UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Controllo delle traiettorie di Morgul durante il pattugliamento

Elaborato di esame di:

Emanuele Piantoni, Alessandro Zigliani

Consegnato il: **03 settembre 2008**

Sommario

Il pattugliamento di un ambiente da parte di un robot è un compito concettualmente semplice: data una mappa su cui in precedenza sono state segnate alcune posizioni che formano una sequenza $P = <P_1, P_2.., P_n>$, il robot deve percorrere un tragitto che lo porti in ognuna delle posizioni specificate (ovviamente nell'ordine specificato). Si noti che, per posizione, si intende una tripla di coordinate in due dimensioni $P_i = \{x, y, \theta\}$, dove con θ si indica l'orientamento del robot rispetto al sistema di coordinate.

Il compito che il programma qui descritto svolge è cercare, se possibile, di portare a termine tale compito senza informazioni dettagliate riguardo all'ambiente e senza un sistema di localizzazione preciso. Si basa infatti sui soli dati dell'odometria e dei sensori sonar. Come vedremo, questo complica un po' le cose, dal momento che in tale situazione non è possibile effettuare un vero calcolo delle traiettorie.

1. Introduzione

Il robot chiamato Morgul è un robot Pioneer della serie 3. È dotato di 10 sensori di contatto, 5 davanti e altrettanti dietro, e di 16 sonar posti ad una altezza di circa 25 centimetri dal terreno, oltre che di una telecamera che non era oggetto di questo lavoro. In figura 1 è schematizzata la disposizione dei sonar di Morgul e la relativa numerazione.

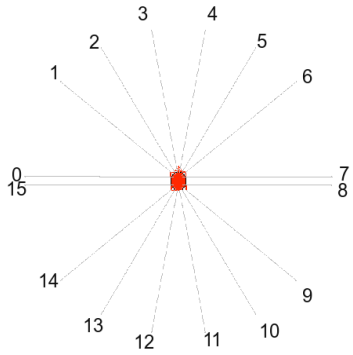

Figura 1 – Disposizione sonar Morgul.

Si suppone che il robot parta per il pattugliamento da un punto le cui coordinate sono conosciute a priori (la stazione di ricarica) e finisca il giro in prossimità di tale punto, ovverosia in una posizione che consenta all'apposito programma di "ritorno a casa" di riportare il robot nella stazione. Tuttavia la responsabilità di individuare il punto finale è delegata all'operatore umano incaricato di redigere la mappa e, in particolare, di segnare su di essa i punti da visitare. A questo scopo l'operatore può utilizzare il programma Mapper3basic.
Nel nostro caso, l'ambiente di lavoro era naturalmente il laboratorio di Robotica Avanzata della facoltà di ingegneria. Il locale è abbastanza ridotto da presentare una sfida in presenza di ostacoli troppo voluminosi o dalla geometria sfavorevole. A tale proposito segnaliamo l'attaccapanni di plastica nera che è dotato di un corpo a cilindri concentrici, e per di più ha una base formata da quattro gambe d'appoggio che sporgono dal corpo per una ventina di centimetri. Essendo alte pochi centimetri sono del tutto invisibili ai sonar e vengono urtate immancabilmente dal robot, se le trova sul suo cammino.

Un altro punto a sfavore era la virtuale invisibilità dei banchi di lavoro che, essendo cavi, non sono parimenti visti dai sonar. Ciò comporta problemi proprio per Morgul, la cui struttura si eleva ben oltre l'altezza delle scrivanie e lo rende propenso a urtarne i ripiani se l'odometria ha accumulato errori troppo consistenti. La situazione si verifica se durante un pattugliamento il robot è costretto spesso a deviare da una traiettoria rettilinea.

2. Il problema affrontato

L'esecuzione del programma prevede la definizione di alcuni punti in modo che coprano l'intero laboratorio e in modo che siano raggiungibili (almeno a priori) da parte del robot Morgul. Inizialmente, è previsto che il robot esca dalla postazione spostandosi all'indietro di circa un metro. A questo punto

esegue una rotazione su se stesso di 180 gradi e comincia a seguire il percorso. Non è prevista una vera e propria pianificazione delle traiettorie da un punto al successivo, ma viene usato un gruppo di azioni che provvede a ruotare il robot nella direzione del prossimo punto e a farlo procedere in linea retta fino al raggiungimento del punto stesso. Contestualmente, il robot tenta di evitare urti contro eventuali ostacoli rallentando fino a fermarsi, se necessario.

