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)

WallTracking: un programma per l'esplorazione perimetrale di un ambiente.

Elaborato di esame di:

Davide Ferrari, Michele Guion

Consegnato il: **13 luglio 2010**

Sommario

Il lavoro svolto consiste nello studio e sviluppo di un software per l'esplorazione perimetrale, da parte di un robot, di un ambiente dotato di pareti. L'obiettivo è di mantenere il robot ad una distanza specificata dalle pareti, prevedendo anche la presenza di ostacoli e angoli.

1. Introduzione

Il problema considerato consiste nella stesura di un programma che permetta ad un robot di esplorare un ambiente, mantenendosi parallelo alle pareti dell'ambiente stesso.

1.1. Precondizioni sull'ambiente

Individuiamo, pertanto, alcune precondizioni sull'ambiente da considerare, in assenza delle quali il corretto funzionamento del programma non può essere garantito.

Innanzitutto l'ambiente, si trovi esso al chiuso o meno, deve essere delimitato da pareti o da ostacoli che il robot possa seguire. Tali ostacoli devono essere alla stessa altezza dei sonar del robot, dal momento che la rilevazione degli ostacoli avviene appunto mediante tali sensori.

Inoltre gli ostacoli devono essere sufficientemente regolari: in particolare devono poter essere assimilati a pareti (non devono quindi essere eccessivamente ondulati, forati, ecc.); la distanza tra gli ostacoli deve essere sufficientemente superiore alle dimensioni del robot, che deve poter passare tra gli ostacoli e avere anche un discreto spazio di manovra in caso di angoli o ostacoli sporgenti.

Infine, la superficie di appoggio delle ruote del robot deve essere adatta alle ruote stesse: in particolare, per l'uso in esterni è necessario utilizzare un robot con ruote adatte a tali ambienti.

Nel caso specifico del Laboratorio di Robotica ARL della Facoltà di Ingegneria dell'Università degli Studi di Brescia, il funzionamento non è garantito nell'area occupata da tavoli e sedie, che non sono infatti assimilabili a pareti. Anche ostacoli eccessivamente ridotti, come il piccolo *Rovio* o il piedistallo del tecnigrafo, sono di difficile rilevazione da parte dei sonar, e pertanto il risultato ottenuto potrebbe non essere quello atteso. Navigando invece parallelamente ai muri, il funzionamento è normalmente quello voluto, tranne in alcuni casi particolari (solitamente in manovra) in cui il robot si trova in posizioni equidistanti da due ostacoli, rendendo il risultato non-deterministico.

Si avrà cura, inoltre, di inserire un controllo di stallo, in modo tale che anche in caso di collisioni il robot possa tentare di riprendere l'esplorazione. La velocità di funzionamento, ovviamente, dovrà essere sufficientemente ridotta da evitare danni al robot o agli accessori ad esso connessi, come marker o calcolatore.

1.2. Robot utilizzato

Il programma è stato scritto in C++, utilizzando le librerie Aria. È stato pensato per un robot del modello Pioneer1 della ActivMedia (diventata poi MobileRobots e ora Adept), nel caso specifico è stato testato sul robot Speedy del laboratorio ARL. Esso è un robot anolonomo, dotato di differential drive, fattore di cui si dovrà tenere conto nella stesura del programma, per quanto riguarda le modalità di manovra. I sonar sono installati solamente nella parte anteriore, e pertanto le misure effettuate devono tener conto di tale limitazione.

Si dovrebbe fare in modo, inoltre, che l'idea di base su cui si baserà il software sia adattabile ad altri modelli di robot, comunque dotati di ruote e sonar, semplicemente modificando i parametri del sistema.

1.3. Idea di base

Il comportamento voluto verrà ottenuto creando un programma, con la struttura standard introdotta a lezione, che utilizza alcune azioni contenute nel pacchetto Aria ed un'ulteriore azione creata appositamente, che perseguirà l'obiettivo di seguire gli ostacoli, mantenendo il robot il più possibile parallelo ad essi.

A grandi linee, l'azione viene creata passando ad essa alcuni parametri, che determinino la velocità del robot e la distanza da mantenere rispetto agli ostacoli. Vengono utilizzate le misure di distanza effettuate mediante i sonar. Il robot cerca l'ostacolo più vicino e si muove verso di esso fino a posizionarsi alla distanza desiderata (misurata coi sonar laterali), dopodiché procede in avanti, alla velocità impostata, in direzione parallela all'ostacolo.

