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)

LabCleaner

Elaborato di esame di: **Fabio Bresciani, Giacomo Carlotti, Nicola Gelfi, Fabio Mostarda**

Consegnato il: **27 Giugno 2005**

Sommario

Per realizzare il progetto LabCleaner abbiamo sviluppato, utilizzando le librerie Aria, un programma che facesse vagare il robot nell'ambiente in maniera casuale, evitando eventuali ostacoli a portata di sonar, con le pinze aperte: nel caso in cui un oggetto fosse stato rilevato dalle fotocellule delle pinze, il robot avrebbe dovuto prelevarlo e riportarlo alla posizione di partenza. Una volta depositato l'oggetto, la macchina doveva riprendere la sua ricerca casuale dal punto in cui l'aveva interrotta. Per ottenere questi risultati abbiamo scomposto il comportamento complesso desiderato in comportamenti "elementari" che abbiamo sviluppato singolarmente. Successivamente abbiamo composto i comportamenti con l'ausilio di parti algoritmiche. Infine abbiamo testato l'applicazione integrata e abbiamo stilato la documentazione.

1. Introduzione

Per comprendere meglio il lavoro svolto è utile introdurre il robot utilizzato per l'elaborato, al fine di intendere meglio le difficoltà incontrate e le motivazioni delle soluzioni adottate.

1.1. Il robot Tobor

Il robot utilizzato nell'elaborato è un modello Pioneer 1 della Activmedia, denominato Tobor.

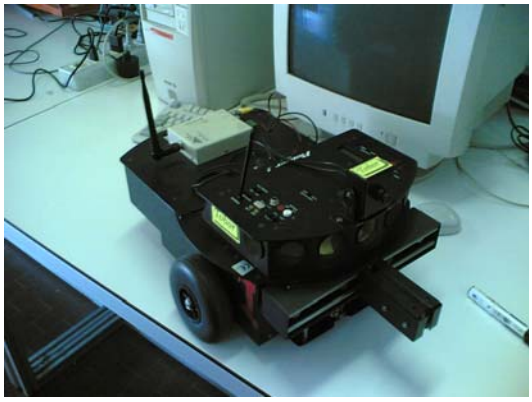

Figura 1.1: Tobor

Dal punto di vista sensoristico, Tobor dispone di 5 sonar frontali e 2 laterali; non è dotato di giroscopio.

Il suo organo di presa è una pinza dotata di bumpers frontali; all'interno delle ganasce vi sono due fotocellule, in grado di determinare la presenza di un oggetto. La pinza ha la possibilità di sollevare ciò che afferra, ma questo movimento è integrato nel cinematismo di presa degli oggetti. Per vincoli costruttivi, quando la pinza è chiusa le sue ganasce si trovano in posizione di elevazione massima; per aprirle, viene effettuato un movimento prima in verticale verso il basso, poi le ganasce vengono separate fino a raggiungere il fine corsa.

Gli oggetti sollevabili dalla pinza sono dell'ordine di una pallina di carta: l'organo meccanico non può alzare un gran peso, ma è in compenso molto sensibile e può afferrare oggetti fragili o molto elastici senza esercitare forze in grado di deformarli.

Tobor dispone inoltre di una videocamera, ed è controllato mediante radio modem.

2. Il problema affrontato

Il problema da risolvere è un particolare esempio di applicazione di navigazione per robot: era infatti richiesta la scrittura di un programma che facesse muovere il robot Tobor e ne sfruttasse la pinza per raccogliere piccoli oggetti dal terreno, per poi portarli in una posizione predefinita. Il progetto, denominato espressivamente "LabCleaner", avrebbe quindi portato alla creazione di un "robot spazzino", un pulitore autonomo di ambienti in grado di occuparsi di comuni rifiuti d'ufficio (come, ad esempio, palline di carta).