Nonostante di solito il programma funzioni correttamente, possono verificarsi situazioni spiacevoli e comportamenti non tollerabili, in quanto posso causare danni al robot stesso oppure far sì che il robot si blocchi e non porti a termine la propria missione di sorvegliante. In particolare, tali situazioni appartengono a tre categorie principali. Una prima classe di problemi è quella riguardante il superamento degli ostacoli. Infatti nella versione operativa su Morgul non è prevista nessuna specifica procedura che permetta un superamento di un ostacolo che è posto sulla traiettoria e che intralcia il movimento del robot: quello che Morgul tenterà di fare è di mantenersi in linea retta verso il punto destinazione, urtando inevitabilmente contro l'ostacolo. Quello che è previsto dal programma nel caso in cui il robot abbia davanti a sé un ostacolo, sono soltanto due azioni di diminuzione della velocità: comunque non è sufficiente perché non evita che il robot sbandi contro l'ostacolo. Questa classe di problemi si presenta solo quando ci sono ostacoli e non si presenta in una situazione di normalità. La conseguenza più rilevante che causa, infine, è il fallimento dell'esecuzione dovuto al controllo posto sui motori per rilevare situazioni di stallo, cioè quando il robot è bloccato.

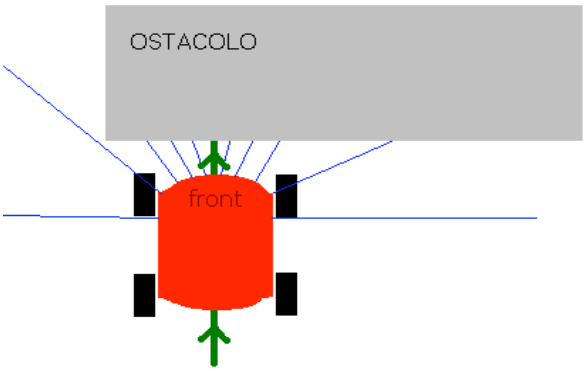

Figura 2 – Esempio di cosa succede nel caso accada un problema rientrante nella prima classe

Una seconda classe di sottoproblemi è quella riguardante il non raggiungimento di un particolare punto destinazione, nonostante non ci siano ostacoli presenti che impediscono il suo raggiungimento. Infatti durante l'esecuzione, può capitare che il robot venga a trovarsi in un punto abbastanza vicino al un punto destinazione, ma che non coincide con il punto destinazione stesso. Il problema che si presenta in questo caso è che da quel punto il robot non riesce a raggiungere il punto destinazione, perché non esistono delle facili manovre che lo permettono. Questo problema è accentuato anche dalle caratteristiche fisiche di Morgul: infatti le ruote rispettano la struttura *skid steering*. Ciò si potrebbe verificare anche in assenza di ostacoli e nel caso ce ne siano viene comunque amplificato. Questa classe di problemi genera un comportamento inaccettabile: causa un movimento circolare su se stesso pressoché infinito. Anche nel caso in cui riesca a raggiungere il punto, la continuazione del programma è falsata perché Morgul usa, per il calcolo della sua posizione, un metodo odometrico che consiste nel misurare lo spostamento, valutando il moto delle ruote: quell'inutile movimento circolare causa un calcolo sbagliato della sua posizione (Morgul crede di essere in una diversa posizione rispetto a quella reale) che ha ripercussione sull'intero programma in modo irrimediabile.