Quando i sonar frontali rilevano una diminuzione della distanza, significa che il muro presenta un angolo acuto, e pertanto il robot vira di conseguenza.

Quando invece i sonar compresi tra il fronte e il lato rilevano misure in notevole aumento è probabile che l'ostacolo stia per terminare oppure abbia un angolo ottuso; pertanto il robot viene fatto leggermente curvare nella direzione dell'angolo per facilitare la svolta in tale direzione, e non trovarsi in condizioni critiche tali per cui il robot si allontana eccessivamente dall'ostacolo, anche al punto di abbandonarlo.

2. Il problema affrontato

A questo punto è possibile analizzare, con maggiore livello di dettaglio, i principali problemi da risolvere durante lo sviluppo del software introdotto.

Come già affermato, infatti, il programma da sviluppare deve permettere al robot, su cui viene eseguito, di esplorare un certo ambiente, cercando, in particolare, di seguirne i muri o, comunque, tutti quegli ostacoli che ne definiscono il perimetro (ad esempio, armadi, cassettiere o, ove possibile, tavoli). Per raggiungere un tale scopo, è necessario:

- per prima cosa, fare in modo che il robot, collocato in una posizione iniziale qualsiasi, sia in grado di avvicinarsi autonomamente al perimetro che deve essere seguito, ossia ad uno degli ostacoli che lo definiscono;
- successivamente, una volta che tale perimetro da seguire è stato raggiunto, rendere il robot in grado di muoversi su una direzione, per quanto possibile, parallela alla linea del perimetro stesso, mantenendo una certa distanza, imposta dall'esterno, e, eventualmente avvicinandosi o allontanandosi opportunamente, qualora tale distanza non sia rispettata;
- poiché il perimetro da seguire non deve essere necessariamente rettilineo (anzi, potenzialmente è caratterizzato da molti angoli, cambi di direzione ecc.), fare sì che il robot riesca a riconoscere la

presenza di eventuali angoli, e, di conseguenza, a ruotare opportunamente, in modo da proseguire con la propria esplorazione;

- evitare, ovviamente, di urtare i vari ostacoli che si presentano sulla traiettoria del robot, primi fra tutti gli ostacoli che delineano il tracciato da seguire.

Per quanto riguarda, in particolare, la necessità di rotazione in presenza di eventuali angoli o cambi di direzione, si deve essere in grado di proseguire con la propria esplorazione sia a fronte di angoli acuti, in prossimità dei quali il robot è in grado di vedere, frontalmente, il proseguimento del perimetro che sta seguendo, sia a fronte di angoli ottusi, presso i quali non è possibile fare affidamento su alcun tipo di "indicazione", come quella costituita dal proseguimento degli ostacoli (fatto che potrebbe rappresentare una difficoltà).

Come requisito non funzionale, inoltre, è stato imposto di sviluppare il programma corrente in modo che esso potesse essere utilizzato da modelli di robot con sensori sonar disposti soltanto nella parte anteriore, in particolare i robot ActivMedia Pioneer1, disponibili anche presso il laboratorio ARL (*Speedy* e *Tobor*).

Le modalità con cui il programma è stato realizzato, risolvendo i problemi appena introdotti, sono descritte nei paragrafi successivi.

3. La soluzione adottata

I problemi appena introdotti sono stati risolti sviluppando un'opportuna *Action*, che permette al robot di seguire il profilo degli ostacoli che esso incontra, mantenendo una certa distanza, fornita in ingresso, ed una direzione parallela, dopo essersi avvicinato all'ostacolo percepito come più vicino (partendo da posizione iniziale qualsiasi); tale *Action* è stata, poi, inserita in un apposito programma main che, sfruttando anche altre azioni disponibili nelle librerie Aria, permette, sì, al robot di seguire il tracciato delimitato dai vari ostacoli, ma anche di muoversi con velocità costante, di evitare gli ostacoli sul proprio cammino, ecc.

L'azione che permette di seguire il perimetro, denominata *ArActionTrack*, è contenuta nei file ArActionTrack.cpp e ArActionTrack.h; il programma main che, istanziando tale azione, oltre ad altre azioni della libreria Aria, permette di seguire il perimetro senza urtare ostacoli, è composta dai file WallTracking.cpp e WallTracking.h, andando a costituire il programma che, una volta compilato, è stato denominato *WallTracking*.