2.1. Le specifiche

Più precisamente, le specifiche imponevano i seguenti vincoli:

- Possibilità di sfruttare i programmi e le librerie esistenti.
- Movimento del robot con la pinza aperta.
- Movimento più o meno casuale nell'ambiente per ricercare oggetti.
- Ogni volta che un oggetto viene rilevato tra le pinze, deve essere afferrato, sollevato e portato in una posizione predefinita.
- Una volta riportato un oggetto, il robot deve tornare nella posizione in cui ha interrotto la ricerca.

La posizione predefinita per depositare gli oggetti è stata scelta prendendo per convenzione la posizione di partenza del robot.

2.2. Le difficoltà da affrontare

Il soddisfacimento delle specifiche avrebbe necessariamente implicato il superamento delle seguenti difficoltà:

- Gestione della navigazione del robot.
- Gestione del movimento in presenza di ostacoli.
- Gestione della pinza.
- Gestione della posizione del robot.

Bisogna inoltre considerare l'impatto sul progetto causato dall'introduzione delle versioni aggiornate delle librerie Aria: a fronte di nuove potenzialità, ci si è trovati ad affrontare una situazione di carenza di documentazione. La completa sezione informativa allegata alle librerie copriva infatti il significato di ogni funzione, ma mancava di una sezione di introduzione che proponesse un approccio organico alla programmazione usando Aria. L'assenza di un simile manuale introduttivo ha complicato la ricerca e l'utilizzo delle funzioni corrette per svolgere i compiti desiderati.

Infine va considerato anche il fatto che non è stato possibile utilizzare il simulatore (MobileSim) poiché in primo luogo non incorporava la gestione della pinza e in secondo luogo forniva risultati con grosse discrepanze rispetto alla realtà.

3. La soluzione adottata

Il progetto si è articolato in quattro fasi principali. In primo luogo è stato necessario imparare a gestire la pinza di Tobor. In parallelo si è studiato il problema della navigazione. Risolti questi due primi punti si è passati alla realizzazione dei comportamenti, ed infine all'integrazione nell'applicazione finale.

3.1. La gestione della pinza

Abbiamo verificato che le primitive messe a disposizione dalla classe ArGripper della libreria Aria non funzionavano correttamente a causa della rimappatura dei bit degli ingressi digitali su Tobor. Abbiamo sviluppato la classe ToborsGripper che controlla la pinza, permettendo le seguenti operazioni:

- Apertura della pinza
- Chiusura della pinza
- Rilevazione del movimento della pinza
- Arresto del movimento della pinza
- Rilevazione della presenza di un oggetto tra le pinze
- Rilevazione della pressione sui bumpers posizionati alle estremità della pinza

Per verificare il corretto funzionamento di questa classe abbiamo creato un programma di test (gripperDemo) che permette di controllare da tastiera le operazioni della pinza e di stampare a video il relativo byte di stato.

3.2. La navigazione

Per iniziare ad affrontare il problema della navigazione siamo partiti dal programma d'esempio wander (fornito nella directory examples della libreria Aria). Per poterlo compilare in una qualsiasi directory abbiamo creato un semplice script di shell (mymake.sh, che abbiamo reso disponibile sul sito web del gruppo ARLStudents). Wander offre già i comportamenti necessari per effettuare una navigazione casuale (con obstacle avoidance) all'interno di un ambiente. Come primo passo abbiamo cercato un modo per far tornare il robot alla posizione di partenza, simulando il fatto di aver raccolto un oggetto trascorso un certo intervallo di tempo. Abbiamo ottenuto questo comportamento utilizzando la classe ArActionGoto della libreria Aria: quando, tramite il metodo setGoal di questa classe, viene fissato l'obiettivo che il robot deve raggiungere, il comportamento si attiva e la macchina cessa di vagare casualmente convergendo al punto specificato. Le coordinate dell'obiettivo vengono passate al metodo setGoal incapsulate in un oggetto di tipo ArPose (classe della libreria Aria) che contiene le coordinate X e Y e l'azimut θ rispetto a un sistema di riferimento la cui origine è il punto di partenza del robot. Il calcolo della posizione attuale e della traiettoria da percorrere per raggiungere l'obiettivo viene effettuato esclusivamente tramite misure odometriche.