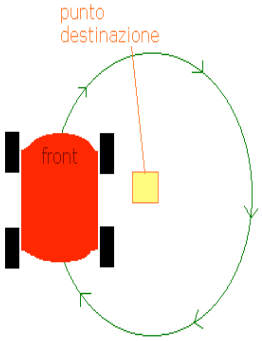

Figura 3 – Esempio di cosa succede nel caso accada un problema rientrante nella seconda classe

Una terza ed ultima classe di problemi è quello riguardante il non raggiungimento di un particolare punto destinazione, in questo caso a causa di un o più ostacoli che intralciano il robot nel raggiungimento del punto. Infatti, nella versione che Morgul usa per adempire il suo dovere di sorvegliante, nel caso in cui si verifica questo problema non si accorge che il punto destinazione non è raggiungibile ed insiste in continuazione per poterci arrivare con un movimento in linea retta. Le conseguenze sono simili ai problemi della prima classe: nella maggior parte dei casi infatti accadrà che l'esecuzione del programma verrà bloccata perché sarà rilevata una situazione di stallo. Quello che non viene tollerato in questo caso non è tanto l'atteggiamento del robot nei confronti dell'ostacolo (altrimenti anche questo problema rientrerebbe nella prima classe), ma il fatto che il robot si intestardisca nel raggiungere il punto, proprio perché non è detto che se un punto destinazione non sia raggiungibile, gli altri non lo siano.

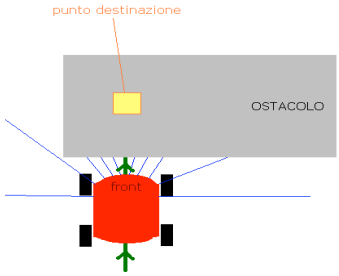

Figura 4 – Esempio di cosa succede nel caso accada un problema rientrante nella terza classe

Come detto nel paragrafo precedente, esistono altri tipi di errori causati dalle imprecisioni dei sensori o dalle caratteristiche dell'ambiente circostanze a cui si può fare molto poco: si tratta di errori nella lettura della distanza con sensori sonar oppure di errori nel calcolo della posizione affidato ad un metodo odometrico oppure la presenza di ostacoli cavi (per esempio sedie o tavoli) che non sono visibili al robot. In tutti questi casi Morgul rischia di schiantarsi e di capovolgersi.

Nel prossimo paragrafo saranno proposte delle soluzione alle classi di problemi rilevate durante l'esecuzione del programma e descritte qui sopra.

3. La soluzione adottata

Il metodo usato per risolvere il problema è in realtà la composizione di tre metodi diversi a cui corrispondono tre delle sei diverse modalità di funzionamento del robot, atte a risolvere tre varianti del problema. Ad ognuna di queste modalità corrisponde nel codice un tipo enumerativo chiamato `Behaviour` che ha come possibili valori `NONE`, `SIMPLE`, `AVOID`, `WANDER`, `FAIL_NEAR` e `FAIL`. Il passaggio tra una modalità e l'altra è compiuto nella funzione `checkBehaviour()`, che viene chiamata periodicamente.

3.1. NONE (Nessun ActionGroup associato)

Si tratta dello stato iniziale, in cui il robot si trova all'inizio del programma e ogni volta in cui ha raggiunto la destinazione corrente. In questo caso non è attivo nessun comportamento e perciò il robot sta fermo nella posizione in cui si trova. Da questa modalità si passa necessariamente a `SIMPLE`.

3.2. SIMPLE (ActionGroup simpleAG)

Questa modalità di funzionamento era già presenta nella prima versione del programma e considera il problema ridotto all'osso: muoversi da un punto ad un altro punto in linea retta alla velocità indicata. Quando il robot si trova in questo stato non effettua alcuna azione per modificare la propria direzione, ma rallenta fino a fermarsi in caso rilevi un ostacolo in avvicinamento di fronte a sé. Nel linguaggio della libreria Aria, è la composizione di tre tipi di *action*:

- `ArActionGoto`, che si occupa di raggiungere il punto specificato
- `ArActionLimiterForwards`, che si occupa di limitare la velocità in presenza di ostacoli
- `ArActionAvoidSide`, che si occupa di non passare troppo radente al muro o a eventuali ostacoli laterali che la precedente non è in grado di rilevare

Nel caso ideale, il robot permane in questo stato per la maggior parte del tempo.

