corridoi dal patrolling così da evitare che gli agenti perdano del tempo percorrendoli.

Ad ogni modo è stato dimostrato da Chevaleyre in [2] che tendenzialmente la strategia migliore da utilizzare è quella che prevede che i vari agenti pattuglino tutti sullo stesso tragitto; tenendo a mente ciò, notiamo come anche Machado et al. in [4] arrivarono ad un risultato del tutto simile: qui viene affermato che agenti senza alcun tipo di capacità comunicativa la cui strategia consista nel visitare i nodi con il più alto valore di idleness - quindi che eseguono un percorso ciclico - sono in grado di avvicinare i risultati che solitamente erano raggiunti dai loro più complessi algoritmi.

Tenendo conto di queste ultime considerazioni, quindi, nel caso in cui l'idleness garantita da un singolo pattugliatore non sia sufficientemente bassa e dovrà esserne inserito un secondo nell'ambiente, per diminuirla opteremo per l'introduzione di un secondo pattugliatore sul percorso già definito senza dover così dividere il grafo in più sottografi e riformulare il problema dall'inizio.

2.3 In sintesi

Concludiamo questa sezione riassumendo in un elenco puntato le principali caratteristiche del nostro scenario.

- Patrolling non perimetrale, bensì ad obiettivi.
- Ambiente rappresentato come grafo metrico.

- Un solo pattugliatore.
- Un solo invasore.
- Il comportamento del pattugliatore è predefinito e quindi predicibile.
- L'intruso si suppone essere il rivale più forte possibile: conosce la strategia del patroller.
- Per semplicità, il tempo si suppone essere discretizzato.
- Per violare un target l'intruso impiega un tempo predefinito detto *tempo di penetrazione*.
- I target non hanno diversi gradi di priorità, e comunque - anche se l'avessero - ciò non influenzerebbe la strategia di patrolling.
- Nel caso in cui venga inserito un secondo pattugliatore esso non comunicherà con gli altri.
- L'eventuale secondo pattugliatore verrà inserito all'interno del percorso già definito.

Capitolo 3

Realizzazione

Dopo aver definito quindi il nostro scenario, ora cercheremo di realizzarlo. Come è ovvio che sia, per effettuare del patrolling serve innanzi tutto un robot pattugliatore; nel nostro caso questo ruolo verrà ricoperto da un robot realizzato tramite LEGO Mindstorms, che diventerà quindi il protagonista del nostro scenario. In questo capitolo adotteremo un semplice caso di studio che useremo come esempio per applicare su di esso la nostra strategia; successivemante, dopo una panoramica sul sistema Mindstorms, vedremo in che modo verrà assemblato il patroller e come esso potrà svolgere il suo compito di pattugliatore. In seguito, mostreremo come il robot sarà in grado di seguire il percorso da noi definito tramite il software Concorde, che verrà brevemente descritto, e come esso interagirà con l'ambiente ed eventuali intrusi.

3.1 Il caso di studio

Una volta definite quelle che sono le carateristiche che il nostro scenario dovrà possedere, andiamo ora a definire formalmente tale scenario.

Come già detto all'inizio di questa tesi, lo scopo per cui può essere realizzata una operazione di patrolling è essenzialmente quello di difendere una o più risorse o, più

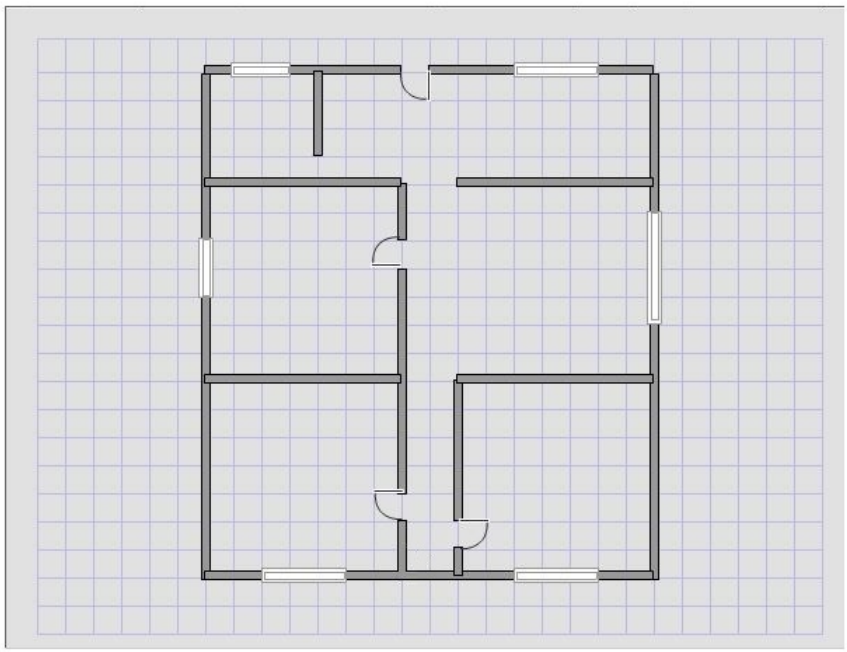

Figura 3.1: Pianta del piano terra di una abitazione

in generale, un'area che vogliamo rimanga inviolata. Ciò ha fatto sì che il patrolling venisse negli ultimi anni sempre più posto al centro dell'attenzione, grazie alla sua evidente utilità ma anche e soprattutto alla sua versatilità ed adattabilità agli scenari più diversi.

Andiamo ora a definire formalmente uno di questi scenari, e applichiamo su di esso la strategia di patrolling da noi definita nel capitolo precedente.

3.1.1 Analisi dell'ambiente

Come è logico pensare, non tutti gli scenari sono adatti alla strategia che andremo a descrivere; al pari di alcune situazioni in cui è richiesta una descrizione dell'ambiente sotto forma di grafo ne esistono altre in cui tale passaggio può essere inutile o quantomeno non necessario. Immaginiamo ad esempio di dover sorvegliare dall'interno il piccolo ambiente rappresentato in figura 3.1. Supponiamo che esso

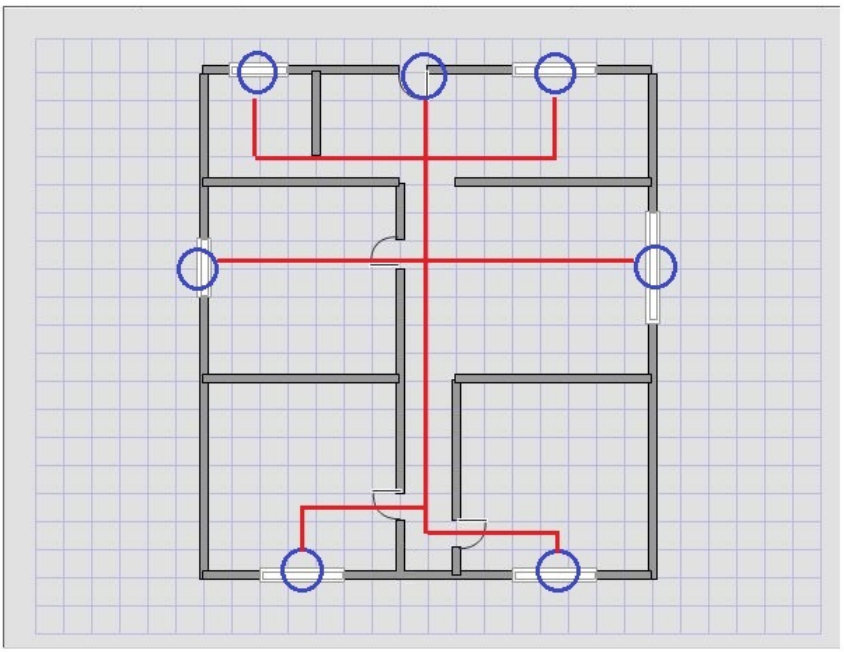

Figura 3.2: Individuazione target e percorso di pattugliamento

rappresenti la piantina del pian terreno di una abitazione. Innanzi tutto va considerato il fatto che il nostro robot pattugliatore potrà avere dei vincoli di natura fisica da dover rispettare; un banale esempio è quello di notare come probabilmente esso non sarà in grado di salire o scendere dalle scale, e questo ci impone di effettuare la nostra operazione di pattugliamento in un ambiente che si sviluppa su un livello soltanto.

O ancora, va tenuto presente che gli ambienti sono spesso caratterizzati da una suddivisione in stanze: se i target che il nostro patroller deve proteggere o comunque sorvegliare si trovano in stanze separate da pareti allora lo scenario diventa banale e il percorso di patrolling, da un punto di vista logistico, diviene scontato. Infatti, assunzione ancor più banale della precedente, è plausibile supporre che i pattugliatori non siano in grado di attraversare le pareti; in un ambiente come quello rappresentato in figura 3.1 un robot per spostarsi da una stanza all'altra dovrà necessariamente pasare dalle porte e il percorso diventerà prevedibile, se non

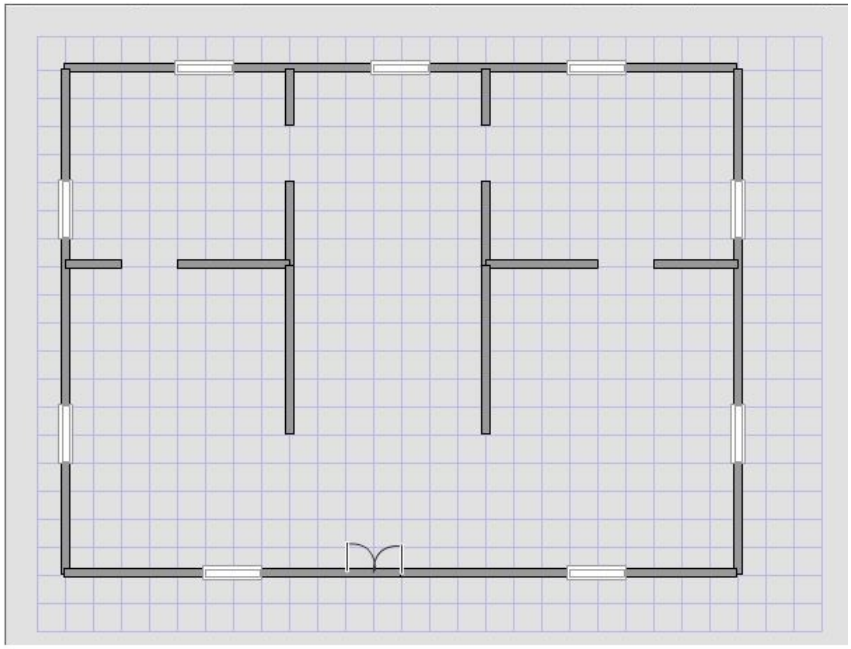

Figura 3.3: Pianta della sala di un museo

ovvio.