Le specifiche richiedevano inoltre che, una volta tornato alla posizione di partenza, tornasse dove aveva trovato e raccolto l'oggetto. Per realizzare questo comportamento è bastato fissare come obiettivo da raggiungere la posizione dalla quale il robot aveva virtualmente (per ora!) prelevato il pezzo. Questa era stata salvata in un oggetto ArPose tramite il metodo getPose della classe ArRobot appena prima di aver ordinato alla macchina di tornare alla posizione iniziale.

Durante i test abbiamo verificato la presenza di consistenti (e inevitabili) errori odometrici, che talvolta causavano dei comportamenti non desiderabili del robot nel raggiungimento degli obiettivi fissati. Per diminuirne l'incidenza abbiamo fissato una tolleranza più lasca rispetto a quella di default (tramite la funzione setCloseDist della classe ArActionGoto) nel test di avvenuto raggiungimento degli obiettivi, realizzato dalla funzione haveAchievedGoal della classe ArActionGoto.

3.3. I comportamenti

I comportamenti sono gestiti in Aria da un resolver, che decide quale behavior attivare in base alla priorità ad esso associata. Per aggiungere un nuovo behavior alla lista dei comportamenti del robot si sfrutta il metodo addAction (classe ArRobot), che richiede come parametri, oltre al behavior stesso, anche la sua priorità. Nella libreria Aria i comportamenti sono delle classi che estendono la classe astratta ArAction, sovrascrivendone alcuni metodi. Obbligatoriamente vanno sovrascritti i metodi setRobot e fire; il primo è semplicemente un metodo di servizio per fornire al comportamento una reference al robot, mentre il secondo è il metodo principale che contiene le azioni da compiere ogniqualvolta il resolver attiva il behavior.

Abbiamo tenuto come base il programma d'esempio wander, che conteneva già alcuni comportamenti necessari alla navigazione causale con obstacle avoidance. In seguito abbiamo aggiunto i nuovi comportamenti per completare le funzionalità richieste.

3.3.1. ArAction delle librerie Aria utilizzate nel progetto

Abbiamo utilizzato all'interno del progetto le ArAction StallRecover, AvoidFront, ConstantVelocity e Goto.

- StallRecover: tenta una serie di contromisure per uscire dallo stallo se una o più ruote si bloccano.
- AvoidFront: implementa una politica di obstacle avoidance; in particolare abbiamo definito due varianti, AvoidFrontFar e AvoidFrontNear. La prima permette di evitare gli ostacoli più lontani mantenendo i parametri di default della classe ArActionAvoidFront; la seconda è stata configurata con opportuni parametri per avvicinarsi il più possibile agli ostacoli e alle pareti per poter eventualmente rilevare la presenza di oggetti da raccogliere.
- ConstantVelocity: fa procedere il robot in avanti alla velocità costante specificata. Abbiamo scelto una velocità sufficientemente bassa per evitare violente collisioni contro ostacoli non rilevabili dai sonar (200 mm/sec).
- Goto: il comportamento che fa raggiungere al robot la posizione fissata con il metodo setGoal (vedere paragrafo 3.2). Il comportamento, anche se aggiunto alla lista di behavior del robot, rimane inattivo fino a che non viene fissato un obiettivo.

3.3.2. ArAction implementate ex novo

Abbiamo creato ex novo le seguenti ArAction:

- GripperCatch: se la pinza non è in movimento e viene rilevato un ostacolo tra le ganasce il robot viene fermato e la pinza viene chiusa, raccogliendo quindi l'oggetto.
- GripperOpen: se la pinza è chiusa e non ci sono oggetti tra le ganasce la pinza viene aperta. Quest'azione serve per aprire la pinza all'inizio del lavoro e ogniqualvolta il robot non riesce a raccogliere l'oggetto oppure lo perde durante il tragitto.
- GripperMovingStop: se la pinza è in movimento, mantiene il robot fermo. Questo comportamento ha permesso di risolvere il problema che avevamo riscontrato tentando di mantenere fermo il robot durante il processo di raccolta dell'oggetto. Inizialmente avevamo implementato questo controllo con un ciclo while che terminava quando la pinza era chiusa e non più in movimento. Questo ciclo non terminava mai in quanto il byte di stato della pinza non veniva mai liberato e lo StateReflector non poteva aggiornarne il valore. Utilizzando un behavior separato si demanda al resolver il dispatching tra i comportamenti e lo StateReflector. Abbiamo scelto di mantenere il robot fermo finché la pinza non viene completamente sollevata. Nel caso in cui un oggetto si incastri sotto la pinza, e quindi questa rimanga in uno stato non direttamente rilevabile (non completamente aperta né chiusa) oltre un certo timeout (GRIPPER_STUCK_TIMEOUT, pari di default a 5 secondi), forziamo la chiusura della pinza; in caso contrario il robot penserebbe sempre di avere la pinza in movimento e rimarrebbe immobile. In seguito gli altri behavior provvederanno a riaprire la pinza e a far allontanare il robot, evitando eventuali "deadlock".

- GripperBumper: se il robot urta un ostacolo con i bumpers (posizionati alle estremità della pinza) viene arrestato.

3.4. Integrazione

Una volta creati e testati singolarmente i comportamenti siamo passati all'integrazione degli stessi nell'applicazione finale, organizzandoli in una gerarchia di priorità e combinandoli attraverso una parte prettamente algoritmica.