La descrizione della soluzione adottata per risolvere i problemi elencati in precedenza, può proseguire, a questo punto, con la descrizione del main e dell'Action appena introdotti; si parte, in particolare, descrivendo, dapprima, i dettagli del programma main, per avere un'idea globale del comportamento del software completo, e del modo in cui le varie azioni riescono a collaborare, per raggiungere lo scopo descritto; solo successivamente vengono presentati i dettagli di *ArActionTrack*, che specificano come viene realizzata l'esplorazione perimetrale.

3.1. WallTracking

Il programma main che gestisce il comportamento completo del robot, contenuto in WallTracking.cpp, può essere descritto, in prima battuta, nel seguente modo: dopo avere creato una variabile che rappresenta il robot, denominata *tobor*, ed una variabile che ne rappresenta i sensori sonar, vengono create varie azioni, che, con parametri opportuni, permettono al robot di recuperare da eventuali stalli dei motori, di evitare ostacoli frontali vicini e lontani, di muoversi con velocità costante, oltre a consentirgli di esplorare il perimetro dell'ambiente, delineato dagli ostacoli; le azioni sono, in seguito, aggiunte al robot, ciascuna con una certa priorità. Come noto, il programma permette di eseguire le azioni, create ed aggiunte al robot, secondo una logica di tipo *fuzzy*, in cui le varie azioni hanno la priorità imposta durante la stesura del codice.

Analizzando più in dettaglio le *Action* aggiunte al programma, si nota che: inizialmente, viene aggiunta un'azione per il recupero da eventuali stalli dei motori, ossia *recoverAct*; si adotta, in seguito, un'istanza dell'azione *ArActionTrack* sviluppata in questo progetto, e denominata *segui*, cui sono passati, come

parametri (oltre al nome dell'azione), la distanza da mantenere rispetto all'ostacolo di cui seguire il profilo (in questo caso 130 mm), una misura di tolleranza, o, in altri termini, un'isteresi, associata alla distanza da mantenere (30 mm), l'angolo con cui ruotare qualora ci si sia avvicinati o allontanati troppo dal perimetro da seguire (10 gradi), la velocità da mantenere durante l'esecuzione dell'*Action* corrente (50 mm/s); si aggiungono, successivamente, due azioni, istanza di *ArActionAvoidFront*, che permettono di evitare ostacoli frontali, rispettivamente *avoidFrontNearAct*, i cui valori dei parametri di distanza sotto cui spostarsi, velocità da mantenere, angolo di rotazione necessari a non urtare l'ostacolo, consentono di evitare ostacoli a breve distanza (90 mm, con velocità nulla e angolo di 90 gradi), e *avoidFrontFarAct*, con parametri atti ad evitare ostacoli più distanti (190 mm, velocità 100 mm/s, angolo 15 gradi) – è da notare, in più, che è proprio il comportamento dovuto a queste due *Action* che consente al robot di comportarsi correttamente a fronte dei cosiddetti angoli acuti –; si adotta, infine, un'azione, *constantVelocityAct*, che permette di mantenere velocità costante (parametro di velocità di 50 mm/s).

Le priorità imposte per le varie azioni sono: priorità massima, pari a 100, per l'azione *recoverAct*, che permette di recuperare da stalli dei motori (quando questa evento si verifica, l'azione indicata deve essere in grado di togliere il controllo a qualsiasi altra azione); priorità molto simili per le due azioni *avoidFrontNearAct* e *avoidFrontFarAct*, ossia, rispettivamente, 50 e 49, minori di quella di *recoverAct* ma più elevate di qualsiasi altra azione (evitare gli ostacoli è un fatto fondamentale); priorità minima per l'azione *constantVelocityAct*, pari a 25; priorità leggermente più alta per l'azione istanza di *ArActionTrack*, pari a 30.