Se infatti volessimo portare avanti lo studio del caso in esame, ci accorgeremmo di ciò semplicemente fissando i target che andrebbero difesi, ovvero porte e finestre, e noteremmo immediatamente che quanto appena afffermato corrisponde a realtà. Il risultato è rappresentato in figura 3.2.

Proviamo quindi ad inoltrarci in qualcosa di più complesso. Immaginiamo un ambiente più ampio e all'interno del quale un pattugliatore possa muoversi con più libertà; la possibilità di seguire più strade incrementa notevolmente la difficoltà nel momento in cui si dovrà cercare il cammino minimo e rende necessario l'utilizzo di un metodo di calcolo di tale percorso. Mostreremo in questo modo anche l'utilità del software Concorde, che ci permetterà dicalcolare il cammino migliore senza alcuno sforzo.

Ipotizziamo che l'ambiente su cui effettuare il patrolling sia il grande salone di un museo rappresentato in figura 3.3, e che i target siano le varie opere d'arte esposte. Questa situazione è ben diversa dalla precedente perchè gli obiettivi sono

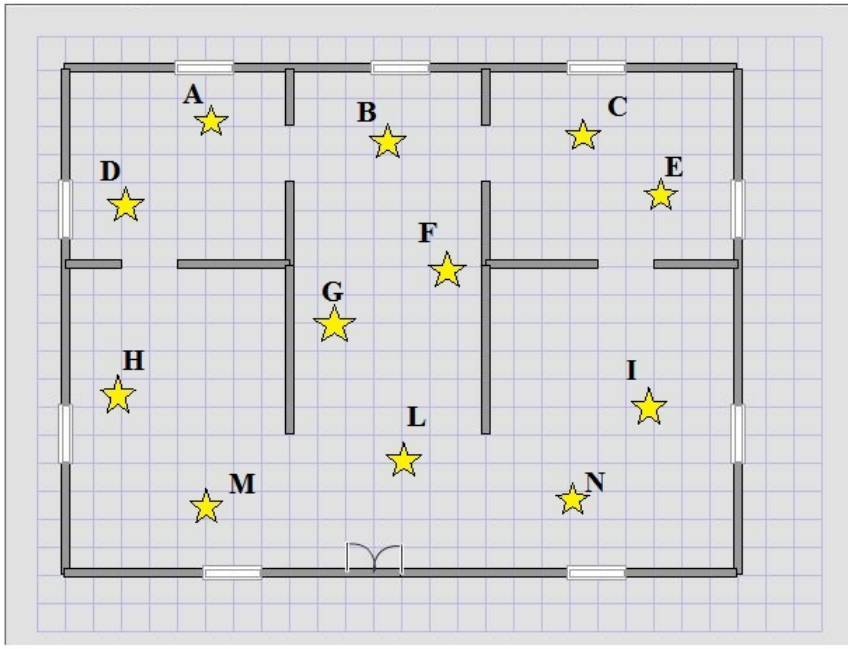

Figura 3.4: Identificazione target nel salone del museo

'sparsi' anche all'interno della sala e non sono necessariamente sul perimetro della stanza, come invece avveniva per le porte e le finestre dello scenario precedente. Inoltre in questo caso il robot è molto più libero di muoversi, in quanto l'ambiente risulta meno partizionato e non vi è una forte suddivisione in vere e proprie stanze. Si faccia riferimento alla figura 3.4 per una rappresentazione dello scenario.

Una volta definito l'ambiente e i target all'interno di esso passiamo alla ricerca del cammino ottimo che dovremo seguire. Questa fase risulta molto semplice grazie alla possibilità di utilizzare il software Concorde, del quale daremo ura una breve descrizione.

3.1.2 Concorde

Come accennato nei capitoli precedenti, Concorde è il software che utilizzeremo per calcolare il cammino chiuso che sia una soluzione 'quasi ottima' al Problema del Commesso Viaggiatore (TSP).

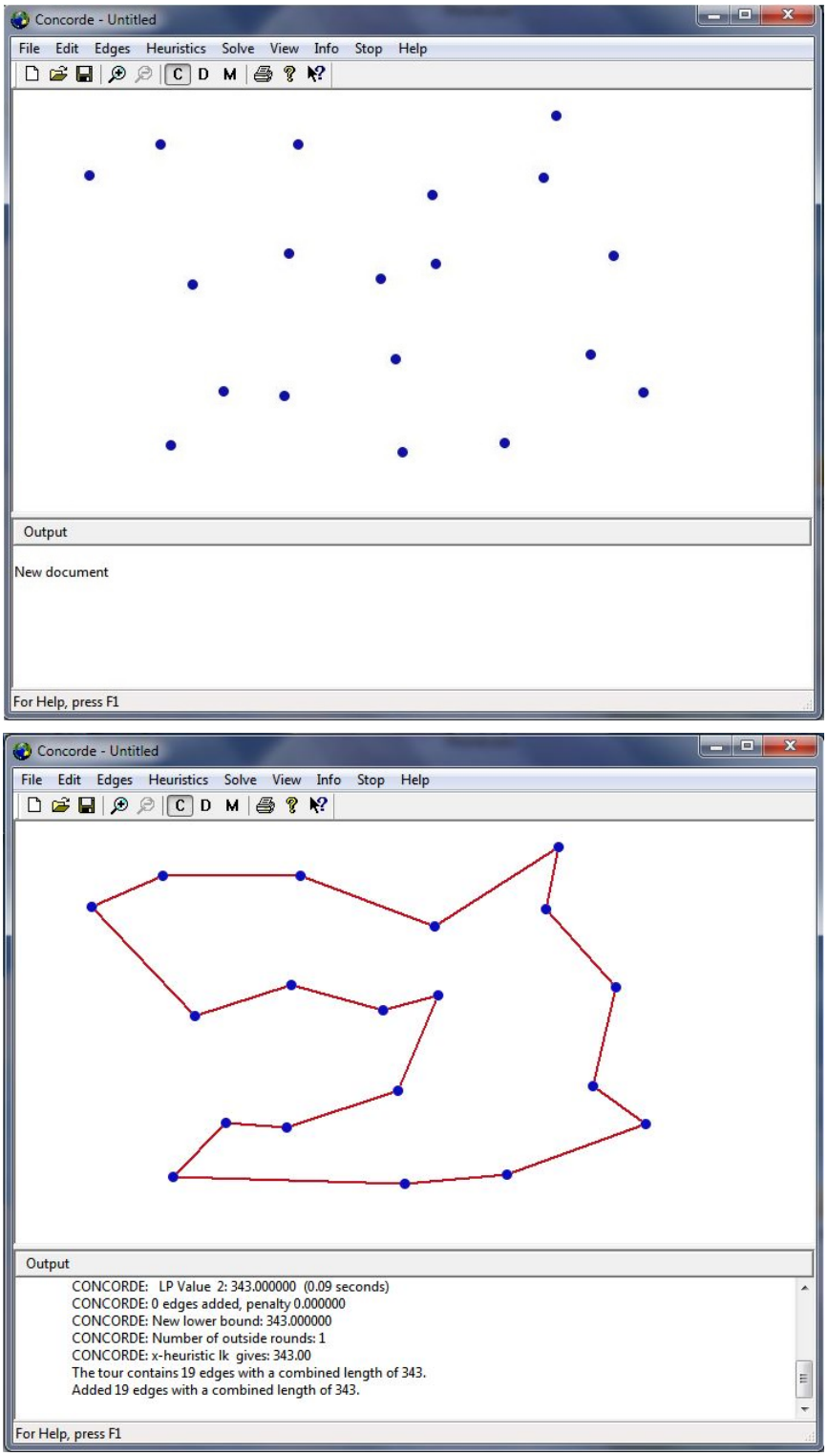

Figura 3.5: Interfaccia del software Concorde

Il software è scritto in linguaggio ANSI C, ed è stato realizzato da alcuni docenti della Facoltà di Matematica della University of Waterloo; è stato utilizzato per generare la soluzione ottima a tutti i 110 casi della TSPLIB [11], una libreria che contiene istanze d'esempio del TSP - perlopiù mappe di città e schemi di circuiti stampati - il più grande dei quali presenta un totale di 85900 nodi.

L'interfaccia che si presenta all'utente è composta da due display: uno è utilizzato per mostrare il grafico, l'altro invece stampa a video informazioni riguardo al grafico e allo stato d'esecuzione dell'algoritmo. Il software è in grado di leggere grafici da file salvati in diversi formati, quindi modificarli e infine salvarli nel formato più opportuno; esso da anche la possibilità di aggiungere, posizionare e rimuovere i vari vertici del grafico. Per un riferimento visuale si veda la figura 3.5.

Concorde permette di risolvere il TSP del grafo che si sta studiando utilizzando diversi algoritmi euristici, come ad esempio quello di Lin Kernighan, quello del Nearest Neighbour o quello Greedy; inoltre, Concorde permette di generare degli archi all'interno di un set di nodi secondo diversi algoritmi, come quello della disuguaglianza triangolare o del minimo albero di copertura.