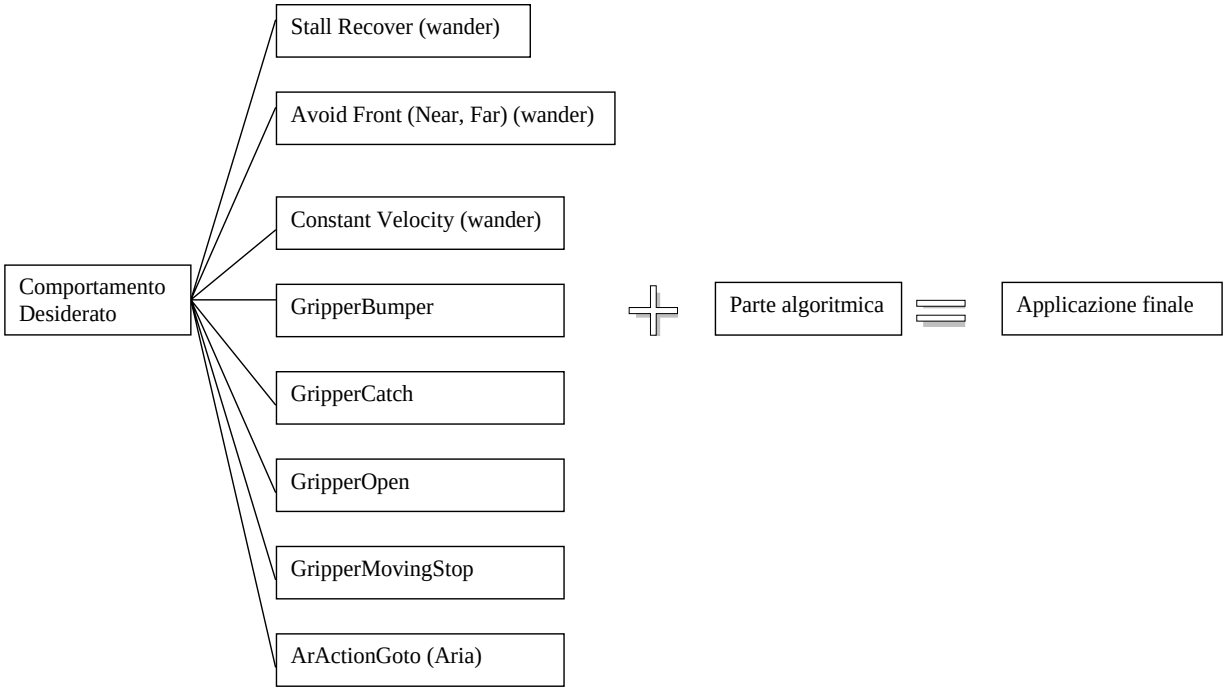

Figura 3.1: Integrazione comportamenti e parte algoritmica

3.4.1. Priorità dei behavior

Al vertice della gerarchia troviamo il comportamento di reazione all'attivazione dei bumpers, che deve prelazionare tutti gli altri in quanto critico per la sicurezza del robot. Immediatamente al di sotto abbiamo posizionato il comportamento di StallRecover, e in seguito tutti i comportamenti relativi ai movimenti della pinza. In particolare se il robot sta raccogliendo un oggetto e quindi ha la pinza in movimento è necessario che stia fermo. Inoltre la macchina deve sempre mantenere la pinza aperta a meno che non stia trasportando un oggetto. Infine se viene rilevato qualcosa tra le ganasce il robot deve chiudere la pinza e raccogliere l'oggetto. Con priorità inferiore seguono i comportamenti legati alla navigazione, ovvero la obstacle avoidance (AvoidFrontNear e AvoidFrontFar), il raggiungimento di obiettivi (Goto) e il mantenimento della velocità di crociera (ConstantVelocity).

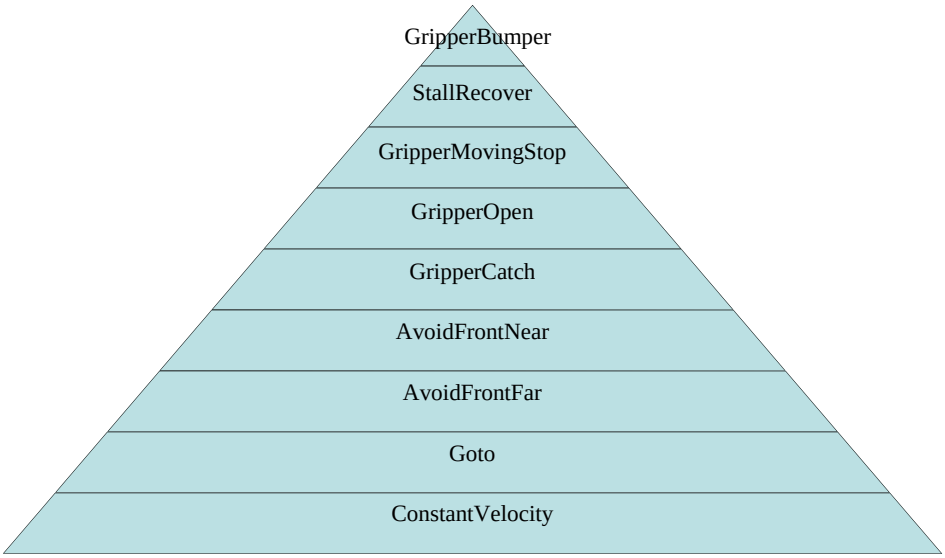

Figura 3.2: Gerarchia delle priorità dei behavior

3.4.2. La parte algoritmica

L'algoritmo, inserito nella funzione main dell'applicazione, è costituito da un loop infinito, all'interno del quale vengono eseguiti svariati controlli opportunamente temporizzati (tramite la funzione Sleep della classe ArUtil della libreria Aria) per dar modo allo StateReflector di aggiornare i byte di stato del robot. Prima del codice rappresentante la parte algoritmica è necessario chiamare il metodo runAsync della classe ArRobot (chiamata asincrona che attiva il robot), e dopo la chiusura del loop è buona regola terminare l'applicazione invocando nell'ordine i metodi stopRunning e waitForRunExit della classe ArRobot e infine Aria::shutdown.