Il programma è stato sviluppato basandosi fortemente sul programma principale dell'esempio *secondeaction* presentato a lezione. Di tale programma, in particolare, vengono mantenute invariate le istruzioni per l'aggiunta di un robot dotato di sonar, e per l'istanziazione e l'aggiunta al robot delle *Action* che gli permettono di evitare ostacoli frontali, vicini e lontani, di recuperare da un eventuale stallo dei motori, e di muoversi con velocità costante: le azioni sono le stesse, a meno di piccole differenze nei parametri e nelle priorità. L'azione che permette di evitare ostacoli laterali, istanza di *ArActionAvoidSide*, è stata sostituita, invece con un'istanza dell'azione *ArActionTrack*, sviluppata in questo contesto.

3.2. ArActionTrack

Dopo avere capito quale sia il comportamento globale del sistema e come faccia il robot ad evitare gli ostacoli frontali, è possibile passare a descrivere i dettagli della *Action* che permette di seguire un opportuno perimetro.

3.2.1. Parametri di ingresso

Come già affermato precedentemente, l'azione *ArActionTrack* richiede cinque parametri di ingresso. Il primo di tali parametri consiste nel nome da assegnare all'istanza dell'azione; gli altri parametri, tutti di tipo *double*, sono:

- un valore, denominato *obstacleDistance*, che rappresenta la distanza dall'ostacolo da seguire, e che il robot dovrebbe mantenere;
- un parametro denominato *hyst*, che rappresenta il valore dell'isteresi relativa alla distanza da mantenere, necessaria ad evitare che il robot continui ad avvicinarsi o ad allontanarsi dall'ostacolo anche a fronte di minime variazioni della distanza misurata da esso; sostanzialmente usando tale parametro, come si potrà vedere in seguito, si fa in modo che il robot si allontani dall'ostacolo quando tale spazio scende sotto il valore di *obstacleDistance*, mentre si avvicini solo quando essa supera il valore *obstacleDistance+hyst*;
- un parametro, *turnAmount*, che specifica quale sia l'angolo di rotazione da adottare per allontanarsi o avvicinarsi ad un ostacolo; questa quantità è utilizzata anche per calcolare alcuni valori di angoli di rotazione, di ampiezza minore, da adottare in alcune situazioni specifiche (un esempio fra tutti, quando ci si rende conto di dovere descrivere i cosiddetti angoli ottusi);
- un ultimo parametro, *vel*, che indica la velocità da mantenere quando l'azione corrente è attiva.

3.2.2. Parametri rilevati dal robot

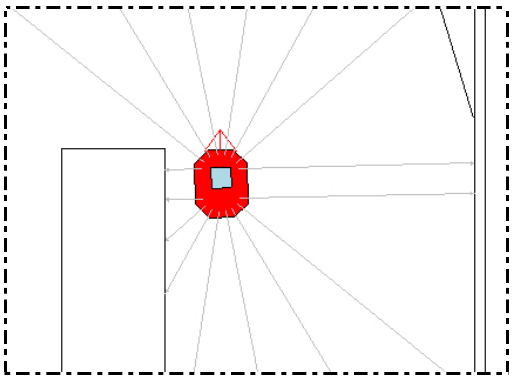
Oltre ai parametri passati dal programma, l'azione corrente si serve di alcuni valori misurati dal robot stesso, in particolare attraverso i sonar. Tali parametri, ancora una volta di tipo *double*, sono:

- *leftDist* e *rightDist*, che costituiscono una misura delle distanze da ostacoli laterali, a sinistra e a destra, rilevate dai sensori del robot; secondo la logica con cui vengono eseguite tali misurazioni, le due distanze introdotte rappresentano la distanza minima valutata entro un angolo di 30 gradi a partire dall'asse perpendicolare all'asse del robot, sia a sinistra sia a destra;
- *angSinistra* e *angDestra*, che, a dispetto del nome, misurano una distanza, ossia, più in dettaglio, le distanze degli angoli anteriori del robot da eventuali ostacoli, misura particolarmente utile a fronte di angoli ottusi; più precisamente, si misura la distanza minima da ostacoli entro un'area che va da 30 gradi a 80 gradi rispetto all'asse del robot, ancora una volta sia a sinistra sia a destra;
- *distAvanti*, ossia la distanza da ostacoli frontali, calcolata come distanza minima percepita entro un angolo di 60 gradi, che ha come bisettrice l'asse del robot.