In maniera del tutto inspiegabile, il software presenta un grosso limite: esso non da alcuna possibilità di verificare che la distanza dei nodi che si creano all'interno della GUI sia quella desiderata. Non vi è una griglia e nemmeno la possibilità né di visualizzare né tantomeno di settare il peso degli archi. Questo limite è aggirabile avendo l'accortezza e l'intuito di creare una nuova mappa con un semplice editor di testo e poi salvare questo file con estensione .tsp, che quindi potrà essere aperto da Concorde. La comodità risulta evidente quando si nota che, realizzando la mappa in formato testo, vi è la possibilità di inserire quelle che sono le coordinate - paragonabili a quelle cartesiane - di ogni nodo e realizzare così il grafo desiderato in maniera assolutamente precisa.

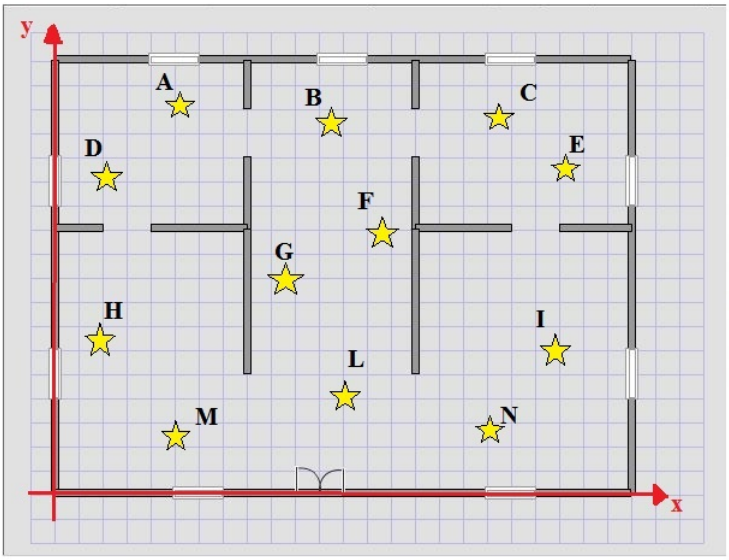

Figura 3.6: Coordinate che identificano i target

3.1.3 Posizionamento dei target

Ora che abbiamo preso atto del funzionamento e dell'utilità di Concorde, non ci resta che sfruttarlo. Come detto nella sezione precedente, esso non consente di fissare i nodi in maniera precisa, o comunque non fornisce alcuno strumento (come righelli o griglie) che consenta all'utente di posizionare tali nodi impostandone le reciproche distanze.

L'unico modo possibile è quello di modificare con un editor di testo un file .tsp che rappresenta l'estensione delle mappe che è in grado di leggere Concorde, come già detto.

Serve quindi costruire un sistema di coordinate all'interno del quale posizionare la piantina del nostro museo e ottenere così le posizioni relative rispetto all'origine dei nostri target. Torniamo quindi a lavorare con la piantina vista in precedenza e su di essa costruiamo un sistema di assi cartesiani e sfruttiamolo per calcolare le coordinate dei vari target, come fatto in figura 3.6.

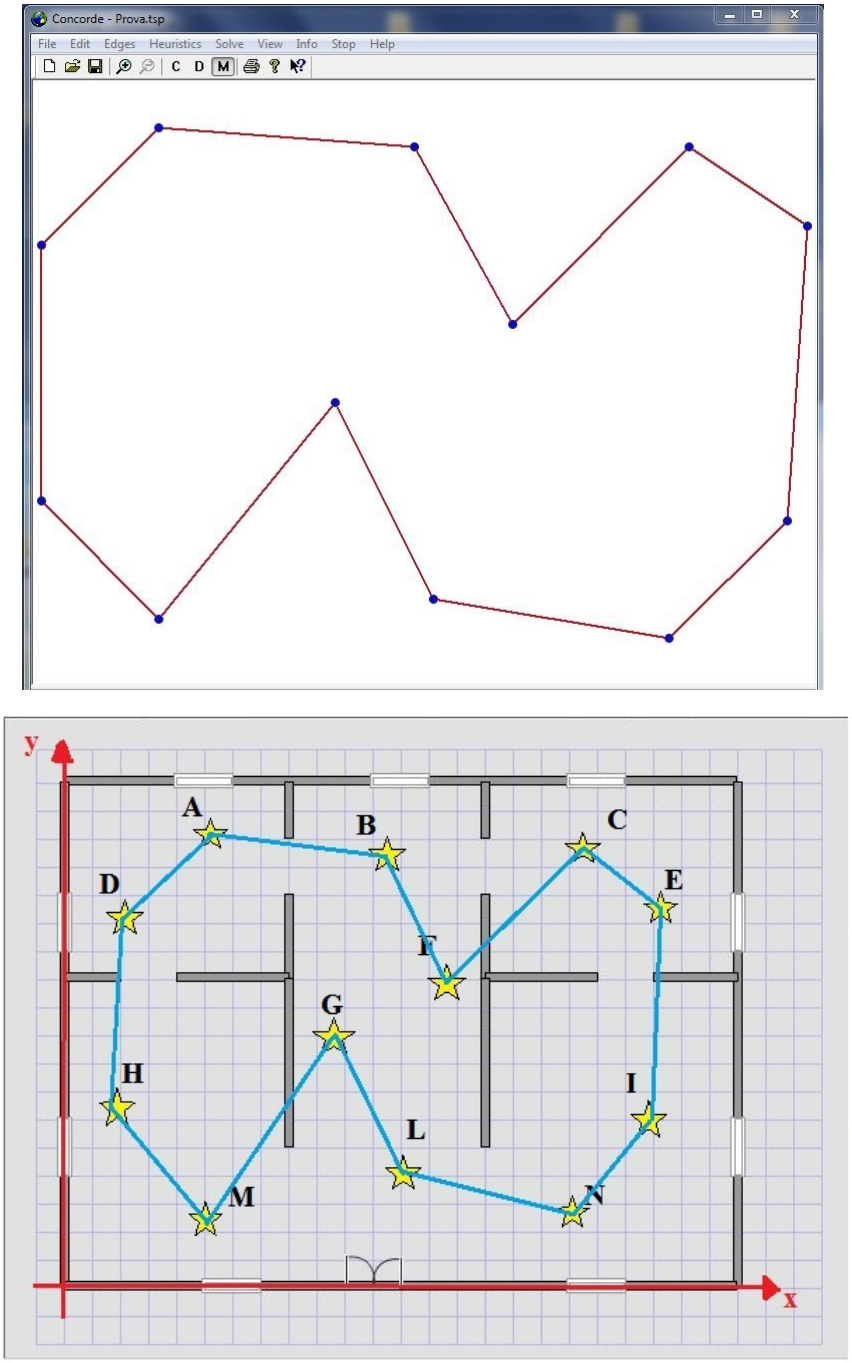

Figura 3.7: Percorso elaborato da Concorde e riportato sulla mappa

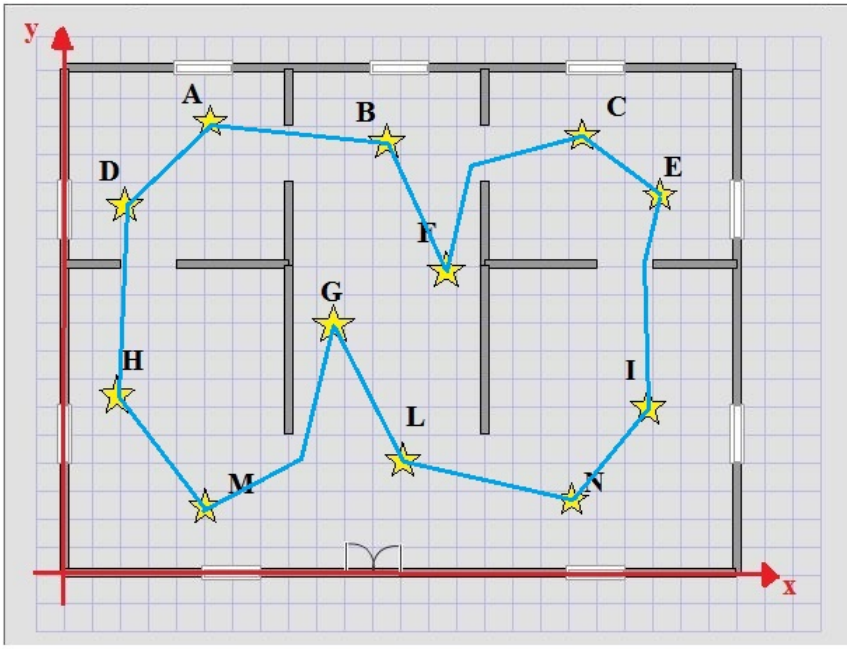

Figura 3.8: Percorso adattato alle caratteristiche dell'ambiente

3.1.4 Calcolo percorso

Arrivati a questo punto non resta che creare un file .tsp che riporti le coordinate appena calcolate. Una volta specificate tali coordinate ed il numero di nodi, salveremo il file e lo apriremo con Concorde. Ci troveremo di fronte l'insieme dei nodi esattamente come erano posizionati all'interno della mappa, e risolvendo tale configurazione otteremo il percorso ottimo rappresentato in figura 3.7, dove è mostrato anche lo stesso percorso riportato sulla mappa.

Si noti come il risultato ottenuto sia un percorso che però non tiene conto dei limiti imposti dalle pareti dell'ambiente, in quanto due di queste vengono letteralmente attraversate dal percorso ideale. Ovviamente questa situazione va risolta e per farlo sarà sufficiente adattare il percorso ottenuto a quelli che sono i vincoli imposti dall'ambiente, come mostrato in figura 3.8; infatti seppur questo nuovo percorso non sia più quello ideale ottenuto inizialmente, va comunque fatto notare come esso ne sia una buona approssimazione.

3.1.5 Calcolo dell'idleness

Proseguiamo ora con il calcolo dell'idleness, ovvero la durata dell'intervallo che separa due passaggi successivi del patroller per lo stesso nodo. Come già accennato nei capitoli precedenti, in un percorso chiuso del tipo che stiamo considerando, ovvero in cui ogni singolo nodo è visitato una ed una sola volta, la lunghezza totale del percorso sarà pari alla somma delle lunghezze dei singoli archi. Partendo dalla rappresentazione sulla mappa e facendo una stima delle lunghezze degli archi arriviamo ad ottenere una lunghezza totale del percorso pari a 134 metri.