La catena di controlli è organizzata come segue:

1) se il robot rileva tra le pinze e quindi raccoglie un oggetto
    a. memorizza la posizione in cui è stato trovato l'oggetto
    b. fissa come obiettivo da raggiungere la posizione iniziale
    c. se durante il tragitto perde l'oggetto riapre le pinze
2) una volta raggiunta la posizione iniziale il robot
    a. rilascia l'oggetto aprendo la pinza
    b. disattiva temporaneamente il comportamento di raccogliere oggetti qualora rilevati, altrimenti raccoglierebbe nuovamente l'oggetto appena depositato
    c. si muove all'indietro di 30 cm
    d. riattiva il comportamento di raccogliere oggetti
    e. fissa come obiettivo da raggiungere la posizione dove aveva raccolto l'oggetto (memorizzata al punto 1a).
3) se durante il tragitto verso la posizione dove aveva trovato l'ultimo oggetto ne trova un altro, esegue nuovamente l'algoritmo da capo.
4) se in qualsiasi momento durante l'esecuzione dei punti precedenti i bumpers vengono sollecitati da un ostacolo, viene eseguita una procedura di recover algoritmica da noi realizzata (diversa dal behavior StallRecover comunque sempre attivo).

La procedura di recover è una funzione che

- diminuisce la velocità del robot a 50 mm/sec
- lo fa arretrare di 20 cm
- lo fa girare di 90° in senso orario o antiorario; la scelta del senso di rotazione è randomizzata per aumentare la probabilità che il robot possa uscire da situazioni di impasse.

3.5. Modifiche fisiche al robot

Al fine di evitare che eventuali oggetti sul terreno potessero incastrarsi sotto alle ruote di Tobor, si è pensato di apportare alle struttura fisica del robot una lieve modifica: due semplici “cunei deviatori”, due parallelepipedi cavi a base triangolare, ottenuti mediante deformazione di una lamina plastica. La base triangolare delle appendici deviatrici ha dimensione 3x4x5 cm.

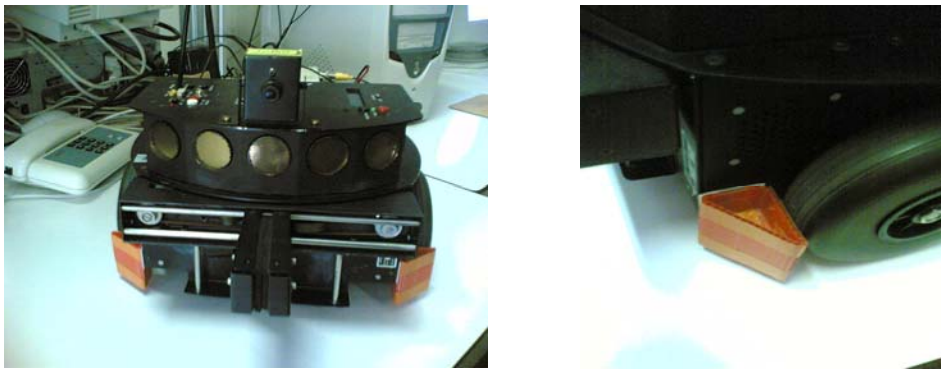

Figura 3.3: Alette deviatrici

Per aumentarne la resistenza alla deformazione, all’interno di ogni cuneo è stata inserita trasversalmente una lamina triangolare, opportunamente fissata con supporti adesivi.

I test effettuati hanno evidenziato un buon funzionamento delle appendici, in grado effettivamente di deviare verso l’esterno gli ostacoli incontrati.

4. Modalità operative

Per raggiungere l'operatività con il pacchetto LabCleaner è necessario compiere alcuni semplici passi: l'installazione delle librerie richieste (Aria, compilatore GCC) e la compilazione del codice sorgente del programma. Una volta eseguiti questi passi, il lancio dell'eseguibile sarà l'unica interazione con l'utente richiesta dal programma; questo potrà poi essere terminato semplicemente premendo il tasto ESC del calcolatore.

4.1. Componenti necessari

Per far funzionare il software LabCleaner è necessario disporre di:

- Un robot mobile dotato di pinza; la pinza deve essere dotata di opportuni sensori per rilevarne lo stato e di segnalare la presenza di un oggetto all'interno delle ganasce. Il software è stato sviluppato per robot di tipo Activmedia Pioneer 1.
- Librerie Aria 2.3 release 3, scaricabili dal sito www.activmedia.com.
- GCC 3.4.1 per la compilazione del software.