3.2.3. Valori di soglia e perturbazioni

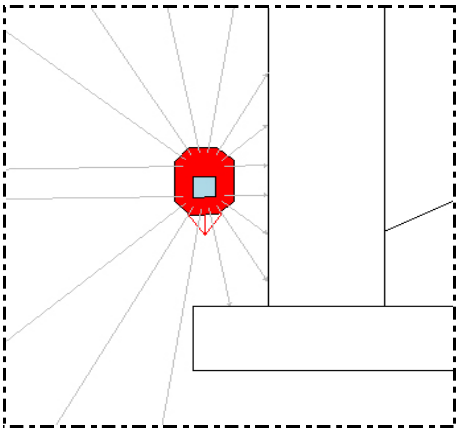
Prima di iniziare a descrivere i dettagli dell'algoritmo di funzionamento dell'azione corrente, è bene soffermarsi a discutere alcuni valori, in particolare valori di soglia e valori che rappresentano perturbazioni, sfruttati dal robot per discriminare il proprio comportamento in varie situazioni. Tali valori, calcolati empiricamente, sono stati definiti come macro all'inizio del codice dell'*Action* e sono elencati di seguito.

- *START_TURNING*: questo valore, calcolato come *2.5*obstacleDistance*, costituisce una soglia che interessa le distanze *angSinistra* e *angDestra*; in particolare, quando una delle distanze laterali (*leftDist* o *rightDist*) è conforme alla distanza cui il robot deve mantenersi (con eventuale tolleranza), mentre la distanza dell'angolo nello stesso lato è superiore al valore *START_TURNING*, significa che, entro poco spazio, il profilo dell'ostacolo non seguirà più la stessa direzione, e, quindi, il robot dovrà descrivere un angolo ottuso per seguirlo; in una tale situazione, occorreranno opportune perturbazioni nel movimento, per evitare che il robot non riesca a seguire il profilo di tale ostacolo perché trova, improvvisamente (al termine del lato dell'ostacolo corrente), ostacoli più vicini da seguire. La situazione appena descritta può essere rappresentata dalla seguente immagine.



- *START_TURNING_AMOUNT*: questo valore rappresenta l'angolo, calcolato come un decimo del parametro *turnAmount*, con cui iniziare a ruotare, nel verso dell'ostacolo, quando si verifica una situazione come quella descritta al punto precedente; si introduce questa perturbazione, a fronte di angoli ottusi, per fare in modo che il robot continui a vedere il profilo dell'ostacolo anche quando termina il lato che si è seguito fino a questo punto.
- *PREPARE_TO_TURN_AVANTI* e *PREPARE_TO_TURN_LATO*: questi due valori di soglia, rispettivamente pari a *3*obstacleDistance* e *3*obstacleDistance/2*, vengono utilizzati in un'altra situazione problematica, che si verifica quando il robot trova degli ostacoli frontali costituiti da piccole "sporgenze" (ossia, ostacoli tali da non occupare tutto l'orizzonte visto dal robot); in una tale situazione, infatti, il robot potrebbe giungere all'angolo orientato in una posizione tale da causare una rotazione errata del robot stesso (perché, ad esempio, si sta ri-avvicinando all'ostacolo dopo avere misurato una distanza troppo elevata, e, dunque, invece di seguire il profilo

dell'ostacolo, il robot descrive un angolo piatto che lo porta a ripercorrere la direzione percorsa finora, con verso opposto); il robot considera di essere in una situazione del genere quando la distanza frontale (*distAvanti*) è inferiore a *PREPARE_TO_TURN_AVANTI*, mentre la distanza nell'angolo nella parte opposta all'ostacolo che si sta seguendo è superiore a *PREPARE_TO_TURN_LATO*. In questa situazione, descritta nella figura seguente, si introduce un'opportuna perturbazione nella direzione, descritta in seguito.



- *REDUCED_TURNING_AMOUNT*: questa misura di angolo di rotazione, un quinto di *turnAmount*, costituisce la perturbazione nel movimento del robot, introdotta a fronte di situazioni come quella descritta al punto precedente; in tale situazione, si fa in modo di ruotare il robot della quantità indicata dalla parte opposta a quella in cui si trova l'ostacolo che si sta seguendo.