3.3. AVOID (ActionGroup avoidAG)

Nel caso in cui il robot rallenti, entra in funziona questa modalità più cauta. Essa è una variante della precedente, ed è utile per ostacoli di dimensioni limitate che in realtà intralciano poco il cammino del robot e che è possibile evitare con relativa facilità senza deviare troppo dal percorso. Si noti che in questo caso il robot non ha bisogno di rallentare, essendo che la bassa velocità è ciò che lo induce a entrare in questa modalità di funzionamento.

Il robot rimane in questo stato finché o *la velocità aumenta* oppure *il robot si allontana dal punto che deve raggiungere*, data una certa tolleranza all'allontanamento o avvicinamento (es. 200 mm). Infatti, periodicamente il programma salva la distanza minima raggiunta dall'attuale destinazione e con essa il tempo *t* al quale viene fatta tale rilevazione. Se la distanza non diminuisce e il robot *non* si trova nelle strette vicinanze della destinazione corrente (nel qual caso si assume che l'ostacolo ostruisca il punto in questione e si entra nella modalità `FAIL_NEAR`[1]), dopo un certo numero di secondi si passa alla modalità `WANDER`[2].

L'ArActionGroup relativo al modo AVOID è composto dai seguenti tipi di action:

- `ArActionGoto`, analogo al precedente
- `ArActionAvoidFront`, che si occupa di far deviare in modo consistente il robot in presenza di ostacoli frontali
- `ArActionAvoidSide`, analogo alla modalità precedente
- `ArActionStallRecover`, che in sostanza tenta di liberare il robot se questo si trova in stallo perché in una posizione che non gli permette di proseguire (accade sempre nel caso di urto con l'attaccapanni, invisibile ai sonar)

---

[1] Il valore per cui l'ostacolo si assume "vicino" è dato dalla define `FAIL_NEAR_TOLERANCE`.

[2] Il valore di tale intervallo di tempo è dato dalla define `MAX_AVOID_TIME`. Sembra inoltre opportuno rimarcare che il timer viene reimpostato ogni volta che il robot muove un passo che lo avvicina all'attuale destinazione, il che significa che può rimanere nel modo `AVOID` molto più della durata specificata.

3.4. WANDER (ActionGroup wanderAG)

Com'è intuibile, il robot nella modalità `AVOID` è soggetto a cadere in trappola in presenza di minimi relativi, come ad esempio nel caso di ostacoli non convessi in grado di "abbracciare" il robot e le cui concavità siano rivolte proprio nella direzione opposta al movimento del robot verso il punto. Occorre quindi che esso smetta temporaneamente di rivolgere cocciutamente il proprio moto verso la destinazione[3] e tenti dei percorsi alternativi che potrebbero, se la fortuna gli arride, portalo fuori dal minimo relativo.

Il robot rimane in questo stato finché, vagando in modo casuale, non riesce a rilevare (usando tutti i sedici sonar) nella direzione verso il punto destinazione una zona sgombra: se sussiste tale situazione allora il robot effettua una rotazione verso tale direzione e ritorna nella modalità AVOID.

Il cambio alla modalità AVOID avviene anche se scade un timer[4]. Inoltre, il robot ha a disposizione un numero di tentativi limitati per utilizzare questa modalità casuale di funzionamento, dopodiché si assume il fallimento e si entra in modalità FAIL.

L'ArActionGroup relativo al modo WANDER è composto dalle seguenti action:

- `ArActionLimiterForwards`, analogo al precedente
- `ArActionAvoidFront`, analogo al precedente
- `ArActionStallRecover`, analogo al precedente
- `ArActionConstantVelocity`, che si occupa di garantire un moto rettilineo con velocità costante

Nella modalità WANDER vengono definite due azioni ArActionLimiterForwards: una agisce quando il robot è distante dall'ostacolo, l'altra invece quando il robot è vicino all'ostacolo.

La definizione della modalità WANDER risulta essere semplice, ma nello stesso tempo molto efficace nello svolgere il proprio compito. Questa è risultato di varie esperienze e un compromesso tra diverse definizioni: si è passati da definizioni inutilmente complesse (si cercava di dare un comportamento meno casuale, facendo seguire il robot la via più promettente) alla definizione dell'ActionGroup WANDER della libreria Aria, accentuando inutilmente la casualità del movimento del robot .