Ovviamente, elemento imprescindibile per il calcolo dell'idleness è senza dubbio il robot; al variare delle sue caratteristiche cambieranno anche le sue prestazioni e quindi il valore di idleness che caratterizzerà il percorso. Non avendo a che fare al momento con un robot in maniera concreta, faremo delle stime sulla base delle quali proseguiremo l'anali del caso di studio che stiamo affrontando.

Supponiamo di avere a nostra disposizione un robot i cui motori sono in grado di raggiungere velocità pari a 200 rpm (*giri al minuto*); supponiamo inoltre che le ruote che usa per spostarsi abbiano un diametro di 20 centimetri.

$$200 \text{ rpm} \Longrightarrow 200 \times (2\pi \times 0.1) \simeq 125.7 \text{ metri al minuto}$$

Considerando che il moto non è perfettamente uniforme e che il pattugliatore potrebbe perdere del tempo per verificare che effettivamente un target si trovi al sicuro, possiamo scendere ad una stima di 100 metri al minuto (6 Km/h). In base a quest'ultimo valore stimato possiamo affermare che per percorrere interamente il percorso il patroller, ogni singolo giro, impiegherà

$$134 \div 100 = 1.34 \text{ minuti} = 80.4 \simeq 81 \text{ secondi}$$

Ecco quindi il risultato che cercavamo, ovvero il valore della nostra idleness: 81 secondi. Teniamo sempre presente che tale risultato è stato raggiunto facendo solo delle stime per quanto riguarda sia l'ambiente che il robot; in ogni caso,

avendo a disposizione i dati effettivi dell'uno e dell'altro, tale valore potrebbe essere assolutamente preciso.

Si noti anche come questa stima sia stata effettuata 'per difetto', tenendo conto di eventuali perdite di tempo che il robot puo subire durante l'operazione di patrolling. Se non avessimo effettuato tali considerazioni e avessimo mantenuto il valore ottenuto inizialmente (ovvero 125.7 metri al minuto, poi abbassato a 100) avremmo ora a che fare con un valore di idleness pari a 64 secondi; portandolo ad 81 secondi abbiamo effettuato un incremento di 17 secondi, andando così a tollerare una diminuzione delle prestazioni pari al 26.6% del valore inizialmente stimato.

3.1.6 Considerazioni finali

Il lavoro di analisi può ormai considerarsi concluso. Ora non resta che confrontare il valore dell'idleness con il tempo di penetrazione. Al nostro stato attuale, riguardo a quest'ultimo non posssiamo dire nulla con certezza; anche in questo caso siamo costretti a basarci su stime.

Ipotizziamo quindi che i nostri target siano esposti all'interno di una teca: possiamo quindi far coincidere concettualmente la violazione di un target con quella della teca corrispondente. Immaginiamo che la società produttrice delle teche usate per difendere i nostri target ci comunichi che esse siano realizzate in modo tale che una loro apertura, rottura o violazione di qualsiasi altro tipo non possa avvenire entro un tempo di 30 secondi.

Questo ci permette agilmente di prendere una decisione in merito al numero di patroller da utilizzare: utilizzare un solo pattugliatore significa transitare per ogni target una volta ogni 81 secondi, lasciando quindi tutto il tempo ad un ipotetico ladro di arrivare al suo obiettivo. Introducendo un secondo patroller, il cui punto di partenza sia esattamente all'opposto di quello del primo, otterremo come risultato un raddoppiamento della frequenza con cui tali pattugliatori controllano lo stato

dei propri target: nonostante ciò, avremmo comunque un valore di idleness ancora troppo alto (40.5 secondi) per impedire ad un ladro di violare una delle teche. Aggiungendo invece un terzo pattugliatore arriveremo ad ottenere il risultato da noi cercato: infatti in questo modo il valore dell'idleness diventerà un terzo del valore iniziale, ovvero 27 secondi, impendendo così che un ladro possa avere a sua disposizione i 30 secondi di cui necessita. La violazione di un target da parte di un intruso non potrà avvenire con certezza.

Da un punto di vista concettuale, abbiamo definito una strategia di patrolling e la abbiamo applicata ad uno scenario. Arrivati a questo punto possiamo però affermare che la bontà di questa strategia dipende anche dai mezzi a disposizione da chi decidesse di adottarla. Come già detto, un robot più prestante potrebbe facilmente ridurre drasticamente l'idleness; o ancora, l'avere a disposizione strumenti avanzati per quel che concerne l'elaborazione delle immagini potrebbe permettere una più rapida individuazione di anomalie dell'ambiente, e ciò si tradurrebbe ancora una volta in un abbassamento dell'idleness.

Vediamo ora come, con i mezzi a nostra disposizione, potremmo realizzare un patroller che sia in grado di adottare la nostra strategia per difendere determinati target. Tenteremo di realizzare un pattugliatore mendiante Lego Mindstorms, quindi dopo una rapida panoramica su cosa sia e cosa metta a nostra disposizione la linea Mindstorms, vedremo come potremo sfruttare ciò per realizzare un patroller.

3.2 LEGO Mindstorms

Lego Mindstorms è una linea di prodotti LEGO realizzata per costruire robot più o meno complessi, combinando tra loro diversi tipi dei mattoncini del famoso produttore di giocattoli danese. Pur potendo apparire come un semplice passatempo, Lego Mindstorms si è affermato sempre più sul panorama mondiale come uno dei

principali strumenti per la realizzazione di robot, soprattutto in ambito didattico, per la semplicità e la duttilità che lo caratterizzano.

L'ultima evoluzione di Mindstorms è il robot Lego Mindstorms NXT, sviluppato e distribuito da Lego nel 2006; il componente principale del kit è il mattoncino intelligente NXT Brick, capace di ricevere input da quattro sensori e controllare fino a tre motori elettrici.

Figura 3.9: Mattoncino NXT

3.2.1 Specifiche tecniche [10]

- microprocessore centrale a 32-bit ARM7 - 256 KB memoria flash, 64 KB RAM;
- microcontroller a 8-bit AVR- 4 KB memoria flash, 512 Byte RAM;
- Comunicazione Bluetooth - V 2.0;
- Una sola porta USB 1.1 a piena velocità (12 Mbit/s);
- 4 porte di input, piattaforma digitale a 6 fili;
- 3 porte di output, piattaforma digitale a 6 fili;
- Display LCD monocromatico da 100×64 pixel;

- Altoparlanti a 8 KHz - risoluzione 8-b, sample rate 2-16 KHz;
- Potenza: 6 batterie di tipo AA.

3.2.2 Sensori

Insieme al mattoncino NXT viene fornita una serie di sensori di diverso genere, che possono essere utilizzati e combinati in diversi modi così da realizzare i più svariati tipi di robot. Vediamo una rapida panoramica dei sensori a disposizione.

Sensore ad ultrasuoni

Il sensore ad ultrasuoni è, insieme a quello della luminosità e a quello del colore, ciò che permette al nostro robot di vedere. Esso infatti permette di identificare ed evitare oggetti, nonché di percepire le distanze e i movimenti. Il principio di funzionamento è quello classico di quando si parla di ultrasuoni, lo stesso che usano le navi per misurare la profondità del fondale marino: viene emesso un impulso ad ultrasuoni, che rimbalza sull'ostacolo e torna indietro per poi essere raccolto dal sensore, che in base al tempo trascorso è in grado di calcolare la distanza tra sé e l'oggetto. Questo sensore è in grado di misurare distanze fino a 2.5 metri con una incertezza di $\pm$ 3 cm.

Figura 3.10: Sensore ad ultrasuoni

Sensore di colori

Il sensore di colori è in grado di distinguere 6 colori differenti, ed è inoltre in grado di coglierne le sfumature al variare delle condizioni di luce; esso può inoltre essere utilizzato come una lampada in grado di emettere luce colorata. Può essere sfruttato per separare oggetti di colore differente oppure per seguire una linea colorata sul pavimento.

Sensore di luce

Il sensore di luce, invece, fornisce un valore proporzionale alla luminosità che esso rileva nell'ambiente, nonché di distinguere le varie tonalità di grigio.

Figura 3.11: Sensore di luce

Sensore di contatto

Il sensore di contatto è invece ciò che fornisce il nostro robot del senso del tatto. Come è facilmente intuibile, esso reagisce alla pressione ed al rilascio del pulsante; anche i tasti del NXT possono essere settati in modo da funzionare come sensori di contatto.

Sensore di suoni

Il sensore di suoni può percepire i rumori dell'ambiente circostante secondo due scale: la scala *decibel* e quella *adjusted decibels*; in quest'ultima categoria sono presenti solo i suoni che sono udibili dall'uomo, mentre utilizzando la scala decibel classica vengono captati anche i suoni troppo bassi o troppo alti per l'orecchio umano.

Figura 3.12: Sensore di suoni

Altri sensori

Infine abbiamo altri sensori, che evitiamo di elencare, i quali permettono di fornire altre informazioni come ad esempio la temperatura dell'ambiente, la posizione in gradi rispetto al nord oppure l'accelerazione lungo un asse prestabilito. O ancora, altri sensori che consentono di rilevare fonti di segnali ad infrarossi oppure di controllare altri dispositivi, sempre tramite IR.

3.2.3 Servomotori

I servomotori forniti all'interno del kit Lego Mindstorms consentono, ovviamente, al nostro robot di muoversi. Essi funzionano in corrente continua con una tensione di 9V e possono raggiungere velocità fino a 170 rpm.

La caratteristica principale di questi servomotori è che ognuno di essi ha al suo interno un sensore di rotazione che permette di misurare (e settare) sia la distanza percorsa che la velocità. Il sensore può misurare la rotazione in gradi (con una precisione di $\pm$ 1 grado) oppure in rotazioni complete, in modo che una rotazione completa corrisponda a 360 gradi.