4.2. Modalità di installazione

Per l'installazione delle librerie Aria si rimanda all'apposita guida; per l'installazione del software LabCleaner è semplicemente necessario copiare i file nella cartella desiderata. Per far funzionare il programma è necessaria la sua compilazione, per la quale è possibile sfruttare uno script incluso nel pacchetto LabCleaner; è comunque possibile compilare manualmente sfruttando GCC.

Per estrarre l'archivio, digitare nella shell ($ indica il prompt):

$ mkdir LabCleanerProject

$ unzip LabCleaner.zip –d ./LabCleanerProject

Per compilare GripperDemo, dalla directory in cui si è estratto l'archivio digitare:

$ cd GripperDemo

$ chmod +x makeGripperDemo.sh

$ ./makeGripperDemo.sh

Verrà quindi prodotto il file eseguibile GripperDemo. Per lanciarlo è sufficiente digitare:

$ ./GripperDemo

A questo punto per comandare la pinza del robot è sufficiente digitare una lettera da tastiera seguita dalla pressione del tasto INVIO. In particolare, è possibile:

- Premere 'a' per aprire la pinza
- Premere 'c' per chiudere la pinza
- Premere 's' per fermare il movimento della pinza
- Premere 't' per visualizzare lo stato della pinza, delle fotocellule e dei bumpers
- Premere 'x' per uscire dall'applicazione

Per compilare LabCleaner, dalla directory in cui si è estratto l'archivio digitare:

$ cd LabCleaner

$ chmod +x makeLabCleaner.sh

$ ./makeLabCleaner.sh

oppure

$ ./makeLabCleaner.sh debug

Verrà quindi prodotto il file eseguibile LabCleaner o LabCleaner-debug. La differenza tra i due eseguibili è che il secondo mostra a video una serie di messaggi aggiuntivi per segnalare quali comportamenti vengono attivati e quali azioni algoritmiche vengono intraprese.

Per lanciarlo è sufficiente fare:

$ ./LabCleaner

4.3. Avvertenze

- **E' importante sincerarsi della presenza delle corrette versioni delle librerie Aria e del compilatore GCC: questi componenti potrebbero infatti portare a complicazioni qualora non fossero della versione indicata.**

- **Durante l'utilizzo del programma, è necessario fare attenzione che il robot non entri in ambienti con fili posti sul terreno: questo sia al fine di evitare danni al robot (le cui ruote potrebbero impigliarsi) sia danni a persone o oggetti (ad esempio, potrebbero venire strappati i fili di alimentazione di importanti macchinari o elaboratori). Si sconsiglia inoltre di lasciare alla portata del robot oggetti fragili di grande valore.**

5. Conclusioni e sviluppi futuri

Il progetto LabCleaner ha portato alla realizzazione di software in grado di far muovere il robot in un ambiente con ostacoli al fine di raccogliere piccoli oggetti sul terreno. Utilizzando sonar e odometria, sono stati raggiunti dei buoni risultati. Sviluppi futuri potrebbero riguardare l'utilizzo di sistemi di visione per:

- delimitare le aree di lavoro del robot, ad esempio attraverso degli opportuni marker (cromatici, luminosi, ecc…) per indicare aree di deposito oppure zone off-limits
- riconoscere gli oggetti da raccogliere, ed eventualmente discriminare ostacoli non rilevabili dai sonar. Tobor è del resto dotato di videocamera.

Un grosso passo avanti potrebbe essere compiuto sfruttando sistemi di misura della posizione più precisi dell'odometria; questo si scontra però con le limitazioni imposte dai sensori del robot.

Bibliografia

[1] ActivMedia Robotics, LLC: “ARIA (Activmedia Robotics Interface for Application) Reference Manual”, 2002.

[2] Archivio della mailing list ARIA-users (http://robots.activmedia.com/archives/aria-users/threads.html).

[3] ActivMedia Incorporated, “Pioneer 1 Gripper & Experimenter’s Module Manual”, 1997.

Indice