3.5. FAIL_NEAR

Quando un punto destinazione non è raggiungibile ma è molto vicino (a una distanza minore di `FAIL_NEAR_TOLERANCE`) alla posizione che dovrebbe raggiungere, rinuncia. Tuttavia non esce dal programma, ma passa al punto successivo. Si noti che si arriva in questa modalità solo dalla modalità `AVOID`.

3.6. FAIL

Stato del robot che si verifica quando un punto destinazione non è raggiungibile, cioè quando la modalità WANDER fallisce. La modalità FAIL non ha alcuna azione: una volta che il robot si trova in questo stato si ferma e non procede oltre. Il pattugliamento finisce.

Questo modo forse un po' brusco è cautelativo. Si è ritenuto infatti che, in presenza di una mappa corretta e di punti raggiungibili, il fatto che robot non riesca a raggiungere una certa posizione può essere dovuto ad errori rilevanti nell'odometria. Riteniamo che, specie in ambienti pericolosi come il laboratorio di robotica, sia controproducente farlo procedere oltre.

---

[3] Sembra opportuna rimarcare che il timer viene reimpostato ogni volta che il robot muove un passo che lo avvicina all'attuale destinazione, il che significa che può rimanere nel modo `AVOID` molto più di 30 secondi.

[4] Il periodo massimo di permanenza nella modalità `WANDER` è dato dalla define `MAX_WANDER_TIME`.

3.7. Risoluzione del problema

Riepilogando, l'idea di risoluzione dei problemi, spiegati al paragrafo precedente, sta nel cambiare ActionGroup: così facendo il superamento degli ostacoli avviene con successo (se non è troppo ingombrante) e la rinuncia di arrivare in un punto destinazione è implicito nel meccanismo appena definito.

Per quanto riguarda la classe di problemi della rotazione del robot su se stesso, la soluzione del problema è data dai meccanismi del metodo AVOID, che garantiscono l'uscita del robot da questa situazione dopo che per un certo tempo questi tenta di avvicinarsi alla destinazione senza riuscirci.

Per quanto riguarda gli altri problemi legati all'ambiente e alle imprecisioni del robot, si può fare molto poco: esiste comunque una tolleranza nell'azione `ArActionGoto`, di circa 20 cm. Inoltre, quando si tratta di decidere se il robot si sta effettivamente avvicinando alla destinazione, non vengono considerati significativi spostamenti inferiori a 20 cm. L'effetto delle due garantisce una certa elasticità.

4. Modalità operative

Per poter eseguire il programma patrolmap, si deve installare sul proprio calcolatore la libreria Aria (scaricabile dal sito del laboratorio di Robotica Avanzata) e un compilatore C++, come ad esempio g++. Il programma può essere eseguito sul robot o su un simulatore (ad esempio MobileSim, scaricabile gratuitamente dall'azienda produttrice). Se si vuol far eseguire il programma sul robot, bisogna collegarsi al robot stesso: via rete (anche wireless) oppure direttamente via seriale. Se invece si vuole eseguirlo con un simulatore basta avviare il simulatore. Nel programma esiste una procedura automatica di connessione che cerca, se esiste, un simulatore oppure un robot disponibile.

Per poter compilare basta eseguire il comando "`make`" dalla shell. Lanciandolo dalla cartella del programma o dalla sottocartella `Release` si ottiene una versione ottimizzata, mentre lanciandolo dalla cartella `Debug` si ottiene una versione non ottimizzata. Buona norma è quella di compilare ogni volta il codice sorgente se si lavora su diversi calcolatori, perché poterebbero esserci installate, per esempio, versioni diverse delle librerie Aria oppure diverse versioni del compilatore sulle diverse macchine: non basta trasferire l'eseguibile da un calcolatore ad un altro.

Con il comando "`make dist`" viene creato un archivio con tar e compresso gzip dei file del progetto, mentre con il comando "`make install`" il programma viene compilato e installato nella cartella "`/usr/bin`".