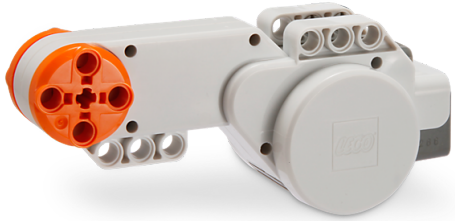

Figura 3.13: Servomotore

3.2.4 Programmazione

Alcuni programmi molto semplici possono essere scritti usando il menu del'NXT, mentre programmi più complicati possono essere scaricati sul 'mattoncino' utilizzando il Bluetooth oppure semplicemente tramite cavetto USB. Tali programmi possono essere realizzati mediante diversi linguaggi di programmazione:

- **NXT-G**: è il software di programmazione che si trova incluso nel kit Lego Mindstorms NXT. Esso fornisce un linguaggio di programmazione visuale per scrivere semplici programmi che potranno poi essere scaricati sull'NXT. Il software mette a disposizione un ambiente grafico molto intuitivo e facile da utilizzare di tipo drag and drop.

- **BricxCC**: è il software che utilizzeremo per programmare il nostro Lego Mindstorms. Consente di realizzare programmi per l'NXT utilizzando sia il linguaggio NBC che quello NXC, permettendone anche la compilazione. Fornisce diversi tool piuttosto utili, tra i quali il controllo in remoto e la possibilità di verificare direttamente lo stato di sensori ed attuatori.

- **NBC**: il Next Byte Codes è un linguaggio di programmazione open source di tipologia assembly, il quale può essere usato per programmare il mattone NXT. Il compilatore NBC traduce il sorgente nel byte-code del NXT, che può cos' essere eseguito direttamente sul brick NXT.

- **NXC**: il Not eXactly C è un linguaggio open source la cui sintassi, come si può intuire, è simile a quella del linguaggio C; Il compilatore NXC non fa altro che tradurre il codice scritto in NXC in formato NBC, per poi richiamare il compilatore NBC.

- **RobotC**: anche questo potente linguaggio di programmazione è basato sul C, ed è molto diffuso nella programmazione di robot. Permette portabilità del codice tra più robot, ed è un linguaggio essenzialmente text-based, contrapponendosi in questo modo all'NXT-G di Lego Minstorms.

- **leJOS NXJ**: anch'esso è un linguaggio di programmazione molto potente ed open source, basato però sul linguaggio di programmazione Java. Esso utilizza un firmware che non è altro che una Java Virtual Machine 'alleggerita' che è stata caricata sul brick NXT. Esso quindi offre una programmazione Object Oriented con tutte le caratteristiche e i tipi supportati da Java.

- **MATLAB**: è un linguaggio di alto livello per il calcolo numerico ed analisi di dati. Può essere utilizzato per controllare i robot Lego Mindstorms attraverso una porta Bluetooth seriale. Basato su MATLAB è anche Simulink, un ambiente di controllo utilizzato per modellare e simulare degli ambienti dinamici. Tramite questo ambiente si possono progettare algoritmi di controlli di cui si può ricavare il relativo codice in linguaggio C, il quale può in seguito essere compilato e caricato sull'NXT.

3.3 Realizzazione del patroller

Dopo questa breve panoramica sul mondo Lego Mindstorms, possiamo quindi procedere con la realizzazione di quello che sarà il nostro patroller.

Ribadiamo ancora una volta che l'obiettivo principale di questa tesi è quello di definire una buona strategia di patrolling che possa essere adottabile nel momento

in cui qualcuno avesse i giusti mezzi per metterla in pratica. Come abbiamo visto, l'analisi da noi realizzata si basa su considerezioni che derivano da importanti studi sull'argomento e, nonostante alcune forti assunzioni, tale analisi pare soddisfare le principali caratteristiche che deve possedere una strategia di patrolling; anzi, in linea teorica tale strategia potrebbe garantire una rilevazione con una probabilità pari ad 1 (evento certo), anche se da un punto di vista concreto questo non sarà probabilmente possibile visto che difficilmente tutte le condizioni necessarie affinchè si realizzi uno scenario ideale è difficile si verifichino.

Un obiettivo secondario è comunque quello di realizzare un patroller che sia in grado di mettere in pratica tale strategia, perciò andremo ora a percorrere questa strada. Per fare questo saremo limitati da quelli che sarenno gli strumenti a nostra disposizione, ma tenteremo comunque di realizzare una buona struttura di base che, in un ipotetico futuro, potrebbe essere riutilizzata e migliorata.

3.3.1 Assemblaggio

La realizzazione del modello è avvenuta seguendo il piu classico degl ischemi adottatti tra i vari progetti Lego Mindstorms: una struttura formata da due ruote motrici anteriori collegate a due motori separati e una ruota posteriore passiva, in grado di moversi in maniera da assecondare il movimento dettato dalle ruote motrici. In particolare, la ruota posteriore non è una semplice ruota, bensì si è deciso di utilizzare una particolare ruota chiamata Rotacaster, che permette spostamenti in tutte le direzioni senza opporre resistenza grazie ad una serie di cuscinetti rotanti che sono in grado di assecondare lo spostamento nella direzione verso la quale si muovono le ruote motrici.

Figura 3.14: Ruote Rotacaster

Per quel che riguarda la struttura essa è stata sviluppata in verticale facendo attenzione a caricare la maggior parte del peso sulle ruote anteriori. Infatti le ruote anteriori sono quelle motrici e caricarle di peso significa aumentare la loro pressione sul terreno e quindi aumentarne l'attrito, garantendo migliori performace; inoltre caricare, al contrario, la ruota posteriore avrebbe significato appesantire una ruota che in un certo senso deve essere 'trascinata' e ciò avrebbe comportato un più difficile spostamento del veicolo nella sua totalità.

L'intelaiatura è stata realizzata in maniera piuttosto solida, con una postazione appositamente creata per fare spazio al mattone NXT, che può venire ulteriormente fissato grazie a due perni che lo legano al resto della struttura. Dall'NXT si diramano i cavi che permettono di collegarlo ai sensori ed agli attuatori; come si può vedere dalla figura 3.15 abiamo utilizzato un sensore ad ultrasuoni e due sensori di luce. Per quel che riguarda gli attuatori, invece, oltre ai due motori, abbiamo una semplice luce collegata ad una delle uscite del blocco NXT.

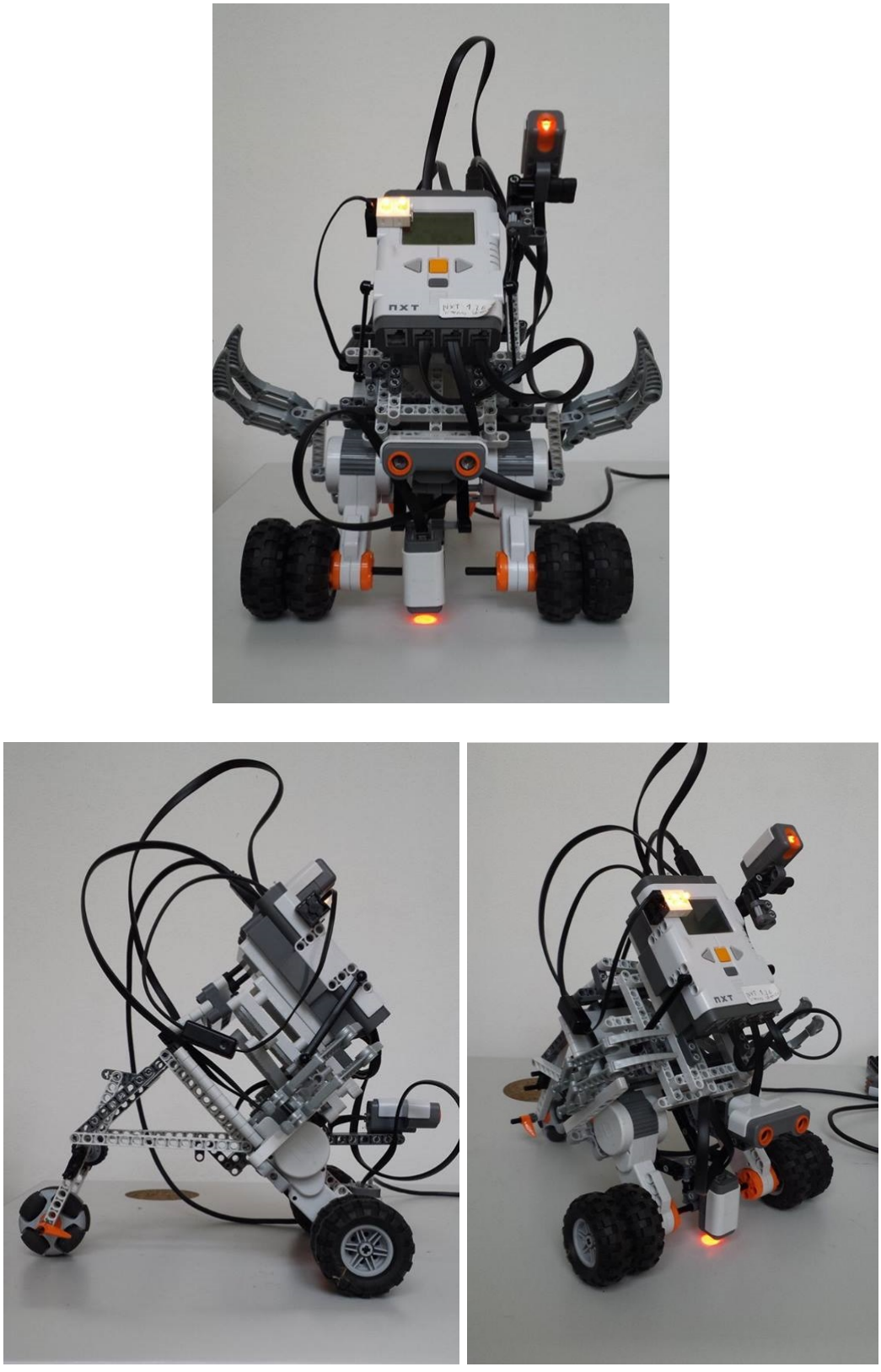

Figura 3.15: Struttura del patroller

3.3.2 Comportamento e funzionamento

Una volta assemblato il nostro robot, non ci resta che definire il suo comportamento. Ovviamente le decisioni riguardanti come esso debba comportarsi sono già state prese in precedenza, perchè conseguentemente ad esse abbiamo deciso di utilizzare i sensori di cui si è detto nella sezione precedente.