3.2.4. Algoritmo

Dopo avere eseguito il passaggio dei parametri dall'esterno, nonché l'esecuzione delle misure necessarie al robot, si può iniziare ad eseguire le istruzioni che descrivono il comportamento voluto. Il codice che le contiene, descritto nel paragrafo corrente, si sviluppa attraverso una struttura condizionale, anche annidata, i cui rami descrivono il comportamento che il robot deve mantenere in certe specifiche situazioni; tali situazioni sono individuate a partire dal valore delle distanze misurate dal robot, anche confrontandole con i valori di soglia descritti precedentemente. Il comportamento è descritto nei seguenti punti, ciascuno dei quali rappresenta uno specifico ramo della struttura condizionale più esterna.

3.2.4.1 Caso 1 – entrambe le distanze maggiori di quella da mantenere

La prima situazione contemplata nel codice, è quella per cui il robot misura, sia a destra sia a sinistra, distanze, dagli ostacoli, superiori a quella che si vuole mantenere dal tracciato da seguire (considerando anche il valore dell'isteresi): in una tale situazione, non si fa altro che imporre al robot di muoversi, con la velocità imposta in *vel*, ruotando con l'angolo specificato in *turnAmount*, verso la direzione laterale in cui trova l'ostacolo a distanza minore (nel caso in cui le distanze misurate da entrambi i lati siano le stesse, ossia, principalmente, quando gli ostacoli laterali sono più distanti rispetto al raggio d'azione dei sonar, il movimento è, per scelta arbitraria, verso sinistra).

3.2.4.2 Caso 2 – una sola distanza minore di quella da mantenere

Quando, invece, si ha una delle distanze laterali minore di *obstacleDistance*, sia essa destra o sinistra, mentre l'altra è, genericamente, maggiore di tale quantità, significa che, da un solo lato, si è troppo vicini all'ostacolo da seguire. Si fa in modo, quindi, di allontanarsi, su tale lato, dall'ostacolo cui si è troppo vicini, muovendosi, verso il lato opposto, con la velocità imposta da *vel*, e ruotando con un angolo pari a *turnAmount*.

3.2.4.3 Caso 3 – entrambe le distanze inferiori a quella desiderata

Un'altra possibile situazione si ha quando si misurano, da entrambi i lati, distanze inferiori rispetto a quella da mantenere: in una tale situazione, dunque, il robot si trova, temporaneamente, in un corridoio o in una strettoia, in cui non gli è possibile allontanarsi da una parte senza avvicinarsi troppo all'altra. A

fronte di questa circostanza, non si fa altro che imporre a zero l'angolo di rotazione: il robot, in tale modo, si muove con moto rettilineo, senza avvicinarsi ulteriormente né all'uno né all'altro lato, eventualmente affidandosi alle altre *Action* adottate nel main per evitare gli ostacoli.

3.2.4.4 Caso 4 – distanza sinistra corretta

Quando la distanza laterale sinistra, misurata dal robot, risulta essere "corretta", ovvero superiore a *obstacleDistance* ma inferiore alla stessa distanza sommata all'isteresi, si possono avere vari comportamenti, anche per fare fronte ai possibili problemi che si possono presentare in una simile circostanza. In particolare: se il robot non è nei pressi di un angolo ottuso (ovvero la distanza *angSinistra* è minore della soglia denominata *START_TURNING*), si valuta se lo stesso non debba prepararsi a ruotare verso destra perché si trova di fronte ad una sporgenza che gli rende difficile girare dalla parte corretta (si considera di essere in un caso di questo tipo, se *distAvanti* è minore della soglia *PREPARE_TO_TURN_AVANTI*, mentre la distanza dell'angolo anteriore destro, *angDestra*, è maggiore di *PREPARE_TO_TURN_LATO*); nel caso in cui si sia nei pressi di tale sporgenza, si impone un angolo di rotazione pari a *REDUCED_TURNING_AMOUNT*, con segno negativo (che rappresenta rotazione verso destra), altrimenti l'angolo di rotazione viene posto a zero (se si è ad una distanza corretta, senza essere vicini ad angoli ottusi o a sporgenze, si può procedere nella direzione in cui ci si sta muovendo); quando, invece, la distanza misurata nell'angolo anteriore destro è superiore alla soglia *START_TURNING* (mentre quella laterale è corretta), significa che si è nei pressi di un angolo ottuso, per descrivere correttamente il quale, si impone un angolo di rotazione pari a *START_TURNING_AMOUNT* (con segno positivo, ossia verso sinistra), introducendo una piccola perturbazione che aiuta a percorrere tale angolo, come nella sequenza di figure seguente.