Per poterlo eseguire, basta digitare "`./patrol_map -map <map>`". Si deve infatti passare al programma un parametro: la mappa dell'ambiente circostante, che il robot si appresta a pattugliare (`<map>` è il nome del file della mappa, evidenziando tutto il percorso del file system se il file non è nella stessa directory del file eseguibile).

Il programma non richiede altro, e può terminare la propria esecuzione con diversi codici di uscita:

0. nessun errore
1. errore di inizializzazione (problema nell'apertura in scrittura dei file di log o nei parametri passati)
2. stallo (il robot si è trovato in una situazione di stallo non reversibile)
3. fallimento di un obiettivo (il programma termina perché non riesce a raggiungere un obiettivo)

4.1. Architettura del programma

L'architettura della nuova versione del programma è migliorata rispetto alla versione precedente. Infatti prima il programma era composto da un solo file chiamato `patrolmap.cpp`, mentre ora il programma è costituito da più file: ogni file ha una precisa responsabilità e un preciso compito da soddisfare.

Esaminando più in dettaglio, questi sono i file più importanti che costituiscono complessivamente il sistema software:

- `behaviour.cpp`: contiene i metodi responsabili della definizione delle diverse ActionGroup e responsabili della gestione di cambiamento da un comportamento ad un altro;
- `dump.cpp`: in questo file è definito il thread che gestisce la scrittura dei file (un file per indicare lo stato della batteria, un file per indicare se si è verificato uno stallo e un sile per indicare la posizione corrente) e il controllo se il robot è in stallo;
- `utils.cpp`: in questo file ci sono tutti quei metodi che svolgono calcoli, indispensabili per far mutare il comportamento al robot oppure le funzioni per costruire il vettore delle posizioni da raggiungere ricavate dal file .map oppure metodi per far muovere il robot in un certo modo (ad esempio rotate);
- `patrolmap.cpp`: in questo file è contenuto il main, inizializza il thread del robot, fa compiere con dei comandi diretti delle azioni al robot e poi, invocando i metodi opportuni degli altri file, controlla che il robot arrivi in prossimità delle posizioni che deve raggiungere.
- `defines.h`: contiene alcune interessanti costanti globali che servono per la messa a punto del programma.

4.2. Componenti necessari

Componenti software necessari per far funzionare il sistema realizzato sono il già citato compilatore del linguaggio C++, la già citata libreria Aria ed un simulatore (anche se non necessario al fine del programma se si ha la disponibilità di un robot, ma potrebbe essere comunque utile soprattutto per una fase di testing). Invece per quanto riguarda componenti hardware è necessario avere collegato al calcolatore, in qualche maniera, un robot: questo, affinché il programma funzioni correttamente, deve essere dotato di sedici sonar. Possibilmente i sonar dovrebbero essere disposti in questo modo:

- sei sonar, disposti nella parte interiore del robot;
- sei sonar , disposti nella parte posteriore del robot;
- due sonar, disposti sul lato sinistro del robot;
- due sonar, disposti sul lato destro del robot.

Questo è l'unico vincolo strutturale: anche se il programma è stato progettato ed implementato per il robot Morgul che è un Pioneer della serie 3, questo non significa che, per funzionare in modo corretto, il robot debba appartenere a questa categoria.

4.3. Modalità di installazione

Il sistema software realizzato non richiede alcun componente né hardware né software specifico. Sono semplicemente richiesti i componenti software descritti al paragrafo precedente, comuni per tutti i programmi che devono essere eseguiti da un robot. Come già detto prima, il programma necessita soltanto di una compilazione del codice sorgente e la sua esecuzione. Per installarlo nel sistema, ovvero nella cartella "`/usr/bin`" è possibile usare il comando "`make dist`".

4.4. Modalità di taratura

Il programma ha dei parametri che è possibile personalizzare a seconda del tipo di ambiente e delle dimensioni degli ostacoli non previsti che si possono trovare sul cammino del robot. Infatti, è intuitivo che se l'ambiente è grande e gli ostacoli imprevisti altrettanto, i tempi in cui il robot ha necessità di trovarsi nelle modalità `AVOID` e `WANDER` possono essere molto diversi. Durante la stesura del programma si è cercato quindi di astrarre in certa misura dall'esempio del laboratorio di robotica.