Ricordiamo ancora una volta che un patroller deve poter essere in grado di seguire un percorso prestabilito e identificare eventuali intrusioni. Con gli strumenti che abbiamo a nostra disposizione cercheremo ora di ricondurci ad una situazione in cui faremo in modo che il nostro patroller sia in grado di svolgere il proprio compito al meglio delle sue possibilità.

Spostamento

Per poter adottare la strategia da noi definita è senza dubbio necessario che il nostro patroller possa seguire un percorso già precedentemente stabilito. Se infatti in alcuni casi si adottano strategie che prevedono uno spostamento casuale o pseudocasuale all'interno dell'ambiente è invece per noi fondamentale che il pattugliatore sappia già quale percorso deve seguire.

Sotto questo punto di vista la situazione è risolvibile piuttosto agilmente, sebbene le componenti per Lego Mindstorms a nostra disposizione siano semplicemente quelle base. Nel nostro stato attuale non sappiamo far sì che il patroller 'apprenda' un percorso che esso dovrà seguire, ma siamo comunque in grado di fare in modo che esso segua un determinato tragitto. Questo è reso possibile grazie al sensore luminoso posizionato nella parte inferiore del robot, esattamente tra le due ruote motrici. Come detto in precedenza, tale sensore fornisce valori differenti al variare della luminosità nell'ambiente.

Quello che non abbiamo detto è che esso è in grado anche di misurare la luce riflessa. Esso infatto può essere regolato in modo che emetta egli stesso luce, che può essere riflessa da una superficie e nuovamente catturata dal sensore stesso, il quale

potrà così misurare quanta ne è stata riflessa. Questo risulta essere particolarmente utile quando si vuole che il robot segua una linea sul pavimento. Infatti, presa una linea scura su una superficie bianca, essa rifletterà un quantitativo di luce minore rispetto al resto della superficie. Come vedremo in seguito nella sezione dedicata alla descrizione del codice, questo aspetto sarà sfruttato proprio per far sì che il nostro patroller sia in grado di seguire una linea sul pavimento da noi precedentemente tracciata.

Così il primo problema, ovvero quello dello spostamento, è risolto; una volta definito quello che sarà il nostro percorso tramite il software Concorde, non ci resterà che tracciarlo sul terreno e lasciare che il patroller segua tale percorso. La soluzione non è sicuramente la più performate o comunque la preferibile in assoluto, ma è ciò che siamo in grado di realizzare sfruttando al meglio i mezzi a nostra disposizione.

Evitare gli ostacoli

Una caratteristica che dovrà possedere il nostro patroller è quella di saper evitare gli ostacoli; caratteristica non strettamente necessaria ma comunque alquanto utile. Infatti, immaginando per esempio di essere all'interno del nostro caso di studio, il saper evitare un impedimento non dovrebbe essere necessrio in quanto una volta definito il percorso di pattugliamento esso non dovrebbe presentare alcun ostacolo.

Ad ogni modo, in un contesto più generale, possiamo affermare che a causa di un qualsiasi fattore esterno potrebbe finire proprio sul nostro tracciato un qualcosa che potrebbe considerarsi un ostacolo per il nostro patroller; risulta quindi utile permettere al nostro robot di riconoscere e risolvere situazioni di questo tipo.

Dunque ecco il perchè del sensore ad ultrasuoni. Come detto esso permette di individuare la presenza di un oggetto e di fornire in maniera piuttosto precisa il valore della distanza alla quale esso si trova. Fornendo il nostro robot di un sensore di questo tipo, rivolto ovviamente verso la direzione dello spostamento del

patroller, siamo in questo modo in grado di impedire che il nostro pattugliatore vada a collidere con eventuali ostacoli situati sul suo percorso.

Ovviamente la sola individuazione degli ostacoli non è sufficiente; è necessario anche aggirarli e per questo è steto realizzato un semplice algoritmo, che descriveremo in seguito, grazie al quale il patroller può aggirare l'ostacolo semplicemente cambiando la sua traiettoria una volta che questo è stato individuato. Giunto in prossimità dell'impedimento, il patroller lo supererà passandovi accanto e in seguito si riposizionerà sulla linea del proprio tragitto, proseguendo l'operazione di patrolling.

Ritrovare il percorso

Durante la fase di spostamento può succedere, per diversi motivi, che il nostro patroller perda la propria posizione e venga a trovarsi al di fuori del proprio tracciato e che quindi perda la linea che stava seguendo. Ovviamente in situazioni come questa il robot deve saper ritrovare la giusta via per poter riprendere la sua operazione di pattugliamento, altrimenti la sicurezza dell'intero ambiente che esso sta pattugliando potrebbe essere messa a rischio.

Il comportamento che terrà il nostro patroller in una situazione del genere è molto semplice: nel momento in cui esso finirà al di fuori del percorso segnato a terra esso inizierà immediatamente a girare su sè stesso, alternado una rotazione a destra con una a sinistra, inizialmente ruotando di piccoli angoli via via sempre maggiori fino al momento in cui non ritroverà nuovamente la linea nera in terra.

Nel momento in cui non dovesse riuscire a ritrovare il percorso esso entrerà in uno stato in cui inizierà a cercarlo intorno a sè; infatti quando il processo di ricerca descritto precedentemente non dovesse fornire alcun risultato, il patroller inizierà a muoversi a spirale, creando un movimento sempre più ampio e andando a cercare un qualsiasi punto della linea dal quale ricominciare il lavoro di patrolling.

Per quanto questa strategia possa sembrare banale e limitativa, in realtà essa è

semplice e si adatta perfettamente alla nostra situazione. Infatti si potrebbe affermare che tale comportamento potrebbe far sì che il robot recuperi la sua posizione sul tracciato in un punto ben distante da dove il patrolling era stato interrotto, rischiando così di saltare il controllo di qualche target o di una porzione di perimetro. Questo è sì possibile, ma in realtà la conseguenza più grave sarebbe quella di dover appunto non controllare alcuni obiettivi per la durata di un giro; infatti essendo l'operazione di patrolling ciclica il controllo di tali target avverrebbe regolarmente al passaggio successivo. Inoltre va anche detto che se per qualunque motivo il robot dovesse finire all'esterno del tracciato, esso probabilmente non dovrebbe finire troppo distante dallo stesso, e conseguentemente a ciò una ricerca a spirale dovrebbe far sì che il patroller possa essere in grado di riprendere l'operazione di pattugliamento in un punto non troppo distante da quello in cui il patrolling era stato interrotto.

Rilevazione di una intrusione

Ultima delle caratteristiche che il nostro patroller dovrà possedere è il saper riconoscere quelle che noi abbiamo definito intrusioni. Questo aspetto non è semplice da modellare, soprattutto con gli strumenti a nostra disposizione; esistono infatti attrezzature - ad esempio videocamere - rilasciate appositamente per la linea Lego Mindstorms, come per le ruote Rotacaster, che ci permetterebbero di arrivare ad adottare una strategia molto più realistica. Purtroppo abbiamo a nostra disposizione solo i sensori di contatto, ad ultrasuoni, di luce e di suono, in particolare gli ultimi due potrebbero essere sfruttati per realizzare ciò di cui abbiamo bisogno.

Ad esempio, un patroller si suppone lavorari in un ambiente in cui sono presenti pochi rumori: ne è un esempio lo scenario che abbiamo descritto in precedenza, ovvero quello del museo, in cui un ipotetico patroller pattuglierebbe la sala durante le ore notturne e quindi in un totale silenzio. L'unico rumore udibile dovrebbe essere quello da lui stesso emesso, e qualsiasi altro rumore captato dovrebbe quindi

essere riconosciuto come una intrusione. Nella nostra simulazione realizzata in laboratorio abbiamo invece deciso di utilizzare il sensore di luce per identificare una eventuale violazione; abbiamo infatti deciso di simulare una intrusione con una fonte luminosa (una lampada) che verrà accesa nel momento in cui dovrebbe iniziare l'intrusione e rimarrà accesa per un tempo pari al tempo di penetrazione. Questo tipo di strategia permette di ricreare alla perfezione il meccanimo di

$$\textit{tentativo di penetrazione} \longrightarrow \textit{rilevamento della penetrazione}$$

che dovrebbe verificarsi in una situazione reale. Durante la fase di patrolling il nostro robot sarà quindi in grado di riconoscere questa intrusione e reagirà ad essa emettendo un suono ed accendendo una spia posizionata sulla sua parte superiore.

Ovviamente questa strategia potrà essere utilizzata solamente in una simulazione e non sarebbe in alcun modo utile nella maggior parte degli scenari in cui sia richiesto di realizzare del patrolling; ad ogni modo ribadiamo il concetto secondo il quale in questa simulazione si vuole sfruttare ciò che abbiamo a nostra disposizione. Chiaramente, avendo strumenti migliori, potremmo realizzare una simulazione che possa rispecchiare meglio la realtà, e realizzare un patroller che possa essere utilizzato anche per realizzare del vero e proprio pattugliamento; basterebbe disporre di strumenti più avanzati, come ad esempio un sensore di calore in grado di riconoscere il calore emesso dal corpo umano, o ancora una telecamera che possa raccogliere delle informazioni rielaborate da un programma di elaborazione delle immagini per verificare che l'ambiente non presenti delle irregolarità.