3.2.4.5 Caso 5 – distanza destra corretta

Il comportamento in caso di distanza destra "corretta" (ancora una volta, entro l'intervallo che va da *obstacleDistance* a *obstacleDistance+hyst*), è esattamente duale a quello precedente: ci si muove, infatti, verso destra o sinistra, delle quantità descritte, oppure si rimane fermi, quando, rispettivamente, si è presso angoli ottusi, sporgenze, o in situazioni normali.

Si può notare che in nessuno dei rami condizionali precedenti viene trattato il caso della descrizione di angoli acuti. Per gestire questo tipo di situazioni, infatti, ci si serve delle azioni *avoidFrontNearAct* e *avoidFrontFarAct*, presenti nel programma main: quando ci si trova nei pressi di tali angoli acuti, il robot, avvicinandosi sempre di più al proseguimento dell'ostacolo che sta seguendo, fa intervenire le azioni indicate, che permettono al robot di ruotare; gli unici "aiuti" provenienti da *ArActionTrack* sono le eventuali perturbazioni, che orientano correttamente il robot (a fronte di "sporgenze").

4. Modalità operative

Capito il funzionamento completo del programma, è possibile passare a descrivere le modalità operative che permettono di eseguire il programma su un robot reale.

4.1. Componenti necessari

Il programma necessita, essenzialmente, di due componenti. In primo luogo, un robot dello stesso modello di quelli per cui è stato sviluppato, ovvero un ActivMedia Pioneer1, come i robot *Speedy* e

Tobor disponibili presso il laboratorio ARL. Tali robot devono essere dotati, inoltre, della libreria Aria, sfruttata durante lo sviluppo ed il funzionamento del programma.

4.2. Modalità di installazione

Il programma, una volta trasferito sul calcolatore del robot, per essere eseguito, non necessita di alcuna installazione, ma deve soltanto essere compilato attraverso il file *Makefile* allegato ai sorgenti del programma (all'interno della directory che lo contiene); successivamente, il programma viene avviato digitando il nome dell'eseguibile indicato. Un esempio di compilazione e di lancio del programma sono i seguenti.

```
[~/WallTracking]$ make
[~/WallTracking]$ ./WallTracking
```

L'esecuzione può essere terminata premendo il pulsante *ESC*.

4.3. Modalità di taratura

Il programma, con i parametri e le priorità assegnati alle varie azioni aggiunte, è stato pensato per funzionare al meglio in un ambiente come quello del laboratorio ARL, ossia un ambiente che, pur non essendo eccessivamente angusto, è caratterizzato da frequenti cambi di direzione dovuti alla presenza di molti ostacoli fissi, che costituiscono sporgenze, rientranze, ecc.

Per adattare il programma facendo in modo che il robot possa comportarsi in maniera corretta, o anche più efficiente, in ambienti di diverso tipo, è bene seguire i seguenti principi-guida:

- in primo luogo, limitare le modifiche apportate, al solo programma main ed ai soli parametri o priorità dell'azione istanza di *ArActionTrack* (lasciando inalterato il codice di tale classe), dal momento che le altre azioni, in particolare quelle che permettono di evitare ostacoli, sono caratterizzate da parametri che permettono di evitare collisioni del robot pur senza eccessiva "prudenza" (i.e. ci si avvicina agli ostacoli quel tanto che basta per evitarli senza urtarli);
- parlando, dunque, dei parametri dell'azione *segui*, istanza di *ArActionTrack*, e, più in particolare, della distanza da mantenere, attualmente a 130 mm, essa deve essere calibrata opportunamente sulla base delle dimensioni dell'ambiente che si sta esplorando; se ci si trova, infatti, in un ambiente sufficientemente esteso, con ostacoli piuttosto distanti gli uni dagli altri, si può anche impostare una distanza più elevata (rimanendo entro i limiti dei sonar); in ambienti più angusti (a patto che siano estesi a sufficienza, in maniera conforme alle dimensioni del robot), questa distanza può anche essere ridotta, fino ad un limite minimo di circa 100-105 mm, per evitare di entrare continuamente in conflitto con quanto imposto dall'azione *avoidFrontNearAct* (tale azione, infatti allontana il robot se la distanza nell'angolo anteriore è di circa 90 mm, e dunque scendendo sotto i 100-105 mm laterali, le due azioni potrebbero interferire l'una con l'altra);
- l'isteresi va modificata sia in base alla precisione che si vuole pretendere dal robot nel mantenimento della distanza imposta (più l'isteresi è bassa più il robot è preciso), ma anche in base alle dimensioni dell'ambiente che si esplora, visto che, essenzialmente, l'isteresi aumenta la distanza che può essere mantenuta dal tracciato da seguire (i.e. se l'ambiente è troppo stretto, un'isteresi eccessiva potrebbe portare il robot a sviare dalla propria traiettoria); è bene ricordare, comunque, che l'isteresi, attualmente impostata a 30 mm, non deve essere nemmeno pressoché nulla, dato che ciò causerebbe continue oscillazioni del robot nell'intorno della distanza imposta;
- per ciò che riguarda l'angolo con cui avvengono le rotazioni, attualmente impostato a 10 gradi, è bene cercare di limitarne l'entità, per evitare di avvicinarsi o allontanarsi troppo dagli ostacoli che si stanno seguendo, né diminuirla eccessivamente, per permettere il recupero di una distanza desiderata in tempi ragionevoli (in sostanza, è bene rimanere entro un intervallo che va dai 5 ai 15-20 gradi);
- la velocità, infine, non deve essere eccessivamente elevata, per permettere al robot di eseguire misurazioni, e movimenti, quanto più precisi possibile; attualmente la velocità impostata è di circa 50 mm/s, adeguata ad ambienti piuttosto ristretti e tortuosi; in ambienti più regolari e ampi tale velocità può anche essere aumentata, fino ad un massimo di circa 150-200 mm/s.

In ogni caso, è consigliabile adottare il seguente approccio: usare, almeno inizialmente, i parametri minimi, magari anche quelli già impostati nel programma, che permettono al robot di comportarsi correttamente anche negli ambienti più angusti (sempre conformemente alle sue dimensioni); successivamente, provare a modificare opportunamente i vari parametri, seguendo le indicazioni fornite precedentemente, anche sulla base dell'ambiente a propria disposizione, per ottenere un comportamento più efficiente.

4.4. Avvertenze

Pur essendo stato sviluppato per i robot ActivMedia Pioneer1, *WallTracking* può girare anche su altri tipi di robot, purché dotati di ruote, sonar anteriori, e della libreria Aria, come, ad esempio, il robot ActivMedia Pioneer3, *Morgul*, disponibile presso il laboratorio ARL. Prima di provare il programma su tale robot, tuttavia, è bene modificare opportunamente il parametro della distanza da mantenere rispetto al perimetro da seguire: tale parametro va aumentato adeguatamente, tenendo conto dello spessore delle ruote, molto maggiore rispetto a quello delle ruote di *Speedy* e *Tobor*.

5. Conclusioni e sviluppi futuri

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

Il robot è in grado di esplorare un ambiente delimitato da ostacoli di vario tipo, a meno di situazioni particolari, in cui le misurazioni dei sonar sui lati destro e sinistro sono simili, e pertanto si possono avere risultati inaspettati.

Il progetto può essere ulteriormente sviluppato prevedendo la possibilità di fornire alcuni parametri al sistema da linea di comando, che vadano a sostituire quelli di default. Questo non è stato fatto perché avrebbe reso necessaria una modellizzazione accurata del problema, invece di risolverlo in maniera empirica come in questo caso. In pratica, si tratterebbe di estendere il software con un meccanismo di auto-taratura che, sulla base di velocità, distanza e tolleranza fissate, calcoli automaticamente i valori degli angoli di curvatura e delle varie condizioni che determinano i comportamenti nella versione attuale.

Un altro possibile sviluppo potrebbe essere lo sfruttamento di quanto qui prodotto per un'attività di perlustrazione e pattugliamento, simile a quanto già fa *Morgul*, utilizzando la telecamera (peraltro già installata su *Speedy*). Per fare ciò è necessario affiancare l'attività di perlustrazione del software qui descritto, con l'attività della telecamera, generando semplicemente un ulteriore programma che faccia da "collante" tra i due, istanziandoli ed avviandoli contemporaneamente.

Bibliografia

[1] MobileRobots: "ARIA Developer's Reference Manual"

Indice