A tal proposito, rimarchiamo i seguenti parametri contenuti nel file "`defines.h`", che potrebbero essere ritoccati per migliorare in certi casi il comportamento del robot:

- `TIMETOSLEEP` – intervallo per il polling da parte dei vari thread del programma, asincroni rispetto al ciclo del robot (predefinito 200 millisecondi)
- `MAXLOOP` – numero di cicli di polling dopo cui si intende che si sia verificato uno stallo; fatti salvo ritardi dovuti alla computazione, si può pensare che se per un tempo pari a `TIMETOSLEEP * MAXLOOP` il robot non si muove, si sia verificato uno stallo irreversibile (il valore predefinito è 5)
- `FAIL_NEAR_TOLERANCE` – si considera "vicino" un punto che non si riesce a raggiungere ma che sta ad una distanza minore di questa. In questo caso si abbandona e si passa al punto successivo.
- `MAX_AVOID_TIME` – tempo massimo in cui rimanere nel modo `AVOID` senza che si compia alcuna mossa che avvicina il robot all'obiettivo (predefinito 10 secondi)
- `MAX_WANDER_TIME` – tempo massimo in cui rimanere nel modo `WANDER` (predefinito 10 secondi)
- `MAX_WANDER_TRIES` – numero massimo di volte in cui è lecito tentare col modo `WANDER` per sbloccarsi da una situazione in cui non si riesce a raggiungere l'obiettivo con `AVOID`
- `SPEED_WANTED` – la velocità desiderata per il robot (predefinita 400 mm/s)

5. Conclusioni e sviluppi futuri

Il sistema realizzato risolve in modo ragionevole i problemi presentati nel paragrafo 2. Si può essere soddisfatti del risultato ottenuto, anche se possono capitare situazioni poco piacevoli dovute alle insidie del mondo reale o dalle imprecisioni dei sensori del robot. Infatti nel simulatore le cose funzionano più che bene, ma nella realtà può succedere qualche inconveniente causato dalla disposizione degli oggetti del laboratorio in posizioni che li rendono non visibili al robot (per esempio la morsa che c'è sul tavolo o il tavolo stesso) oppure per colpa della forma di questi (per esempio il portaombrelli che ha una parte invisibile al robot, in quanto più bassa dei suoi sonar): il pericolo è che il robot si possa ribaltare e danneggiarsi seriamente. Si è cercato comunque di essere abbastanza prudenti nel definire i diversi parametri in modo da rendere queste situazioni il meno frequente possibile, ma ciò non significa che queste non possano più accadere.

Tuttavia, un programma che funzioni meglio è difficile se si usa l'attuale strumentazione installata su Morgul (costituita principalmente dai sonar). Un'altra soluzione potrebbe essere quella di utilizzare la mappa dell'ambiente per svolgere una pianificazione delle traiettorie vera e propria.

Per quanto riguarda gli sviluppi futuri dell'applicazione, a nostro avviso si dovrebbero concentrare sul miglioramento dei meccanismi che inducono il passaggio da una modalità di funzionamento all'altra. L'utilizzo da noi fatto del programma durante la sua stesura ed il debugging ha evidenziato come la struttura di funzionamento sia potenzialmente adatta a funzionare in molte situazioni "ragionevoli" (es. ostacoli che non intrappolino il robot, o zone pericolose per lo stesso robot), fatta salva un'opportuna taratura dei parametri o dei meccanismi che determinano il passaggio tra la modalità `AVOID` e la modalità `WANDER`. Potrebbe essere interessante capire se è possibile adattare tali parametri dinamicamente durante l'esecuzione.

Bibliografia

[1] ActivMedia Robotics: "Pioneer 3 & Pioneer 2 H8-Series Operations Manual", www.ing.unibs.it/*~arl/docs/documentation/Pioneer 3 documentation/P3-P2H8OpMan3.pdf*, versione 3, agosto 2003.

[2] ActivMedia Robotics: "ARIA overview 2.4.1", www.ing.unibs.it */~arl/docs/documentation/Aria-Saphira documentation/Current/ARIA docs/*, 2005.

Indice