3.3.3 Il codice

In questa sezione descriveremo le parti più importanti del codice che abbiamo scritto, il quale verrà mostrato per intero nell'appendice al termine della tesi. Sostanzialmente le parti sulle quali vale la pena concentrarsi sono 4: le parti che descrivono come il robot possa seguire il percorso in terra e come riesca a recupe-

rarlo una volta perso, la parte che permette di evitare gli ostacoli e infine la parte che consente al patroller di individuare una intrusione.

```
1  sub findLineSpiral(){
2       int curve=60;
3       until(SENSOR_3 < LightThreshold ||
4             SensorUS(IN_4) < Near ) {
5            OnFwdSync(OUT_BC, Speed, curve);
6            Wait(200);
7            if(curve > 20){
8                 curve -= 1;
9            }
10      }
11 }
12
13 sub findLine(){
14      bool lineFound = false;
15      bool turnLeft;
16      if(Random(2) > 0){
17           turnLeft = true;
18      }else{
19           turnLeft = false;
20      }
21      long rotateTimeout = 10;
22      long rotateStartTime;
23      while(lineFound == false){
24           rotateStartTime=CurrentTick();
25
26           if(turnLeft){
27                OnFwdSync(OUT_BC, Speed, -100);
28           }else{
```

```
29                OnFwdSync(OUT_BC, Speed, 100);
30          }
31
32          until(CurrentTick()-rotateStartTime >= rotateTimeout
33                 || lineFound){
34                if(SENSOR_3 < LightThreshold){
35                      lineFound = true;
36                }
37          }
38
39          if(lineFound == false){
40                turnLeft = !turnLeft;
41
42                if(rotateTimeout > 1000){
43                     findLineSpiral();
44                     lineFound = true;
45                }else{
46                     rotateTimeout += 100;
47                }
48          }
49     }
50 }
```

Questa sezione di codice descrive il comportamento del patroller nei confronti della posizione da seguire. Le due funzioni, infatti, si occupano di seguire la linea in terra e di cercarla nel momento in cui questa verrà persa. Il comportamento della subroutine *findLineSpiral* è particolarmente semplice ed intuitivo: essa fa sì che il patroller inizi a muoversi disegnando una spirale sempre più ampia finchè non riconoscerà a terra una porzione di tracciato e ricomincerà a seguirla.

La subroutine *findLine* è ben più complicata; essa si occupa di fare in modo che il patroller segua la linea in terra che è utilizzata per tracciare il suo percorso. Come già detto in precedenza, nel momento in cui il robot durante il patrolling dovesse 'accorgersi' di essere finito al di fuori della linea nera esso inizierà a ruotare a destra e a sinistra su sè stesso, finchè non ritroverà la linea in terra. Questa subroutine si occupa proprio di descrivere questo comportamento, e lo fa scegliendo innanzitutto in maniera del tutto casuale la direzione - destra o sinistra - verso la quale ruotare inizialmente; fatto ciò verrà effettivamente eseguita l'istruzione che comanda la rotazione e tale rotazione verrà controllata mediante l'utilizzo di una funzione che permette di tenere traccia dello scorrere del tempo. Passato un determinato intervallo di tempo, se la linea non è stata trovata verrà cambiato senso di rotazione, la quale durerà questa voltà pochi istanti in più, e così via finchè o non viene individuata finalmente la linea oppure finchè non si raggiunge un numero massimo di rotazioni al termine delle quali verrà appunto a verificarsi la ricerca a spirale.

```
1  task move(){
2       while(SENSOR_3 > LightThreshold){
3            findLineSpiral();
4            if (SensorUS(IN_4) < Near){
5                 avoidObstacles();
6            }
7       }
8       while(true){
9            if(SENSOR_3 > LightThreshold){
10                findLine();
11           }
12           if(SensorUS(IN_4) < Near){
13                avoidObstacles();
14           }
```

```
15            OnFwd(OUT_BC, Speed);
16      }
17  }
```

Queste due subroutine sono gestite dal task *move*, che in base ai valori restituiti dai vari sensori si occupa di attivare l'una piuttosto che l'altra. Come si può evincere dal suo nome, questo task si occupa della gestione di tutto ciò che concerne il movimento del patroller, anche dell'aggiramento degli ostacoli; infatti, come si può notare, conseguentemente all'eccessiva vicinanza di un oggetto, il task va ad eseguire la subroutine *avoidObstacles*, mostrata qui in seguito.

```
1   sub avoidObstacles(){
2       RotateMotorEx(OUT_BC, 50, Rot, 100, true, true);
3       OnFwd(OUT_BC, Speed);
4       Wait(1900);
5       RotateMotorEx(OUT_BC, 50, Rot, -100, true, true);
6       OnFwd(OUT_BC, Speed);
7       Wait(1700);
8       RotateMotorEx(OUT_BC, 50, Rot, -100, true, true);
9       until(SENSOR_3 < LightThreshold){
10            OnFwd(OUT_BC, Speed);
11      }
12      OnFwd(OUT_BC, Speed);
13      Wait(500);
14      RotateMotorEx(OUT_BC, 50, Rot, 100, true, true);
15  }
```

Come descritto nella sezione precedentemente, questa funzione non fa altro che permettere al robot di aggirare l'ostacolo una volta identificato; questa subroutine non presenta quindi aspetti complicati, poichè è una semplice alternanza di rotazioni e avanzamenti. Si noti solo l'ultima porzione di codice, che fa in modo che

il patroller, dopo aver oltrepassato l'impedimento, prosegua in avanti finchè non incontrerà nuovamente la linea nera del tracciato.

Per finire, vediamo la parte di codice che permetterà al patrollere di rilevare eventuali intrusioni secondo il criterio descritto in precedenza.

```
1  task alarm(){
2       OnFwd(OUT_A, 100);
3       while(true){
4            PlayTone(440, 250);
5            Wait(200);
6            PlayTone(600, 250);
7            Wait(100);
8            PlayTone(200, 250);
9            Wait(100);
10      }
11 }
12
13 task intrusions(){
14      bool detected = false;
15      Precedes(alarm);
16      while(detected == false){
17           if (SENSOR_2 > IntruderThreshold){
18                detected = true;
19           }
20      }
21 }
```

Si noti subito come questi due frammenti di codici rappresentino dei task. La differenza fondamentale che c'è tra un task ed una subroutine è che più task possono essere eseguiti parallelamente, facendo attenzione a gestire al meglio eventuali risorse condivise; in particolare il task *alarm* viene eseguito dal task *intrusions*,

lanciato dal *main* all'avvio del programma parallelamente al task *move* visto in precedenza. Così mentre *move* si occupa degli spostamenti del patroller, *intrusions* controlla continuamente che non si presentino intrusioni, rappresentate nel nostro caso da una forte luce. Quando questo si verifica, *intrusions* lancia il task *alarm* e termina, mentre questo nuovo task si occupa delle azioni da svolgere per comunicare l'intrusione rilevata, ovvero dell'accensione della spia e dell'emissione del segnale acustico.

Ovviamente il comportamento descritto da *alarm* potrebbe essere gestito all'interno di *intrusions*, ma si è preferito tenerli separati per scelta; infatti essi rappresentano due comportamenti ben diversi, ovvero l'identifcazione dell'intrusione e la segnalazione dell'intrusione stessa, e tenerli separati ci è parsa la soluzione migliore, anche tenendo conto di una possibile estensione futura degli stessi comportamenti.

3.3.4 Verifica sperimentale

La verifica sperimentale realizzata in laboratorio è avvenuta in maniera molto semplice; come già in parte descritto nelle sezioni precedenti, per far sì che il nostro robot sia in grado di muoversi lungo un percorso abbiamo la necessità di tracciare questo percorso a terra. Inizialmente abbiamo tentato di utilizzare delle stampe già presenti in laboratorio, ma ben presto si sono rivelate troppo piccole ed inadeguate per simulare lo scenario che avevamo in mente. Abbiamo quindi segnato un tracciato sul pavimento con del nastro adesivo nero e abbiamo ricreato un ipotetico percorso di patrolling da far seguire al nostro robot.

Abbiamo poi inserito sul tracciato qualche oggetto che potesse rappresentare per il patroller un ostacolo da dover evitare, per poter verificare che esso fosse in grado di farlo. Infine, anche questo già spiegato in precedenza, è data la possibilità di simulare un tentativo di intrusione semplicemente con una lampada che accenderemo nel momento in cui vorremo iniziare quella che potremo considerare una violazione. La figura 3.16 rappresenta schematicamente la struttura della

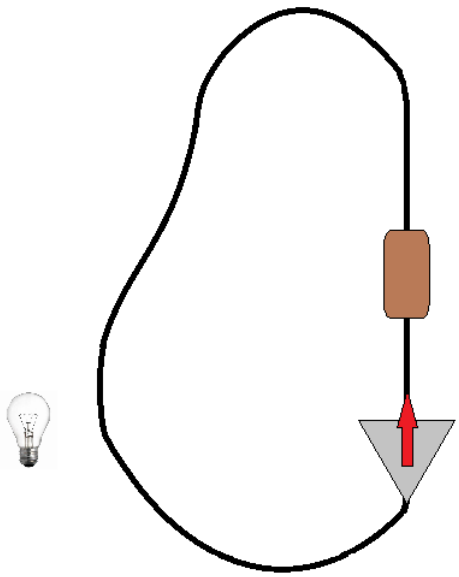

Figura 3.16: Rappresentazione della verifica sperimentale realizzata

simulazione appena descritta.

Il risultato ottenuto dalla simulazione è decisamente positivo: il robot è infatti in grado di realizzare ogni azione descritta. Il patroller percorre il percorso segnato a terra con grande rapidità, aggirando gli ostacoli altrettanto agilmente; inoltre è in grado di identificare quella che noi abbiamo definito come intrusione, ovvero l'accensione di una lampada che emettesse una forte luce. L'esito della verifica sperimentale è risultato, in conclusione, estremamente soddisfacente in quanto il nostro pattugliatore è in grado di svolgere le funzioni desiderate in maniera molto fluida.

Capitolo 4

Conclusioni e sviluppi futuri

In questa tesi è stato studiato il problema del pattugliamento partendo da tre differenti punti di vista; innanzitutto abbiamo affrontato quello che era il nostro obiettivo principale, proponendo una valida strategia di patrolling realizzata sulla base di lavori svolti in passato e sulle relative conclusioni tratte. In seguito abbiamo applicato tale strategia ad un caso di studio da noi definito, descrivendo uno scenario nel quale fosse necessario realizzare del patrolling; abbiamo qui potuto applicare la strategia definita in precedenza e realizzare un sistema di pattugliamento - in linea teorica - adatto alla situazione presentata. Infine abbiamo realizzato concretamente un patroller con i mezzi a disposizione nel nostro laboratorio, ovvero un kit base della linea Lego Mindstorms, che fosse in qualche modo in grado di effettuare del pattugliamento seguendo la strategia da noi definita.

È senz'altro possibile affermare che il lavoro da noi svolto in questa tesi è positivo e fornisce un utile contributo, anche se piccolo, principalmente grazie al raggiungimento del nostro main task, ovvero presentando una strategia di patrolling che sia semplice e al tempo stesso affidabile, nonostante le nostre conoscenze sull'argomento non fossero altro che quelle di base. Infatti tale strategia, nonostante sia stata studiata e proposta sotto alcuni vincoli piuttosto rigidi, permette di

realizzare del pattugliamento la cui difesa dell'ambiente possa considerarsi pressochè ottima, semplicemente negando ai 'ladri' il tempo minimo necessario per una intrusione. La strategia, pur essendo stata proposta conoscendo

Per quanto riguarda lo studio dello scenario da noi proposto, anche in questo caso possiamo affermare che il lavoro da noi svolto possa considerarsi soddisfacente; infatti esso mostra come, secondo la tecnica da noi in precedenza proposta, mettere in pratica la nostra strategia di pattugliamento, considerando uno scenario non semplice come quello descritto. In ottica futura alcune cose andrebbero però riviste e la tecnica dovrebbe essere raffinata: ad esempio, sarà necessario trovare un modo per sostituire il software Concorde nel calcolo del percorso ottimo. Infatti, per quanto questo si sia mostrato utile e prestante, mostra notevoli limiti, i più grandi dei quali sono senza dubbio quello di non poter posizionare dei nodi con precisione o ancora la mancanza della possibilità di impostare la distanza tra due nodi del sistema.

Infine, anche l'esperienza che ci ha visto realizzare un robot pattugliatore può dirsi soddisfacente; infatti considerando il risultato raggiunto tenendo a mente i mezzi a nostra disposizione, anche questa parte del nostro lavoro si è conclusa con un risultato positivo, anche se sicuramente migliorabile. Come più volte detto nei capitoli precedenti, la possibilità di poter sfruttare strumentazioni più adatte o software realizzati appositamente per determinati scopi migliorerebbe di molto il nostro lavoro; gli strumenti a disposizione costituiscono una forte limitazione e in questo senso il nostro lavoro può considerarsi comunque buono. Il nostro patroller è infatti in grado di seguire e rintracciare un percorso, nonchè di evitare ostacoli e di individuare quelle che abbiamo deciso di definire come intrusioni.

Concludendo, possiamo quindi dire che il lavoro svolto è positivo e l'esperienza, oltre ad aver portato buoni risultati, si è dimostrata essere gratificante da un punto di vista personale per i risultati conseguiti e soprattutto per la consapevolezza di aver fornito un contributo - anche se piccolo - al mondo del patrolling.

Appendice A

Program Code

src/patrolbot.nxc

```
1  #define LightThreshold 50
2  #define Speed 65
3  #define Near 25
4  #define Rot 200
5  #define IntruderThreshold 70
6
7  task alarm(){
8       OnFwd(OUT_A, 100);
9       while(true){
10           PlayTone(440, 250);
11           Wait(200);
12           PlayTone(600, 250);
13           Wait(100);
14           PlayTone(200, 250);
15           Wait(100);
16      }
17 }
18
```

```
19 task intrusions(){
20      bool detected = false;
21      Precedes(alarm);
22      while(detected == false){
23           if (SENSOR_2 > IntruderThreshold){
24                detected = true;
25           }
26      }
27 }
28
29 sub avoidObstacles(){
30      RotateMotorEx(OUT_BC, 50, Rot, 100, true, true);
31      OnFwd(OUT_BC, Speed);
32      Wait(1900);
33      RotateMotorEx(OUT_BC, 50, Rot, -100, true, true);
34      OnFwd(OUT_BC, Speed);
35      Wait(1700);
36      RotateMotorEx(OUT_BC, 50, Rot, -100, true, true);
37      until(SENSOR_3 < LightThreshold){
38           OnFwd(OUT_BC, Speed);
39      }
40      OnFwd(OUT_BC, Speed);
41      Wait(500);
42      RotateMotorEx(OUT_BC, 50, Rot, 100, true, true);
43 }
44
45 sub findLineSpiral(){
46      int curve=60;
47      until(SENSOR_3 < LightThreshold ||
48            SensorUS(IN_4) < Near ) {
```

```
49            OnFwdSync(OUT_BC, Speed, curve);
50            Wait(200);
51            if(curve > 20){
52                  curve -= 1;
53            }
54      }
55 }
56
57 sub findLine(){
58      bool lineFound = false;
59      bool turnLeft;
60      if(Random(2) > 0){
61            turnLeft = true;
62      }else{
63            turnLeft = false;
64      }
65      long rotateTimeout = 10;
66      long rotateStartTime;
67      while(lineFound == false){
68            rotateStartTime=CurrentTick();
69
70            if(turnLeft){
71                  OnFwdSync(OUT_BC, Speed, -100);
72            }else{
73                  OnFwdSync(OUT_BC, Speed, 100);
74            }
75
76            until(CurrentTick()-rotateStartTime >= rotateTimeout
77                  || lineFound){
78                  if(SENSOR_3 < LightThreshold){
```

```
79                      lineFound = true;
80                }
81           }
82
83           if(lineFound == false){
84                turnLeft = !turnLeft;
85
86                if(rotateTimeout > 1000){
87                     findLineSpiral();
88                     lineFound = true;
89                }else{
90                     rotateTimeout += 100;
91                }
92           }
93      }
94 }
95
96 task move(){
97      while(SENSOR_3 > LightThreshold){
98           findLineSpiral();
99           if (SensorUS(IN_4) < Near){
100               avoidObstacles();
101          }
102     }
103     while(true){
104          if(SENSOR_3 > LightThreshold){
105               findLine();
106          }
107          if(SensorUS(IN_4) < Near){
108               avoidObstacles();
```

```
109             }
110             OnFwd(OUT_BC, Speed);
111         }
112 }
113
114 task main(){
115         SetSensorLight(IN_3);
116         SetSensorLowspeed(IN_4);
117         SetSensorType(IN_2,IN_TYPE_LIGHT_INACTIVE);
118         SetSensorMode(IN_2,IN_MODE_PCTFULLSCALE);
119         ResetSensor(IN_2);
120
121         Precedes(intrusions, move);
122 }
```

Bibliografia

Testi consultati

[1] M. J. Mataric, *The robotics primer*, The MIT Press, Massachusetts Institute of Technology, 2007

Articoli consultati

[2] Y. Chevaleyre, *The patrolling problem*, Tech. Rep. of Univ. Paris 9, 2003

[3] Y. Chevaleyre, *Theoretical Analysis of the Multi-agent Patrolling Problem*, Lamsade - University Paris 9, International Conference on Intelligent Agent Technology, 2004

[4] A. Machado, G. Ramalho, J.D. Zucker, A. Drogoul, *Multi-Agent Patrolling: an Empirical Analysis of Alternative Architectures*, Third International Workshop on Multi-Agent Based Simulation, 2002

[5] A. Almeida, G. Ramalho, H. Santana, P. A. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre, *Recent advances on multi-agent patrolling*, in SBIA, 2004

[6] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, Tech. Rep. CS-93- 13, Carnegie Mellon University, Graduate School of Industrial Administration, 1976

[7] N. Basilico, N. Gatti, F. Amigoni, *Developing a Deterministic Patrolling Strategy for Security Agents*, Dipartimento di Elettronica e Informazione del Politecnico di Milano, 2009

[8] L. Iocchi, L. Marchetti, D. Nardi, *Multi-Robot Patrolling with Coordinated Behaviours in Realistic Environments*, IEEE RSJ International Conference on Intelligent Robots and Systems 2011, San Francisco, CA, USA, 2011

Siti consultati

[9] Concorde - `www.math.uwaterloo.ca/tsp/concorde/index.html`.

[10] LEGO NXT - `http://mindstorms.lego.com/en-us/whatisnxt/default.aspx`

[11] TSPLIB - `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[12] API NXC - `http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/index.html`

Ringraziamenti

Vorrei innanzitutto rivolgere il mio primo pensiero ai miei genitori e ringraziarli per essermi sempre stati accanto e avermi supportato, non solo in questo mio percorso di studi, ma anche nella vita di tutti i giorni. Grazie per i consigli, le carezze e i rimproveri. Grazie per il tempo dedicato e per gli insegnamenti che mi avete impartito: qualunque cosa di buono io abbia mai fatto o farò in futuro sarà per merito vostro ancor prima che mio.

Ringrazio il professor Andrea Roli, per la grande disponibilità dimostrata nonostante i tanti impegni, e per aver saputo dedicare attenzione alle mie necessità durante lo svolgimento di questa tesi.

Vorrei ringraziare anche Elisa, innanzitutto per avermi sopportato nei difficili momenti di studio e lavoro, ma soprattutto per essermi sempre vicino e dimostrarmi affetto e fiducia ogni giorno sempre di più.

Infine vorrei ringraziare tutti i parenti e gli amici che hanno sempre dimostrato di credere nelle mie capacità ma soprattutto di credere in me come persona; grazie per essermi stati accanto nei momenti più brutti tanto quanto in quelli più belli; grazie per avermi fatto vivere ricordi che porterò per sempre con me e non dimenticherò mai.

Grazie a tutti voi.

Simone